PROGRAMACIÓN EN

SYSTEM RPL

CON DEBUG 4X

Versión 4.03

Eduardo M Kalinowski

Carsten Dominik

César Vásquez Alvarado  
Lima Perú

Abril de 2019



Contenido

[Capítulo 1 Introducción 13](#_Toc5231380)

[1.1 Tu primer programa en System RPL 16](#_Toc5231381)

[1.2 Sobre la lista de entradas 17](#_Toc5231382)

[Capítulo 2 Enteros Binarios (BINTS) 20](#_Toc5231383)

[2.1 Referencia 22](#_Toc5231384)

[Capítulo 3 Números Reales 35](#_Toc5231385)

[3.1 Referencia 36](#_Toc5231386)

[Capítulo 4 Números complejos 43](#_Toc5231387)

[4.1 Referencia 44](#_Toc5231388)

[Capítulo 5 Enteros (ZINTS) 46](#_Toc5231389)

[Capítulo 6 Caracteres y Cadenas 47](#_Toc5231390)

[6.1 Referencia 48](#_Toc5231391)

[6.2 Ejemplos 63](#_Toc5231392)

[Capítulo 7 Cadenas Hexadecimales (HXS) 67](#_Toc5231393)

[7.1 Referencia 68](#_Toc5231394)

[Capítulo 8 Identificadores 71](#_Toc5231395)

[Capítulo 9 Objetos Etiquetados (Tags) 72](#_Toc5231396)

[9.1 Referencia 73](#_Toc5231397)

[Capítulo 10 Arreglos (Arrays) 74](#_Toc5231398)

[10.1 Referencia 77](#_Toc5231399)

[10.2 Ejemplos 87](#_Toc5231400)

[Capítulo 11 Objetos Compuestos 108](#_Toc5231401)

[11.1 Referencia 110](#_Toc5231402)

[11.2 Ejemplos 115](#_Toc5231403)

[Capítulo 12 Objetos Meta 129](#_Toc5231404)

[12.1 Referencia 130](#_Toc5231405)

[12.2 Ejemplos 133](#_Toc5231406)

[Capítulo 13 Objetos Unidad 137](#_Toc5231407)

[13.1 Referencia 138](#_Toc5231408)

[13.2 Ejemplos 142](#_Toc5231409)

[Capítulo 14 Objetos Simbólicos (SYMB) 144](#_Toc5231410)

[14.1 Referencia 147](#_Toc5231411)

[Capítulo 15 Objetos Gráficos (Grobs) 150](#_Toc5231412)

[15.1 Referencia 151](#_Toc5231413)

[15.2 Ejemplos 164](#_Toc5231414)

[Capítulo 16 Bibliotecas y Objetos de Respaldo 167](#_Toc5231415)

[16.1 Referencia 168](#_Toc5231416)

[16.2 Ejemplos 173](#_Toc5231417)

[Capítulo 17 Datos de Biblioteca (Library Data) 175](#_Toc5231418)

[17.1 Ejemplos 176](#_Toc5231419)

[Capítulo 18 Operaciones con la Pila 180](#_Toc5231420)

[18.1 Referencia 181](#_Toc5231421)

[Capítulo 19 Entornos Temporales 185](#_Toc5231422)

[19.1 Variables locales con nombre 185](#_Toc5231423)

[19.2 Variables locales sin nombre 186](#_Toc5231424)

[19.3 Entornos Temporales Anidados 186](#_Toc5231425)

[19.4 Otras Formas de Crear LAMS 188](#_Toc5231426)

[19.5 Viendo las Variables Locales con Debug4x 189](#_Toc5231427)

[19.6 Referencia 190](#_Toc5231428)

[Capítulo 20 Control del Runstream 195](#_Toc5231429)

[20.1 Algunos Conceptos 195](#_Toc5231430)

[20.2 Comandos Runstream 197](#_Toc5231431)

[20.3 Usos del comando COLA 199](#_Toc5231432)

[20.4 Viendo la Pila de Retornos con Debug4x 201](#_Toc5231433)

[20.5 Referencia 202](#_Toc5231434)

[20.6 Ejemplos 207](#_Toc5231435)

[Capítulo 21 Condicionales 214](#_Toc5231436)

[21.1 Tests 214](#_Toc5231437)

[21.2 If. . . Then. . . Else 215](#_Toc5231438)

[21.3 Case 215](#_Toc5231439)

[21.4 Referencia 218](#_Toc5231440)

[Capítulo 22 Bucles (Loops) 233](#_Toc5231441)

[22.1 Bucles Indefinidos 233](#_Toc5231442)

[22.2 Bucles Definidos 235](#_Toc5231443)

[22.3 Referencia 237](#_Toc5231444)

[22.4 Ejemplos 239](#_Toc5231445)

[Capítulo 23 Manejo de Errores 247](#_Toc5231446)

[23.1 Atrapando Errores 247](#_Toc5231447)

[23.2 Generando Errores 249](#_Toc5231448)

[23.3 Referencia 250](#_Toc5231449)

[Capítulo 24 La Pila Virtual (Virtual Stack) 254](#_Toc5231450)

[24.1 Viendo la Pila Virtual con Debug4x 255](#_Toc5231451)

[24.2 Referencia 256](#_Toc5231452)

[24.3 Ejemplos 259](#_Toc5231453)

[Capítulo 25 Operaciones con la Memoria 260](#_Toc5231454)

[25.1 Referencia 261](#_Toc5231455)

[Capítulo 26 Tiempo y Alarmas 274](#_Toc5231456)

[26.1 Referencia 274](#_Toc5231457)

[Capítulo 27 Funciones de Sistema 277](#_Toc5231458)

[27.1 Referencia 277](#_Toc5231459)

[Capítulo 28 Comunicación 282](#_Toc5231460)

[28.1 Referencia 282](#_Toc5231461)

[Capítulo 29 EL FILER 285](#_Toc5231462)

[29.1 Usando el Filer 285](#_Toc5231463)

[29.1.1 El Argumento Filer\_Tipos 286](#_Toc5231464)

[29.1.2 El Argumento Filer\_RutaInicial 287](#_Toc5231465)

[29.1.3 El Argumento Filer\_Comportamiento 287](#_Toc5231466)

[29.1.4 Atajos de tecla predefinidos 293](#_Toc5231467)

[29.2 El Comando FLASHPTR BrowseMem.1 296](#_Toc5231468)

[29.3 Referencia 298](#_Toc5231469)

[29.4 Ejemplos 299](#_Toc5231470)

[Capítulo 30 Verificación de Argumentos 305](#_Toc5231471)

[30.1 Número de Argumentos 305](#_Toc5231472)

[30.2 Tipo de Argumento 308](#_Toc5231473)

[30.2.1 Comando TYPE 310](#_Toc5231474)

[30.3 Referencia 312](#_Toc5231475)

[30.3.1 Verificando el Tipo de un Objeto 313](#_Toc5231476)

[30.4 Ejemplos con Verificación de Argumentos 318](#_Toc5231477)

[Capítulo 31 Control del Teclado 323](#_Toc5231478)

[31.1 Localización de Teclas 324](#_Toc5231479)

[31.2 Planos Shift Hold 325](#_Toc5231480)

[31.3 Esperando una tecla 326](#_Toc5231481)

[31.4 Referencia 328](#_Toc5231482)

[31.3.1 Conversión de Códigos de Tecla 328](#_Toc5231483)

[31.3.2 Esperando Teclas 328](#_Toc5231484)

[31.3.3 El flag ATTN 329](#_Toc5231485)

[31.3.4 Bad Keys 330](#_Toc5231486)

[31.3.5 Teclado de Usuario 330](#_Toc5231487)

[31.5 Ejemplos Control de Teclado 331](#_Toc5231488)

[Capítulo 32 Usando InputLine 332](#_Toc5231489)

[32.1 Asignaciones de las Teclas de Menu 333](#_Toc5231490)

[32.2 Los LAMS de InputLine 333](#_Toc5231491)

[32.3 Referencia 334](#_Toc5231492)

[32.4 Ejemplos InputLine 335](#_Toc5231493)

[Capítulo 33 El Bucle Externo Parametrizado (POL) 341](#_Toc5231494)

[33.1 Comandos del POL 342](#_Toc5231495)

[33.2 La Pantalla 343](#_Toc5231496)

[33.3 Asignaciones de Teclas 343](#_Toc5231497)

[33.4 Asignaciones de Teclas de Menú 344](#_Toc5231498)

[33.5 Previniendo Entornos Suspendidos 345](#_Toc5231499)

[33.6 La Condición de Salida 345](#_Toc5231500)

[33.7 Manejador de Errores 345](#_Toc5231501)

[33.8 Referencia 346](#_Toc5231502)

[33.8.1 POL 346](#_Toc5231503)

[33.8.2 Argumentos del POL 346](#_Toc5231504)

[33.9 Ejemplos 347](#_Toc5231505)

[Capítulo 34 Browser 49 351](#_Toc5231506)

[34.1 El Meta de Ítems 351](#_Toc5231507)

[34.2 La Cadena del Título 351](#_Toc5231508)

[34.3 El Ítem Inicial 351](#_Toc5231509)

[34.4 El Message Handler 352](#_Toc5231510)

[34.4.1 Message Handler número 1: MsgDispBox 352](#_Toc5231511)

[34.4.2 Message Handler número 2: MsgDispTitle 352](#_Toc5231512)

[34.4.3 Message Handler número 3: MsgEndInit 354](#_Toc5231513)

[34.4.4 Message Handler número 4: MsgKeyPress 354](#_Toc5231514)

[34.4.5 Message Handler número 5: MsgMenu 354](#_Toc5231515)

[34.4.6 Message Handler número 6: MsgEndEndDisp 354](#_Toc5231516)

[34.5 Los LAMs del Browser 49 355](#_Toc5231517)

[34.6 Accediendo al Ítem Seleccionado 356](#_Toc5231518)

[34.7 Guardando y Restaurando la Pantalla 357](#_Toc5231519)

[34.8 Referencia 358](#_Toc5231520)

[34.9 Ejemplos 361](#_Toc5231521)

[Capítulo 35 Browser 48 368](#_Toc5231522)

[35.1 El Parámetro MessageHandler 368](#_Toc5231523)

[35.2 El Parámetro $Título 373](#_Toc5231524)

[35.3 El Parámetro ::Converter 374](#_Toc5231525)

[35.4 El Parámetro {}Items 374](#_Toc5231526)

[34.5 El Parámetro Init 374](#_Toc5231527)

[35.6 Uso Típico del Browser 48 375](#_Toc5231528)

[35.7 NULLLAMs Usados por el Browser 48 376](#_Toc5231529)

[35.8 Referencia 377](#_Toc5231530)

[35.8.1 Cambiando el Ítem Actual 379](#_Toc5231531)

[35.8.1 Dibujando la Pantalla del Browser 48 379](#_Toc5231532)

[35.9 Ejemplos 381](#_Toc5231533)

[Capítulo 36 Browser 224 398](#_Toc5231534)

[36.1 El Parámetro Menú 399](#_Toc5231535)

[36.2 El Parámetro $Título 399](#_Toc5231536)

[36.3 El Parámetro AccionesENTER&ON 399](#_Toc5231537)

[36.4 El Parámetro #IndiceSuperior 399](#_Toc5231538)

[36.5 El Parámetro #IndiceInicial 400](#_Toc5231539)

[36.6 El Parámetro Items 400](#_Toc5231540)

[36.7 El Parámetro ::Converter 400](#_Toc5231541)

[36.8 El Parámetro {#letras} 401](#_Toc5231542)

[36.9 Los LAMs del Browser 224 401](#_Toc5231543)

[36.10 Atajos de Tecla del Browser 224 401](#_Toc5231544)

[36.11 Referencia 402](#_Toc5231545)

[36.12 Ejemplos Browser224 403](#_Toc5231546)

[Capítulo 37 Formularios de Entrada con IfMain 406](#_Toc5231547)

[37.1 Definiciones de Etiquetas 407](#_Toc5231548)

[37.2 Definiciones de Campos 407](#_Toc5231549)

[37.3 Número de Etiquetas y de Campos 410](#_Toc5231550)

[37.4 Message Handler 411](#_Toc5231551)

[37.5 El Título del Formulario de Entrada 411](#_Toc5231552)

[37.6 Resultados del Formulario de Entrada 412](#_Toc5231553)

[37.7 Los LAMs del IfMain 412](#_Toc5231554)

[37.8 Referencia 413](#_Toc5231555)

[37.9 Mensajes para Campos y para Formulario 417](#_Toc5231556)

[37.9.1 Mensaje 0 (campo y formulario) 418](#_Toc5231557)

[37.9.2 Mensaje 1 (campo) 421](#_Toc5231558)

[37.9.3 Mensaje 2 (formulario) 421](#_Toc5231559)

[37.9.4 Mensaje 3 (campo) 421](#_Toc5231560)

[37.9.5 Mensaje 4 (campo) 421](#_Toc5231561)

[37.9.6 Mensaje 5 (campo) 422](#_Toc5231562)

[37.9.7 Mensaje 6 (campo) 422](#_Toc5231563)

[37.9.8 Mensaje 7 (formulario) 422](#_Toc5231564)

[37.9.9 Mensaje 11 (formulario) 423](#_Toc5231565)

[37.9.10 Mensaje 12 (formulario) 424](#_Toc5231566)

[37.9.11 Mensaje 13 (formulario) 425](#_Toc5231567)

[37.9.12 Mensaje 14 (formulario) 425](#_Toc5231568)

[37.9.13 Mensaje 15 (formulario) 426](#_Toc5231569)

[37.9.14 Mensaje 16 (formulario) 427](#_Toc5231570)

[37.9.15 Mensaje 17 (campo y formulario) 429](#_Toc5231571)

[37.9.16 Mensaje 18 (formulario) 429](#_Toc5231572)

[37.9.17 Mensaje 20 (campo y formulario) 429](#_Toc5231573)

[37.9.18 Mensaje 21 (campo y formulario) 430](#_Toc5231574)

[37.9.19 Mensaje 22 (campo) 430](#_Toc5231575)

[37.9.20 Mensaje 23 (campo) 431](#_Toc5231576)

[37.9.21 Mensaje 24 (campo y formulario) 431](#_Toc5231577)

[37.9.22 Mensaje 25 (campo y formulario) 431](#_Toc5231578)

[37.9.23 Mensaje 26 (campo y formulario) 437](#_Toc5231579)

[37.10 Ejemplos 438](#_Toc5231580)

[Capítulo 38 Formularios de Entrada con DoInputForm 539](#_Toc5231581)

[38.1 Definiciones de Etiquetas 540](#_Toc5231582)

[38.2 Definiciones de Campos 540](#_Toc5231583)

[38.3 Número de Etiquetas y de Campos 544](#_Toc5231584)

[38.4 Message Handler 544](#_Toc5231585)

[38.5 El Título del Formulario de Entrada 545](#_Toc5231586)

[38.6 Resultados del Formulario de Entrada 545](#_Toc5231587)

[38.7 NULLLAMs Usados por DoInputForm 546](#_Toc5231588)

[38.8 Referencia 547](#_Toc5231589)

[36.8.1 Comandos Generales 547](#_Toc5231590)

[36.8.2 Menú en DoInputForm 548](#_Toc5231591)

[36.8.3 Visualización en Pantalla 548](#_Toc5231592)

[36.8.4 Teclas de Menú 550](#_Toc5231593)

[38.8.5 Parámetros de los Campos 551](#_Toc5231594)

[38.8.6 Parámetros de las Etiquetas 553](#_Toc5231595)

[38.9 Mensajes para Campos y para Formulario 554](#_Toc5231596)

[38.9.1 Mensaje 1 (formulario) 555](#_Toc5231597)

[38.9.2 Mensaje 2 (formulario) 555](#_Toc5231598)

[38.9.3 Mensaje 3 (formulario) 555](#_Toc5231599)

[38.9.4 Mensaje 4 (formulario) 556](#_Toc5231600)

[38.9.5 Mensaje 5 (formulario) 557](#_Toc5231601)

[38.9.6 Mensaje 6 (formulario) 562](#_Toc5231602)

[38.9.7 Mensaje 7 (campo) 564](#_Toc5231603)

[38.9.8 Mensaje 8 (campo) 565](#_Toc5231604)

[38.9.9 Mensaje 9 (campo) 566](#_Toc5231605)

[38.9.10 Mensaje 10 (campo) 567](#_Toc5231606)

[38.9.11 Mensaje 11 (campo) 568](#_Toc5231607)

[38.9.12 Mensaje 12 (campo) 568](#_Toc5231608)

[38.9.13 Mensaje 13 (campo) 568](#_Toc5231609)

[38.9.14 Mensaje 14 (formulario) 569](#_Toc5231610)

[38.9.15 Mensaje 15 (formulario) 570](#_Toc5231611)

[38.9.16 Mensaje 16 (formulario) 570](#_Toc5231612)

[38.9.17 Mensaje 17 (formulario) 571](#_Toc5231613)

[38.9.18 Mensaje 18 (campo) 571](#_Toc5231614)

[38.9.19 Mensaje 19 (campo) 572](#_Toc5231615)

[38.9.20 Mensaje 20 (campo) 572](#_Toc5231616)

[38.9.21 Mensaje 21 (campo) 572](#_Toc5231617)

[38.9.22 Mensaje 22 (campo) 572](#_Toc5231618)

[38.9.23 Mensaje 23 (campo) 573](#_Toc5231619)

[38.9.24 Mensaje 24 (campo) 573](#_Toc5231620)

[38.9.25 Mensaje 25 (campo) 573](#_Toc5231621)

[38.9.26 Mensaje 26 (campo) 573](#_Toc5231622)

[38.9.27 Mensaje 27 (campo) 574](#_Toc5231623)

[38.9.28 Mensaje 28 (formulario) 575](#_Toc5231624)

[38.9.29 Mensaje 29 (formulario) 575](#_Toc5231625)

[38.9.30 Mensaje 30 (campo) 575](#_Toc5231626)

[38.9.31 Mensaje 31 (campo) 576](#_Toc5231627)

[38.9.32 Mensaje 32 (campo) 576](#_Toc5231628)

[38.9.33 Mensaje 33 (campo) 577](#_Toc5231629)

[38.9.34 Mensaje 34 (campo) 577](#_Toc5231630)

[38.9.35 Mensaje 35 (campo) 577](#_Toc5231631)

[38.9.36 Mensaje 36 (campo) 578](#_Toc5231632)

[38.9.37 Mensaje 37 (campo) 580](#_Toc5231633)

[38.9.38 Mensaje 38 (campo) 582](#_Toc5231634)

[38.9.39 Mensaje 39 (campo) 582](#_Toc5231635)

[38.9.40 Mensaje 40 (campo) 583](#_Toc5231636)

[38.9.41 Mensaje 41 (campo) 584](#_Toc5231637)

[38.9.42 Mensaje 42 (campo) 584](#_Toc5231638)

[38.9.43 Mensaje 43 (campo) 584](#_Toc5231639)

[38.9.44 Mensaje 44 (campo) 585](#_Toc5231640)

[38.9.45 Mensaje 45 (campo) 585](#_Toc5231641)

[38.9.46 Mensaje 46 (campo) 585](#_Toc5231642)

[38.9.47 Mensaje 47 (campo) 586](#_Toc5231643)

[38.9.48 Mensaje 48 (campo) 586](#_Toc5231644)

[38.9.49 Mensaje 49 (campo) 586](#_Toc5231645)

[38.9.50 Mensaje 50 (campo) 586](#_Toc5231646)

[38.9.51 Mensaje 51 (campo) 586](#_Toc5231647)

[38.9.52 Mensaje 52 (campo) 587](#_Toc5231648)

[38.9.53 Mensaje 53 (campo) 587](#_Toc5231649)

[38.9.54 Mensaje 54 (campo) 587](#_Toc5231650)

[38.9.55 Mensaje 55 (campo) 587](#_Toc5231651)

[38.9.56 Mensaje 56 (formulario) 588](#_Toc5231652)

[38.10 Ejemplos DoInputForm 589](#_Toc5231653)

[Capítulo 39 La Pantalla 733](#_Toc5231654)

[39.1 Organización de la Pantalla 733](#_Toc5231655)

[39.2 Preparando la Pantalla 734](#_Toc5231656)

[39.3 Controlando el Refrescar Pantalla 735](#_Toc5231657)

[39.4 Limpiando la Pantalla 735](#_Toc5231658)

[39.5 Mostrando Texto 736](#_Toc5231659)

[39.6 Referencia 737](#_Toc5231660)

[39.6.1 Organización de la Pantalla 737](#_Toc5231661)

[39.6.2 Preparando la Pantalla 737](#_Toc5231662)

[39.6.3 Refrescando Inmediatamente la Pantalla 738](#_Toc5231663)

[39.6.4 Controlando el Refrescar Pantalla 739](#_Toc5231664)

[39.6.5 Limpiando la pantalla 741](#_Toc5231665)

[39.6.6 Anuncios y Modos de Control 742](#_Toc5231666)

[39.6.7 Coordenadas de la Ventana Visible 743](#_Toc5231667)

[39.6.8 Moviendo la Ventana 744](#_Toc5231668)

[39.6.9 Mostrando Objetos 745](#_Toc5231669)

[39.6.10 Fuentes y Altura de Pantalla 745](#_Toc5231670)

[39.6.11 Mostrando Texto en la Pantalla 747](#_Toc5231671)

[39.6.12 Mostrando Texto en el centro de la pantalla: DoMsgBox 751](#_Toc5231672)

[39.7 Ejemplos 753](#_Toc5231673)

[Capítulo 40 El Menú 759](#_Toc5231674)

[40.1 Formato de Menú 760](#_Toc5231675)

[40.2 Propiedades de un Menú 762](#_Toc5231676)

[40.3 Referencia 765](#_Toc5231677)

[40.3.1 Propiedades de Menú 765](#_Toc5231678)

[40.3.2 Creando Menús 767](#_Toc5231679)

[40.3.3 Mostrando el Menú 768](#_Toc5231680)

[40.3.4 Mostrando Etiquetas de Menú y Propiedad LabelDef 769](#_Toc5231681)

[40.3.5 Comandos Generales 769](#_Toc5231682)

[40.3.6 Evaluando contenido del Menú 770](#_Toc5231683)

[40.4 Ejemplos de Menús 772](#_Toc5231684)

[Capítulo 41 El Editor 777](#_Toc5231685)

[41.1 Terminología 777](#_Toc5231686)

[41.2 Referencia 778](#_Toc5231687)

[41.2.1 Estado 778](#_Toc5231688)

[41.2.2 Insertando Texto 779](#_Toc5231689)

[41.2.3 Borrando Texto 780](#_Toc5231690)

[41.2.4 Moviendo el Cursor 781](#_Toc5231691)

[41.2.5 Seleccionar, Cortar y Pegar, el Portapapeles 783](#_Toc5231692)

[41.2.6 Buscar y Reemplazar 785](#_Toc5231693)

[41.2.7 Evaluación 787](#_Toc5231694)

[41.2.8 Empezando el Editor 788](#_Toc5231695)

[41.2.9 Miscelánea 789](#_Toc5231696)

[41.3 Ejemplos Editor 793](#_Toc5231697)

[Capítulo 42 Trazado de Gráficos 797](#_Toc5231698)

[39.1 Referencia 798](#_Toc5231699)

[39.1.1 Comandos Generales 798](#_Toc5231700)

[39.1.2 Variable PPAR 799](#_Toc5231701)

[39.1.3 Coordenadas de los extremos de GBUFF 799](#_Toc5231702)

[39.1.4 Variable Independiente y Rango de Trazado 800](#_Toc5231703)

[39.1.5 Resolución 801](#_Toc5231704)

[39.1.6 Ejes y sus Etiquetas 801](#_Toc5231705)

[39.1.7 Tipo de Gráfico 802](#_Toc5231706)

[39.1.8 Variable Dependiente 802](#_Toc5231707)

[39.1.9 Escala 803](#_Toc5231708)

[39.1.10 Variable EQ 803](#_Toc5231709)

[39.1.11 Coordenadas Relativas y Absolutas 804](#_Toc5231710)

[Capítulo 43 Introducción al CAS de la HP 50g 806](#_Toc5231711)

[43.1 Problemas con Estos Capítulos 806](#_Toc5231712)

[43.2 Objetos Simbólicos 807](#_Toc5231713)

[43.3 Ejemplos 809](#_Toc5231714)

[Capítulo 44 Verificar Tipos y Conversión 810](#_Toc5231715)

[44.1 Referencia 810](#_Toc5231716)

[Capítulo 45 Enteros 812](#_Toc5231717)

[45.1 Referencia 812](#_Toc5231718)

[45.1.1 Enteros Ya Incorporados en ROM 812](#_Toc5231719)

[45.1.2 Enteros Ya Incorporados con Manipulación de Pila 813](#_Toc5231720)

[45.1.3 Conversión 813](#_Toc5231721)

[45.1.4 Operaciones Generales con Enteros 815](#_Toc5231722)

[45.1.5 Factorización de Enteros y Números Primos 817](#_Toc5231723)

[45.1.6 Enteros Gausianos 818](#_Toc5231724)

[45.1.7 Tests con Enteros 819](#_Toc5231725)

[Capítulo 46 Matrices 821](#_Toc5231726)

[46.1 Referencia 825](#_Toc5231727)

[46.1.1 Creando una Matriz Identidad 825](#_Toc5231728)

[46.1.2 Creando una Matriz Constante 826](#_Toc5231729)

[46.1.3 Creando Matrices con Elementos Aleatorios 827](#_Toc5231730)

[46.1.4 Creando Matrices con Elementos que estén en Función de su Fila y de su Columna 827](#_Toc5231731)

[46.1.5 Redimensionando Matrices 829](#_Toc5231732)

[46.1.6 Conversión 830](#_Toc5231733)

[46.1.7 Tests 831](#_Toc5231734)

[46.1.8 Calculos con matrices 833](#_Toc5231735)

[46.1.9 Transpuesta 837](#_Toc5231736)

[46.1.10 Determinante 838](#_Toc5231737)

[46.1.11 Norma de una Matriz 838](#_Toc5231738)

[46.1.12 Algebra Lineal y Reducción Gaussiana 839](#_Toc5231739)

[46.1.13 Resolviendo Sistemas Lineales 841](#_Toc5231740)

[46.1.14 Otras operaciones con matrices 842](#_Toc5231741)

[46.1.15 Insertar Filas en Matrices e Insertar objetos en Vectores 846](#_Toc5231742)

[46.1.16 Insertar Columnas 848](#_Toc5231743)

[46.1.17 Intercambio o Extracción de Filas, Columnas y Elementos 849](#_Toc5231744)

[46.1.18 Eigenvalores, Eigenvectores, Reducción. 851](#_Toc5231745)

[46.2 Ejemplos 854](#_Toc5231746)

[Capítulo 47 Manipulación de Expresiones 859](#_Toc5231747)

[47.1 Referencia 859](#_Toc5231748)

[47.1.1 Operaciones Básicas y Aplicación de Funciones 859](#_Toc5231749)

[47.1.2 Operadores Trigonométricos y Exponenciales 861](#_Toc5231750)

[47.1.3 Simplificación, Evaluación y Sustitución 863](#_Toc5231751)

[47.1.4 Colección y Expansión 864](#_Toc5231752)

[47.1.5 Transformaciones Trigonométricas 865](#_Toc5231753)

[47.1.6 División, GCD y LCM 866](#_Toc5231754)

[Capítulo 48 Manipulación de Metas Simbólicos 867](#_Toc5231755)

[48.1 Referencia 867](#_Toc5231756)

[48.1.1 Comandos Básicos para Manipular Expresiones 867](#_Toc5231757)

[48.1.2 Operaciones Básicas y Aplicación de Funciones 868](#_Toc5231758)

[48.1.3 Operadores Trigonométricos y Exponenciales 871](#_Toc5231759)

[48.1.4 Infinitos e Indefinidos 872](#_Toc5231760)

[48.1.5 Expansión y Simplificación 873](#_Toc5231761)

[48.1.6 Tests 874](#_Toc5231762)

[Capítulo 49 Polinomios 876](#_Toc5231763)

[49.1 Referencia 876](#_Toc5231764)

[49.1.1 Operaciones con polinomios 876](#_Toc5231765)

[49.1.2 Factorization 877](#_Toc5231766)

[49.1.3 General Polynomial Operations 880](#_Toc5231767)

[49.1.4 Tests 882](#_Toc5231768)

[Capítulo 50 Hallando Raíces de Ecuaciones 883](#_Toc5231769)

[50.1 Referencia 883](#_Toc5231770)

[50.1.1 Encontrando Raíces y Soluciones Numéricas 883](#_Toc5231771)

[Capítulo 51 Operaciones de Cálculo 887](#_Toc5231772)

[51.1 Referencia 887](#_Toc5231773)

[51.1.1 Limites y Expansión de Series 887](#_Toc5231774)

[51.1.2 Derivadas 888](#_Toc5231775)

[51.1.3 Integración 890](#_Toc5231776)

[51.1.4 Fracciones Parciales 890](#_Toc5231777)

[51.1.5 Ecuaciones Diferenciales 890](#_Toc5231778)

[51.1.6 Transformadas de Laplace 891](#_Toc5231779)

[Capítulo 52 Sumatorias 892](#_Toc5231780)

[52.1 Referencia 892](#_Toc5231781)

[Capítulo 53 Operaciones Modulares 893](#_Toc5231782)

[53.1 Referencia 893](#_Toc5231783)

[53.1.1 Operaciones Modulares 893](#_Toc5231784)

[53.1.2 Comando LNAME y Similares 894](#_Toc5231785)

[53.1.3 Comando LVAR y Similares 895](#_Toc5231786)

[Capítulo 54 Tablas de Signos 897](#_Toc5231787)

[54.1 Referencia 897](#_Toc5231788)

[Capítulo 55 Errores 899](#_Toc5231789)

[55.1 Referencia 899](#_Toc5231790)

[Capítulo 56 CAS Configuración 901](#_Toc5231791)

[56.1 Referencia 901](#_Toc5231792)

[Capítulo 57 CAS Menus 907](#_Toc5231793)

[57.1 Referencia 907](#_Toc5231794)

[Capítulo 58 Versiones Internas de los Comandos User RPL 908](#_Toc5231795)

[58.1 Referencia 908](#_Toc5231796)

[Capítulo 59 Miscelánea 912](#_Toc5231797)

[59.1 Referencia 912](#_Toc5231798)

[59.1.1 Rutinas Para Mostrar Modo Verboso 912](#_Toc5231799)

[59.1.2 Evaluación 912](#_Toc5231800)

[59.1.3 Conversión 913](#_Toc5231801)

[59.1.4 Qpi 913](#_Toc5231802)

[59.1.5 Infinito 914](#_Toc5231803)

[59.1.6 Constantes Ya Incorporadas 914](#_Toc5231804)

[59.1.7 Aplicaciones de Listas 914](#_Toc5231805)

[59.1.8 Irrquads 915](#_Toc5231806)

[59.1.9 Miscelánea 916](#_Toc5231807)

[Apéndice A Herramientas para Desarrollar 919](#_Toc5231808)

[A.1 La Biblioteca de Entradas (EXTABLE) 920](#_Toc5231809)

[A.2 Hacking Tools 921](#_Toc5231810)

[A.2.1 Biblioteca 256 921](#_Toc5231811)

[A.3 El Compilador 932](#_Toc5231812)

[A.4 Desensamblando 934](#_Toc5231813)

[Apéndice B Creación de Bibliotecas 935](#_Toc5231814)

[B.1 Creación de bibliotecas desde la Calculadora 935](#_Toc5231815)

[B.2 Creación de bibliotecas desde Debug 4x 937](#_Toc5231816)

[B.3 El Message Handler de la Biblioteca 940](#_Toc5231817)

[B.3.1 Extensiones de Menús 940](#_Toc5231818)

[B.3.2 Ayuda en Línea para Comandos de Biblioteca 945](#_Toc5231819)

[B.3.3 El Mensaje que Maneja el Menú de Bibliotecas 948](#_Toc5231820)

[Apéndice C Pasos Para Crear Bibliotecas con Debug 4x 949](#_Toc5231821)

[C.1 Instalar Debug 4x 949](#_Toc5231822)

[C.2 Ventana Principal (Main Window) 951](#_Toc5231823)

[C.3 Ordena las ventanas en la pantalla 952](#_Toc5231824)

[C.4 Crea un nuevo proyecto 952](#_Toc5231825)

[C.5 Selecciona opciones para tu proyecto. 953](#_Toc5231826)

[C.6 Abre el emulador que usará tu proyecto 953](#_Toc5231827)

[C.7 Crea Archivo Fuente para tus programas 954](#_Toc5231828)

[C.8 Tu primer programa en System RPL. 955](#_Toc5231829)

[C.9 Uso de la ventana Debugger RPL para depurar programas en System RPL 956](#_Toc5231830)

[Apéndice D Preguntas Frecuentes Sobre Debug 4x y System RPL 958](#_Toc5231831)

[D.1 Ver Argumentos y Resultados de Comandos en Debug 4x 958](#_Toc5231832)

[D.2 Autocompletar comandos en Debug 4x 959](#_Toc5231833)

[D.3 Comandos Permitidos y no Permitidos en Algebraicos con Debug4x 961](#_Toc5231834)

[D.4 Insertar un Grob en el Editor de Debug4x 962](#_Toc5231835)

[D.5 Más Preguntas y Respuestas 963](#_Toc5231836)

[Apéndice E Comandos de User RPL 964](#_Toc5231837)

[E.1 Referencia 965](#_Toc5231838)

[Apéndice F Mensajes de Error 998](#_Toc5231839)

Capítulo 1  
Introducción

Si conoces como crear programas en User RPL (si no sabes, deberías aprender primero eso antes de continuar leyendo este libro), entonces sólo conoces parte de lo que la calculadora HP puede hacer. El lenguaje de programación System RPL te da el poder de hacer muchas cosas que ni siquiera imaginaste. Por ejemplo, en System RPL puedes manejar objetos de todos los tipos disponibles. Con User RPL solamente puedes acceder a algunos de estos. También puedes hacer operaciones matemáticas con 15 dígitos de precisión, usar arreglos con elementos no numéricos, y mucho más. System RPL puede ser usado para hacer las mismas cosas que los programas hechos en User RPL, pero mucho más rápido.

Pero antes de que empecemos a hablar de System RPL, vayamos atrás, al User RPL para explicar cómo funciona realmente. Sabemos que estás ansioso por empezar desde ya con las cosas grandes, pero la siguiente información es importante para un buen entendimiento de System RPL.

Los programas (hechos en User o en System) no son guardados internamente usando los nombres de los comandos. Sólo son guardadas las direcciones de los objetos. Cada una de estas direcciones ocupa sólo 2.5 bytes (o más, si la dirección es un rompointer o un flashpointer). Cuando un programa es ejecutado, lo único que realmente hace este es una especie de “salto” a esas direcciones. Esta manera de guardar los programas obedece a dos propósitos. 2.5 bytes es menor que los nombres de los comandos, por lo tanto el programa ocupará una menor memoria. Y la ejecución del programa es mucho más rápida puesto que durante la ejecución, buscar las direcciones de los nombres no toma mucho tiempo. La mayoría de las veces, las direcciones apuntan a otros programas con más saltos a otros programas con más saltos, etc. La calculadora recuerda el camino de las direcciones de los saltos anteriores, por eso puedes tener tantos saltos como sean necesarios sin preocuparte de esto. Cuando el programa llamado termina, retornarás en donde estabas antes. Por supuesto, los saltos deben terminar en algún lugar, aún en un programa escrito en lenguaje de máquina o en un objeto que solamente se coloca en la pila (números, cadenas, etc). Esto es muy similar al concepto de llamado a una función o subrutina en lenguajes de alto nivel.

Pero si los programas son sólo direcciones ¿cómo pueden ser editados? La respuesta es que la calculadora tiene una tabla con los nombres de los comandos User y sus correspondientes direcciones. De esta manera, cuando pones un programa User RPL en la pila, la HP busca en la tabla para conseguir el nombre del comando correspondiente a la dirección guardada en memoria, y luego muestra el programa en forma leíble.

Puedes luego editarlo, y después que la edición es hecha, la HP nuevamente busca en la tabla las direcciones de los comandos ingresados, y sólo estas direcciones son guardadas en la memoria. Esta es la razón por la que editar un programa User RPL largo toma mucho tiempo, pero también es la razón de que el programa se ejecute más rápido.

Todo esto funciona para los comandos que tienen nombre. Pero hay más de cuatro mil comandos sin nombre. Esta es una de las diferencias entre User RPL y System RPL.

User RPL, el lenguaje descrito en la "Guía del Usuario" (el lenguaje cuyos programas tienen los delimitadores « ») puede acceder solamente a los comandos con nombre (en realidad, con User también puedes tener acceso a los comandos sin nombre mediante los comandos **SYSEVAL**, **LIBEVAL** y **FLASHEVAL**, si conoces las direcciones de los comandos. Pero esto no es muy eficiente, salvo para un uso ocasional). System RPL puede acceder a todos los comandos. En consecuencia, los programas System RPL no pueden ser editados directamente. Para editarlos se necesitan herramientas especiales. Tienes dos opciones para desarrollar software para la HP en el lenguaje System RPL. Los programas pueden escribirse y probarse en una PC usando Debug4x con el emulador, o también puedes escribir software directamente en la calculadora si descargas e instalas algunas bibliotecas necesarias. En el apéndice A encontrarás información sobre las herramientas disponibles para escribir programas System RPL.

El programar en System RPL es mucho más poderoso y mucho más rápido porque la mayoría de los comandos de System no hace verificación de errores. En System RPL, el programador debe estar seguro de que no ocurrirán errores, pues si ocurren entonces un crash podría suceder. Por ejemplo, si un comando requiere dos argumentos en la pila y estos no están o no son del tipo que la función requiere entonces podría suceder un reinicio en frio (warmstart) o aún peor: una pérdida de memoria. Naturalmente, hay comandos para verificar que hay suficientes argumentos, comandos para verificar el tipo del argumento y comandos para verificar otras posibles condiciones de error. La diferencia es que lo más probable es que tu verifiques que todos los argumentos estén presentes una sola vez, al comienzo del programa, sin necesidad de hacer después más verificaciones. En cambio, en User RPL, cada comando de un programa hace verificación de errores, de esta manera muchas verificaciones son hechas innecesariamente, volviendo más lento el programa.

En este punto, podrías preguntarte “¿Si los comandos no tienen nombre, cómo puedo programar en System RPL?” Como se dijo antes, todos los comandos tienen direcciones, de manera que tu puedes llamar a sus direcciones directamente.

Por ejemplo, para llamar al comando cuya dirección es **331C5** cuando programas en la calculadora puedes usar:

PTR 331C5

Cuando programas en Debug 4x puedes usar:

ASSEMBLE

CON(5) #331C5

RPL

Cualquiera que sea la dirección, esta será ejecutada. Pero hay una forma más fácil. Los comandos tienen nombres. Los nombres no son guardados de la misma manera que los comandos del User. Pero el equipo de diseño de la HP les ha dado nombres y estos nombres son guardados en las herramientas para crear programas en System RPL. Cuando escribes un programa usando aquellos nombres, el compilador System RPL busca los nombres en las tablas y los convierte a direcciones. Esto es llamado compilación. Algunas herramientas pueden también hacer lo contrario: convertir las direcciones en nombres de comandos (como cadenas de caracteres). Esto es llamado descompilación.

Algunos de estos comandos son clasificados como “soportados”: se garantiza su permanencia en la misma posición de la memoria en todas las versiones de ROM de la calculadora. En consecuencia, sus direcciones no cambiarán, de manera que los programadores pueden usarlos de forma segura. (Nota que esto no significa que ellos estarán en la misma dirección en diferentes calculadoras como HP48 series y HP49 series). Pero hay también comandos que son clasificados como “no soportados”. Para estos, no hay garantía de que permanecerán en la misma dirección en diferentes versiones de ROM. En las listas de entradas de este libro, los comandos no soportados son encerrados por paréntesis como por ejemplo el comando: (CURSOR\_PART).

Sin embargo, nota que todas las entradas no soportadas listadas en este libro son estables. El equipo de diseño de la HP ha indicado que todas las direcciones de la HP49 series en los rangos 025EC–0B3C7 y 25565–40000 permanecerán sin cambios, aún los comandos no soportados en esos rangos.

Actualmente hay 3 tipos de entradas: la descripción hecha arriba trató principalmente con las direcciones normales de 2.5 bytes, las cuales apuntan directamente a algunas direcciones de la ROM. La mayoría de entradas son de este tipo. Pero también hay entradas rompointer y entradas flashpointer.

Los **rompointer** apuntan a comandos que están dentro de alguna biblioteca. En este libro a los rompointers les anteponemos el carácter ~. Por ejemplo, para el comando ˜ChooseSimple que se describe en el libro de esta manera:

Direcc. Nombre Descripción .

0630B3 ˜ChooseSimple ( $titulo {items} ob 🡺 T )

Si programas en la calculadora puedes escribirlo de cualquiera de estas 3 maneras:

ROMPTR2 ˜ChooseSimple

ROMPTR B3 63

ROMPTR 0B3 063

Y si programas en el editor de Debug 4x puedes escribirlo de estas 2 maneras:

ROMPTR ChooseSimple

ROMPTR 0B3 063

Las **flashpointers** apuntan a subrutinas que están dentro de la memoria flash. En este libro a los flashpointers les anteponemos el carácter ^. Por ejemplo, para el comando ˆZ>R que se describe en el libro de esta manera:

Direcc. Nombre Descripción .

0F6006 ˆZ>R ( Z 🡺 % )

Convierte un entero a real.

Si programas en la calculadora puedes escribirlo de estas 3 maneras:

FPTR2 ^IfKeyChoose

FPTR 6 F6

FPTR 006 0F6

Y si programas en el editor de Debug 4x puedes escribirlo de estas 2 maneras:

FLASHPTR IfKeyChoose

FLASHPTR 006 0F6

1.1 Tu primer programa en System RPL

Crearemos un programa System RPL muy simple y lo explicaremos en detalles. Este programa hallará el área de un círculo, dándole el radio en la pila. Ver en el Apéndice A la información sobre como compilarlo.

::

CK1NOLASTWD ( verifica si hay un argumento )

CK&DISPATCH1 ( verifica si el argumento es real )

BINT1 ( si es real hace lo siguiente)

::

%2 %^ ( eleva al cuadrado el radio )

%PI ( pone PI en la pila )

%\* ( y multiplica los dos números de la pila )

;

;

Antes de empezar a analizarlo, es importante notar que System RPL distingue mayúsculas de minúsculas. Por lo tanto pi es diferente de PI, el cual es diferente de pI. También, como habrás adivinado, todo lo que está entre paréntesis () es considerado como un comentario. Las líneas que tienen un asterisco \* como su primer carácter son también comentarios.

Continuando, hay un comando llamado CK1NOLASTWD. Este comando verifica si hay un argumento en la pila, y si no hay genera el mensaje de error “Muy pocos argumentos”. El siguiente comando CK&DISPATCH0, verifica el tipo de argumento y permite al programador hacer diferentes cosas para diferentes tipos de argumentos. Nuestro programa sólo soporta un tipo de argumento: los números reales (representado aquí por BINT1, el número uno como un entero binario. Ver el capítulo 2). Si otro tipo de argumento está presente en la pila aparece el mensaje de error “Argumento incorrecto”. La verificación de argumentos es descrita en detalle en el capítulo 30.

Después de esto viene el código para ejecutar en el caso de que el objeto presente en la pila sea un número real. Nota que el código está entre los delimitadores :: y ;. Esto es porque el programa sólo espera un objeto después del tipo de argumento (BINT1). Aquí, este único objeto es un programa (subprograma). Este es un tipo de objeto compuesto. Un objeto compuesto es un objeto único que puede tener varios objetos dentro de el. De manera, que si queremos ejecutar varios comandos, estos comandos deben estar dentro de un programa.

El resto del programa es muy simple. El número real 2 es puesto en la pila, y el radio (ingresado por el usuario) es elevado a ese exponente (2).

Finalmente, π es puesto en la pila y el radio al cuadrado es multiplicado por este. En la pila ahora está el área del círculo.

El tamaño de este programa es de 25 bytes, en contraste a los 20 bytes del programa escrito en User RPL **« SQ p \* ->NUM »**. Sin embargo, la versión User RPL tomó 0.0156 segundos para el cálculo. La versión System RPL tomó solamente 0.0019 segundos.

Recuerda que si bien en este ejemplo el programa System RPL resultó tener un mayor tamaño, generalmente esto no sucederá.

1.2 Sobre la lista de entradas

En los siguientes capítulos, los diagramas de pila para describir a los comandos usan códigos para representar cada tipo de objeto. Aquí, una lista de tales códigos.

**Abreviatura** **Significado**

ob cualquier objeto

1...n n objetos

# entero binario del System (BINT)

HXS, hxs cadena hexadecimal (HXS)

CHR, chr carácter

$ cadena de caracteres

T bandera o flag TRUE

F bandera o flag FALSE

flag bandera o flag TRUE o FALSE

% número real

%% número real extendido

%C número complejo

%%C número complejo extendido

Z, z, ZINT número entero de precisión infinita

N número entero positivo de precisión infinita

s, symb simbólico

u, unit objeto unidad

{} lista

M, MAT, MATRIX matriz

2DMATRIX matriz de dos dimensiones

1DMATRIX matriz de una dimensión

ARRY arreglo (array)

2DARRY, [[]] arreglo de dos dimensiones

1DARRY, [] arreglo de una dimensión

RealARRY arreglo real

CmpARRY arreglo complejo

ExtRealARRY arreglo de reales extendidos

ExtCmpARRY arreglo de complejos extendidos

[%] arreglo real de una dimensión

[[%]] arreglo real de dos dimensiones

[C%] arreglo complejo de una dimensión

[[C%]] arreglo complejo de dos dimensiones

LNKARRY arreglo vinculado (linked array)

2DLNKARRY arreglo vinculado (linked array) de dos dimensiones

1DLNKARRY arreglo vinculado (linked array) de una dimensión

V, [] vector (matriz o arreglo de una dimensión)

P polinomio, una lista de Qs

Q ZINT o P

meta, ob1..obn #n objeto meta

grob objeto gráfico

menu menu: un programa o una lista

sign tabla de signos

Los diagramas de pila de los comandos User RPL usan algunas abreviaturas adicionales:

**Abreviatura** **Significado**

x, y, real, lista, objeto genérico en User RPL

c, (,) número complejo

# cadena hexadecimal (Entero binario del User)

θ ángulo (un número real)

m, n entero (ZINT)

fecha fecha en el formato DD.MMAAAA or MM.DDAAAA

name nombre global

prog, prg programa

f, func función

F integral de f

Capítulo 2  
Enteros Binarios (BINTS)

Los enteros binarios son los objetos que usarás más frecuentemente en System RPL. Estos no son los enteros binarios que usabas en User RPL (aquellos que al escribirlos empezaban con #) ; estos enteros binarios del User RPL se llamarán ahora cadenas hexadecimales, que se describirán en el capítulo 7.

Los enteros binarios del System RPL (llamados bints) son objetos que no son fácilmente accesibles al usuario. Si tu tienes uno de estos en la pila, se mostrará así: ¤ 10h.

Si tienes a la mano tu calculadora, intenta esto: ingresa en la pila el número #331A7h. Ahora escribe **SYSEVAL** y presiona ENTER. Deberías conseguir en la pila ¤ 10h, o quizás ¤ 16d, ¤ 20o ó ¤ 10000b, esto depende de la base numérica que estás usando. Internamente, los bints siempre están en modo hexadecimal. Con la biblioteca 256, puedes usar los comandos R˜SB y SB˜B para convertir números reales y enteros binarios del User a bints y viceversa.

Los bints son los objetos que usarás con más frecuencia debido a que la mayoría de los comandos que requieren de un argumento numérico, lo necesitan en la forma de un entero binario, a diferencia del lenguaje User RPL donde se requerían números reales. Por lo tanto, estos bints deberían ser fáciles de crear. Y en verdad lo son. Puedes poner uno en la pila solamente escribiéndolo en tu programa (en forma decimal). Pero esto no siempre es recomendable porque tu puedes poner un número real en la pila escribiéndolo de la misma manera (después veremos como diferenciar uno de otro). Por lo tanto, es una buena idea usar la siguiente estructura: # <hex>. De esta manera puedes asegurarte que conseguirás un número binario, y tu código es más claro. Al usar esa estructura, debes usar representación hexadecimal.

En la ROM de la calculadora, existen mucho enteros binarios ya incorporados. Tu puedes poner uno de estos en la pila simplemente llamando a su dirección.

Por ejemplo, para conseguir #6h en la pila, solo debes usar la palabra BINT6. La principal ventaja es que si ingresas #6, esto ocupa cinco bytes. En cambio, la palabra BINT6 como todos los otros comandos (excepto los comandos rompointer y flashpointer) ocupan sólo 2.5 bytes. Algunos comandos ponen dos o hasta tres bints en la pila, de manera que el ahorro de memoria es todavía mayor. Abajo hay una lista con los bints ya incorporados en la ROM.

Las cuatro operaciones básicas con bints son #+, #-, #\* y #/. Hay también muchas otras operaciones que están listadas abajo.

Aquí un ejemplo de un programa que pone tres bints en la pila usando los tres métodos.

::

13 ( 13d ó Dh )

# D ( lo mismo, pero usando el método preferido )

BINT13 ( este método ahorra memoria )

;

Los enteros binarios pueden tener hasta cinco cifras (en base hexadecimal).

El menor entero binario que se puede usar es # 0, el cual puedes llamar con el comando BINT0 o ZERO.

El mayor entero binario que se puede usar es # FFFFF (1 048 575 en el sistema decimal), el cual puedes llamar con el comando MINUSONE.

Si a un entero binario le restas otro entero binario mayor, la resta debería ser negativa, pero como no hay enteros binarios negativos, la respuesta no es la adecuada. En este caso, # FFFFF se considera como -1, # FFFFE se considera como -2, etc.

Por ejemplo:

# 7 # 9 #- retornará: # FFFFE

# 0 # 1 #- retornará: # FFFFF

A veces en lugar de usar el comando #-, será mas conveniente usar el comando DIFF\_OR\_ZERO\_. Este comando retorna la diferencia de dos bints, pero si la diferencia es negativa, retornará el bint cero.

2.1 Referencia

2.1.1 Enteros Binarios ya incorporados en ROM

**Direcc. Nombre Descripción**

33107 BINT0 0d 0h

aka: ZERO, any

33111 BINT1 1d 1h

aka: ONE, real, MEMERR

3311B BINT2 2d 2h

aka: TWO, cmp

33125 BINT3 3d 3h

aka: THREE, str

3312F BINT4 4d 4h

aka: FOUR, arry

33139 BINT5 5d 5h

aka: FIVE, list

33143 BINT6 6d 6h

aka: SIX, id, idnt

3314D BINT7 7d 7h

aka: SEVEN, lam

33157 BINT8 8d 8h

aka: EIGHT, seco

33161 BINT9 9d 9h

aka: NINE, symb

3316B BINT10 10d Ah

aka: TEN, sym

33175 BINT11 11d Bh

aka: ELEVEN, hxs

3317F BINT12 12d Ch

aka: TWELVE, grob

33189 BINT13 13d Dh

aka: TAGGED, THIRTEEN

33193 BINT14 14d Eh

aka: EXT, FOURTEEN, unitob

3319D BINT15 15d Fh

aka: FIFTEEN, rompointer

331A7 BINT16 16d 10h

aka: REALOB, SIXTEEN

331B1 BINT17 17d 11h

aka: SEVENTEEN, 2REAL, REALREAL

331BB BINT18 18d 12h

aka: EIGHTEEN

331C5 BINT19 19d 13h

aka: NINETEEN

331CF BINT20 20d 14h

aka: TWENTY

331D9 BINT21 21d 15h

aka: TWENTYONE

331E3 BINT22 22d 16h

aka: TWENTYTWO

**Direcc. Nombre Descripción**

331ED BINT23 23d 17h

aka: TWENTYTHREE

331F7 BINT24 24d 18h

aka: TWENTYFOUR

33201 BINT25 25d 19h

aka: TWENTYFIVE

3320B BINT26 26d 1Ah

aka: REALSYM, TWENTYSIX

33215 BINT27 27d 1Bh

aka: TWENTYSEVEN

3321F BINT28 28d 1Ch

aka: TWENTYEIGHT

33229 BINT29 29d 1Dh

aka: TWENTYNINE

33233 BINT30 30d 1Eh

aka: REALEXT, THIRTY

3323D BINT31 31d 1Fh

aka: THIRTYONE

33247 BINT32 32d 20h

aka: THIRTYTWO

33251 BINT33 33d 21h

aka: THIRTYTHREE

3325B BINT34 34d 22h

aka: THIRTYFOUR

33265 BINT35 35d 23h

aka: THIRTYFIVE

3326F BINT36 36d 24h

aka: TTHIRTYSIX

33279 BINT37 37d 25h

aka: THIRTYSEVEN

33283 BINT38 38d 26h

aka: THIRTYEIGHT

3328D BINT39 39d 27h

aka: THIRTYNINE

33297 BINT40 40d 28h

aka: FORTY, FOURTY

332A1 BINT41 41d 29h

aka: FORTYONE

332AB BINT42 42d 2Ah

aka: FORTYTWO

332B5 BINT43 43d 2Bh

aka: FORTYTHREE

332BF BINT44 44d 2Ch

aka: FORTYFOUR

332C9 BINT45 45d 2Dh

aka: FORTYFIVE

332D3 BINT46 46d 2Eh

aka: FORTYSIX

332DD BINT47 47d 2Fh

aka: FORTYSEVEN

**Direcc. Nombre Descripción**

332E7 BINT48 48d 30h

aka: FORTYEIGHT

332F1 BINT49 49d 31h

aka: FORTYNINE

332FB BINT50 50d 32h

aka: FIFTY

33305 BINT51 51d 33h

aka: FIFTYONE

3330F BINT52 52d 34h

aka: FIFTYTWO

33319 BINT53 53d 35h

aka: FIFTYTHREE, STRLIST, THREEFIVE

33323 BINT54 54d 36h

aka: FIFTYFOUR

3332D BINT55 55d 37h

aka: FIFTYFIVE

33337 BINT56 56d 38h

aka: FIFTYSIX

33341 BINT57 57d 39h

aka: FIFTYSEVEN

3334B BINT58 58d 3Ah

aka: FIFTYEIGHT

33355 BINT59 59d 3Bh

aka: FIFTYNINE

3335F BINT60 60d 3Ch

aka: SIXTY

33369 BINT61 61d 3Dh

aka: SIXTYONE

33373 BINT62 62d 3Eh

aka: SIXTYTWO

3337D BINT63 63d 3Fh

aka: SIXTYTHREE

33387 BINT64 64d 40h

aka: BINT40h, SIXTYFOUR, YHI

33391 BINT65 65d 41h

aka: ARRYREAL

3339B BINT66 66d 42h

aka: FOURTWO

333A5 BINT67 67d 43h

aka: FOURTHREE

333AF BINT68 68d 44h

aka: SIXTYEIGHT

333B9 BINT69 69d 45h

aka: FOURFIVE

333C3 BINT70 70d 46h

aka: SEVENTY

333CD BINT71 71d 47h

333D7 BINT72 72d 48h

333E1 BINT73 73d 49h

**Direcc. Nombre Descripción**

333EB BINT74 74d 4Ah

aka: SEVENTYFOUR

333F5 BINT75 75d 4Bh

333FF BINT76 76d 4Ch

33409 BINT77 77d 4Dh

33413 BINT78 78d 4Eh

3341D BINT79 79d 4Fh

aka: SEVENTYNINE

33427 BINT80 80d 50h

aka: EIGHTY

33431 BINT81 81d 51h

aka: EIGHTYONE, LISTREAL

3343B BINT82 82d 52h

aka: LISTCMP

33445 BINT83 83d 53h

aka: FIVETHREE

3344F BINT84 84d 54h

aka: FIVEFOUR

33459 BINT85 85d 55h

aka: 2LIST

33463 BINT86 86d 56h

aka: FIVESIX

3346D BINT87 87d 57h

aka: LISTLAM

33477 BINT88 88d 58h

33481 BINT89 89d 59h

3348B BINT90 90d 5Ah

33495 BINT91 91d 5Bh

aka: BINT\_91d

3349F BINT92 92d 5Ch

334A9 BINT93 93d 5Dh

334B3 BINT94 94d 5Eh

334BD BINT95 95d 5Fh

334C7 BINT96 96d 60h

aka: BINT\_96d

334D1 BINT97 97d 61h

aka: IDREAL

334DB BINT98 98d 62h

334E5 BINT99 99d 63h

334EF BINT100 100d 64h

aka: ONEHUNDRED

334F9 BINT101 101d 65h

33503 BINT102 102d 66h

3350D BINT103 103d 67h

33517 BINT104 104d 68h

33521 BINT105 105d 69h

3352B BINT106 106d 6Ah

33535 BINT107 107d 6Bh

3353F BINT108 108d 6Ch

33549 BINT109 109d 6Dh

**Direcc. Nombre Descripción**

33553 BINT110 110d 6Eh

3355D BINT111 111d 6Fh

aka: char

33567 BINT112 112d 70h

33571 BINT113 113d 71h

3357B BINT114 114d 72h

33585 BINT115 115d 73h

aka: BINT\_115d

3358F BINT116 116d 74h

aka: BINT\_116d

33599 BINT117 117d 75h

335A3 BINT118 118d 76h

335AD BINT119 119d 77h

335B7 BINT120 120d 78h

335C1 BINT121 121d 79h

335CB BINT122 122d 7Ah

aka: BINT\_122d

335D5 BINT123 123d 7Bh

335DF BINT124 124d 7Ch

335E9 BINT125 125d 7Dh

335F3 BINT126 126d 7Eh

335FD BINT127 127d 7Fh

33607 BINT128 128d 80h

aka: BINT80h

33611 BINT129 129d 81h

3361B BINT130 130d 82h

aka: BINT130d, BINT\_130d, XHI-1

33625 BINT131 131d 83h

aka: BINT\_131d, BINT131d, XHI

3362F (#8F) 143d 8Fh

33639 SYMBREAL 145d 91h

33643 (SYMBCMP) 146d 92h

3364D (SYMBSYM) 154d 9Ah

33657 SYMBUNIT 158d 9Eh

3EAFB (#9F) 159d 9Fh

3366B SYMOB 160d A0h

33675 SYMREAL 161d A1h

3367F (SYMCMP) 162d A2h

39E6B (SYMARRY) 164d A4h

33689 (SYMLIST) 165d A5h

33693 SYMID 166d A6h

3369D SYMLAM 167d A7h

336A7 (SYMSYMB) 169d A9h

336B1 SYMSYM 170d AAh

336BB SYMEXT 174d AEh

3BD4C (#AF) 175d AFh

336C5 (HXSREAL) 177d B1h

38275 (#BB) 187d BBh

336CF (2HXS) 187d BBh

336D9 BINTC0h 192d C0h

**Direcc. Nombre Descripción**

3E7DA (#C8) 200d C8h

336E3 2GROB 204d CCh

3BD65 (#CF) 207d CFh

336ED TAGGEDANY 208d D0h

336F7 EXTREAL 225d E1h

33701 EXTSYM 234d EAh

3370B 2EXT 238d EEh

33715 ROMPANY 240d F0h

3371F BINT253 253d FDh

33729 BINT255d 255d FFh

33733 REALOBOB 256d 100h

3373D #\_102 258d 102h

33747 #SyntaxErr 262d 106h

33751 (BINT\_263d) 263d 107h

3375B (REALREALOB) 272d 110h

33765 3REAL 273d 111h

3E17B (#111) 273d 111h

3376F (Err#Kill) 291d 123h

33779 (Err#NoLstStk) 292d 124h

2777E (#12F) 303d 12Fh

33783 (#NoRoomForSt) 305d 131h

3378D (#132) 306d 132h

33797 (REALSTRSTR) 307d 133h

337A1 (#134) 308d 134h

337AB (#135) 309d 135h

337B5 (#136) 310d 136h

337BF (#137) 311d 137h

337C9 (#138) 312d 138h

337D3 (#139) 313d 139h

337DD (#13A) 314d 13Ah

337E7 (#13B) 315d 13Bh

337F1 (#13D) 317d 13Dh

337FB (Err#Cont) 318d 13Eh

33805 INTEGER337 337d 151h

3380F (CMPOBOB) 512d 200h

33819 (Err#NoLstArg) 517d 205h

3A1C2 (#304) 772d 304h

33823 STRREALREAL 785d 311h

3B9FA (#313) 787d 313h

3C11E (ARRYREALOB) 1040d 410h

3B928 (#411) 1041d 411h

3382D (ARRYREALREAL) 1041d 411h

33837 (ARRYREALCMP) 1042d 412h

3BA2D (#414) 1044d 414h

3B93D (#415) 1045d 415h

33841 (3ARRY) 1092d 444h

3C10F (ARRYLISTOB) 1104d 450h

3B952 (#451) 1105d 451h

3384B (ARRYLISTREAL) 1105d 451h

33855 (ARRYLISTCMP) 1106d 452h

**Direcc. Nombre Descripción**

3BA18 (#454) 1108d 454h

3B913 (#455) 1109d 455h

3A12D (#4FF) 1279d 4FFh

3385F (LISTREALOB) 1296d 510h

33869 (LISTREALREAL) 1297d 511h

3BA09 (#515) 1301d 515h

33873 (LISTLISTOB) 1360d 550h

277F6 (LN\_0) 1541d 605h

27800 (LN\_Neg) 1542d 606h

2780A (InvalidEQ) 1543d 607h

27814 (Cureq#) 1544d 608h

2781E (NoCureq#) 1545d 609h

27828 (EnterEq#) 1546d 60Ah

27832 (EnterName#) 1547d 60Bh

2783C (SelPtype#) 1548d 60Ch

27846 (EmptyCat#) 1549d 60Dh

2768E (#60E) 1550d 60Eh

27698 (NoStatPlot#) 1551d 60Fh

3387D (IDREALOB) 1552d 610h

276AC (SolvingFor#) 1553d 611h

276B6 (NoCurrent#) 1554d 612h

276C0 (PressSig+#) 1555d 613h

276CA (SelectModl#) 1556d 614h

276D4 (NoAlarms#) 1557d 615h

276DE (PressALRM#) 1558d 616h

276E8 (NextALRM#) 1559d 617h

27792 (PastDue#) 1560d 618h

2779C (Acknowledge#) 1561d 619h

277A6 (KeyInAlrm#) 1562d 61Ah

277B0 (SelectRpt#) 1563d 61Bh

277BA (IOSetupMenu#) 1564d 61Ch

277C4 (PlotType#) 1565d 61Dh

277CE (NoExectAct#) 1566d 61Eh

277D8 (OffScreen#) 1567d 61Fh

277E2 (OnlyPtypes#) 1568d 620h

277EC (StatName#) 1569d 621h

276F2 (ZoomPrompt#) 1570d 622h

276FC (CatToStack#) 1571d 623h

27706 (XAutoZoom#) 1572d 624h

27710 (IR/wire#) 1576d 628h

2771A (ASCII/bin#) 1577d 629h

27724 (#62A) 1578d 62Ah

2772E (#62B) 1579d 62Bh

27738 (#62C) 1580d 62Ch

27742 (#62D) 1581d 62Dh

27788 (EnterMatrix#) 1582d 62Eh

33887 (IDLISTOB) 1616d 650h

33891 (FSTMACROROM#) 1792d 700h

3C17A (#710) 1808d 710h

3C16B (#750) 1872d 750h

**Direcc. Nombre Descripción**

08DF7 (#7FF) 2047d 7FFh

27878 (BINT800h) 2048d 800h

3B976 (#822) 2082d 822h

3C83C (#82C) 2092d 82Ch

3B967 (#855) 2133d 855h

3C81E (#85C) 2140d 85Ch

3389B (PROGIDREAL) 2145d 861h

338A5 (PROGIDCMP) 2146d 862h

338AF (PROGIDLIST) 2149d 865h

338B9 (PROGIDEXT) 2158d 86Eh

3E7FF (#8F1) 2289d 8F1h

3E759 (#8FD) 2301d 8FDh

3E7E9 (#9F1) 2545d 9F1h

3E743 (#9FD) 2557d 9FDh

2774C (Lackint#) 2561d A01h

27756 (Constant#) 2562d A02h

27882 Attn# 2563d A03h

338C3 ATTNERR 2563d A03h

27760 (Zero#) 2564d A04h

2776A (RevSgn#) 2565d A05h

27774 (Extremum#) 2566d A06h

338CD (SYMREALREAL) 2577d A11h

338D7 (SYMREALCMP) 2578d A12h

338E1 (SYMREALSYM) 2586d A1Ah

338EB (SYMCMPREAL) 2593d A21h

338F5 (SYMCMPCMP) 2594d A22h

338FF (SYMCMPSYM) 2602d A2Ah

33909 (SYMIDREAL) 2657d A61h

33913 (SYMIDCMP) 2658d A62h

3391D (SYMIDLIST) 2661d A65h

33927 (SYMIDEXT) 2670d A6Eh

33931 (SYMSYMREAL) 2721d AA1h

3393B (SYMSYMCMP) 2722d AA2h

33945 (3SYM) 2730d AAAh

3394F (XFERFAIL) 3078d C06h

33959 (PROTERR) 3079d C07h

33963 (InvalServCmd) 3080d C08h

3396D Connecting 3082d C0Ah

33977 (Retry) 3083d C0Bh

3C800 (#C2C) 3116d C2Ch

3C7E2 (#C5C) 3164d C5Ch

3B904 (#C22) 3106d C22h

3B8F5 (#C55) 3157d C55h

33981 #CAlarmErr 3583d DFFh

3398B EXTOBOB 3584d E00h

3C8D0 (#2111) 8465d 2111h

03FEF (TYPEINT) 9748d 2614h

03FF9 (TYPEMATRIX) 9862d 2686h

03F8B TYPEREAL 10547d 2933h

03FDB (TYPEEREL) 10581d 2955h

**Direcc. Nombre Descripción**

03F95 (TYPECMP) 10615d 2977h

03F9F (TYPELIST) 10868d 2A74h

03FC7 (TYPERRP) 10902d 2A96h

03FBD (TYPESYMB) 10936d 2AB8h

03FE5 (TYPEEXT) 10970d 2ADAh

03FA9 (TYPEIDNT) 11848d 2E48h

03FD1 (TYPELAM) 11885d 2E6Dh

3C8DF (#5B11) 23313d 5B11h

3D50D (SYMRRANY) 41232d A110h

3D52B (SYMRSYMANY) 41376 A1A0h

3D51C (SYMSYMRANY) 43536d AA10h

2C4D2 (SYMSYMSYMANY) 43680d AAA0h

3B7AD (#BBBB) 48059d BBBBh

08F1F (#D6A8) 54952d D6A8h

38266 (#FFFF) 65535d FFFFh

03880 (#102A8) 66216d 102A8h

091B4 (#2D541) 185665d 2D541h

350F5 (#37258) 225880d 37258h

0803F (#414C1) 267457d 414C1h

08ECE (#536A8) 341672d 536A8h

0657E (#61441) 398401d 61441h

33995 #EXITERR 458752d 70000h

03826 (#A8241) 688705d A8241h

39277 (#B437D) 738173d B437Dh

038DC (#E13A8) 922536d E13A8h

3399F MINUSONE 1048575d FFFFFh

2.1.2 Poniendo varios BINTS

**Direcc. Nombre Descripción**

37287 ZEROZERO ( 🡺 #0 #0 )

37294 #ZERO#ONE ( 🡺 #0 #1 )

37305 #ZERO#SEVEN ( 🡺 #0 #7 )

36B12 ONEONE ( 🡺 #1 #1 )

aka: ONEDUP

37315 #ONE#27 ( 🡺 #1 #27d )

37328 #TWO#ONE ( 🡺 #2 #1 )

3733A #TWO#TWO ( 🡺 #2 #2 )

3734A #TWO#FOUR ( 🡺 #2 #4 )

3735C #THREE#FOUR ( 🡺 #3 #4 )

3736E #FIVE#FOUR ( 🡺 #5 #4 )

37380 ZEROZEROZERO ( 🡺 #0 #0 #0 )

37394 ZEROZEROONE ( 🡺 #0 #0 #1 )

373A8 ZEROZEROTWO ( 🡺 #0 #0 #2 )

3558C DROPZERO ( ob 🡺 #0 )

355A5 2DROP00 ( ob ob 🡺 #0 #0 )

3596D DROPONE ( ob 🡺 #1 )

36AD6 DUPZERO ( ob 🡺 ob ob #0 )

36AEA DUPONE ( ob 🡺 ob ob #1 )

36B26 DUPTWO ( ob 🡺 ob ob #2 )

36AFE SWAPONE ( ob ob' 🡺 ob' ob #1 )

**Direcc. Nombre Descripción**

35E75 ZEROSWAP ( ob 🡺 #0 ob )

360BB ZEROOVER ( ob 🡺 ob #0 ob )

36568 ZEROFALSE ( 🡺 #0 F )

35EA2 ONESWAP ( ob 🡺 #1 ob )

3657C ONEFALSE ( 🡺 #1 F )

2.1.3 Conversión

**Direcc. Nombre Descripción**

262F1 COERCE ( % 🡺 # )

35D08 COERCEDUP ( % 🡺 # # )

35EB6 COERCESWAP ( ob % 🡺 # ob )

3F481 COERCE2 ( % %' 🡺 # #' )

262EC %ABSCOERCE ( % 🡺 # )

Halla el valor absoluto de un número real y lo convierte a bint.

2F244 (Flag%isUser?) ( % 🡺 # flag )

TRUE si real es mayor que cero, de lo contrario pone FALSE.

05A03 HXS># ( hxs 🡺 # )

2F17E 2HXSLIST? ( { hxs hxs' } 🡺 # #' )

Convierte lista de dos hxs a dos bints. Genera el error "Argumento: valor incorrecto" para entradas inválidas.

05A51 CHR># ( chr 🡺 # )

Retorna el código ASCII correspondiente a un caracter.

0EF006 ˆZ2BIN ( Z 🡺 # )

Convierte un número entero (Z) a bint.

Retorna #FFFFF para enteros positivos muy grandes.

Returna #0 para enteros negativos del -1 al -9.

19D006 ˆZ># ( Z 🡺 # )

Convierte Z a #, pone mensaje de error si Z<0 ó Z>9999.

0F0006 ˆCOERCE2Z ( Z2 Z1 🡺 #2 #1 )

Convierte 2 zints a bints.

2.1.3 Funciones Aritméticas

**Direcc. Nombre Descripción**

03DBC #+ ( # #' 🡺 #+#' )

Suma de dos bints.

Si la suma es mayor a FFFFF, pone las 5 últimas cifras.

03DEF #1+ ( # 🡺 #+1 )

03E2D #2+ ( # 🡺 #+2 )

355FD #3+ ( # 🡺 #+3 )

35602 #4+ ( # 🡺 #+4 )

35607 #5+ ( # 🡺 #+5 )

3560C #6+ ( # 🡺 #+6 )

35611 #7+ ( # 🡺 #+7 )

35616 #8+ ( # 🡺 #+8 )

3561B #9+ ( # 🡺 #+9 )

35620 #10+ ( # 🡺 #+10 )

35625 (#11+) ( # 🡺 #+11 )

3562A #12+ ( # 🡺 #+12 )

**Direcc. Nombre Descripción**

03DE0 #- ( # #' 🡺 #-#' )

Diferencia de dos bints.

Si #' es mayor que #, retorna FFFFF - (#'-#)+1, de lo contrario retorna #-#'.

2F13D (DIFF\_OR\_ZERO) ( # #' 🡺 #'' )

Si #' es mayor que #, retorna #0, de lo contrario retorna #-#'.

03E0E #1- ( # 🡺 #-1 )

03E4E #2- ( # 🡺 #-2 )

355DF #3- ( # 🡺 #-3 )

355DA #4- ( # 🡺 #-4 )

355D5 #5- ( # 🡺 #-5 )

355D0 #6- ( # 🡺 #-6 )

355CB (#7-) ( # 🡺 #-7 )

355C6 (#8-) ( # 🡺 #-8 )

355C1 (#9-) ( # 🡺 #-9 )

03EC2 #\* ( # #' 🡺 #\*#' )

Producto de dos bints.

Si el producto es mayor a FFFFF, pone las 5 últimas cifras.

2632D #\*OVF ( # #' 🡺 #\*#' )

Producto de dos bints.

Si el producto es mayor a FFFFF, pone FFFFF.

03E6F #2\* ( # 🡺 #\*2 )

356B8 #6\* ( # 🡺 #\*6 )

3569B #8\* ( # 🡺 #\*8 )

35675 #10\* ( # 🡺 #\*10 )

03EF7 #/ ( # #' 🡺 #r #q )

03E8E #2/ ( # 🡺 #/2 )

Redondea hacia abajo.

36815 #1-- ( # #' 🡺 #-#'+1 )

aka: #-+1

36851 #+-1 ( # #' 🡺 #+#'-1 )

aka: $1-+

35552 #-#2/ ( # #' 🡺 (#-#')/2 )

357FC #+DUP ( # #' 🡺 #+#' #+#' )

35E39 #+SWAP ( ob # #' 🡺 #+#' ob )

36093 #+OVER ( ob # #' 🡺 ob #+#' ob )

3581F #-DUP ( # #' 🡺 #-#' #-#' )

35E4D #-SWAP ( ob # #' 🡺 #-#' ob )

360A7 #-OVER ( ob # #' 🡺 ob #-#' ob )

35830 #1+DUP ( # 🡺 #+1 #+1 )

35E61 #1+SWAP ( ob # 🡺 #+1 ob )

2F222 #1+ROT ( ob ob' # 🡺 ob' #+1 ob )

35841 #1-DUP ( # 🡺 #-1 #-1 )

28071 #1-SWAP ( ob # 🡺 #-1 ob )

aka: pull

3601B #1-ROT ( ob ob' # 🡺 ob' #-1 ob )

281D5 #1-UNROT ( ob ob' # 🡺 #-1 ob ob' )

35E89 #1-1SWAP ( # 🡺 1 #-1 )

Retorna el bint ONE y el resultado.

**Direcc. Nombre Descripción**

35912 DUP#1+ ( # 🡺 # #+1 )

3571E DUP#2+ ( # 🡺 # #+2 )

35956 DUP#1- ( # 🡺 # #-1 )

3674D 2DUP#+ ( # #' 🡺 # #' #+#' )

aka: DUP3PICK#+

3683D DROP#1- ( # ob 🡺 #-1 )

357BB SWAP#- ( # #' 🡺 #'-# )

3592B SWAP#1+ ( # ob 🡺 ob #+1 )

aka: SWP1+

29786 ('RSWP1+) ( # 🡺 nob #+1 )

Donde nob es el siguiente objeto del programa.

Por ejemplo, el siguiente programa retorna 123.25 BINT8

:: BINT7 'RSWP1+\_ 123.25 ;

2979A ('R'RROT2+) ( # 🡺 nob1 nob2 #+2 )

Donde nob1 y nob2 son los siguientes objetos del programa.

28099 SWAP#1+SWAP ( # ob 🡺 #+1 ob )

36829 SWAP#1- ( # ob 🡺 ob #-1 )

280AD SWAP#1-SWAP ( # ob 🡺 #-1 ob )

28989 (SWAPDROP#1-) ( ob # 🡺 #-1 )

367ED SWAPOVER#- ( # #' 🡺 #' #-#' )

36775 OVER#+ ( # #' 🡺 # #'+# )

367C5 OVER#- ( # #' 🡺 # #'-# )

36761 ROT#+ ( # ob #' 🡺 ob #'+# )

367B1 ROT#- ( # ob #' 🡺 ob #'-# )

36801 ROT#1+ ( # ob ob' 🡺 ob ob' #+1 )

28001 ROT#1+UNROT ( # ob ob' 🡺 #+1 ob ob' )

35E07 ROT#+SWAP ( # ob #' 🡺 #'+# ob )

aka: ROT+SWAP

36789 3PICK#+ ( # ob #' 🡺 # ob #'+# )

3679D 4PICK#+ ( # ob1 ob2 #' 🡺 # ob1 ob2 #'+# )

35E20 4PICK#+SWAP ( # ob1 ob2 #' 🡺 # ob1 #'+# ob2 )

aka: 4PICK+SWAP

35511 #MIN ( # #' 🡺 #'' )

3551D #MAX ( # #' 🡺 #'' )

03EB1 #AND ( # #' 🡺 #'' )

AND bit a bit.

2.1.5 Tests

**Direcc. Nombre Descripción**

03D19 #= ( # 🡺 flag )

03D4E #<> ( # 🡺 flag )

03CE4 #< ( # 🡺 flag )

03D83 #> ( # 🡺 flag )

03CC7 #0<> ( # 🡺 flag )

03CA6 #0= ( # 🡺 flag )

3530D #1<> ( # 🡺 flag )

352FE #1= ( # 🡺 flag )

36711 #2<> ( # 🡺 flag )

352F1 #2= ( # 🡺 flag )

352E0 #3= ( # 🡺 flag )

**Direcc. Nombre Descripción**

366FD #5= ( # 🡺 flag )

366BC #<3 ( # 🡺 flag )

36739 #>1 ( # 🡺 flag )

aka: ONE#>

358C2 2DUP#< ( # #' 🡺 # #' flag )

358F8 2DUP#> ( # #' 🡺 # #' flag )

363CE ONE\_EQ ( # 🡺 flag )

Usa EQ como test.

35268 OVER#= ( # #' 🡺 # flag )

358DC 2DUP#= ( # #' 🡺 # #' flag )

36694 OVER#0= ( # #' 🡺 # #' flag )

352BD DUP#0= ( # 🡺 # flag )

366A8 OVER#< ( # #' 🡺 # flag )

3531C DUP#1= ( # 🡺 # flag )

36725 OVER#> ( # #' 🡺 # flag )

3532B DUP#0<> ( # 🡺 # flag )

366D0 DUP#<7 ( # 🡺 # flag )

Retorna TRUE si el argumento es menor que #7.

36676 2#0=OR ( # # 🡺 flag )

Retorna TRUE si alguno de los dos bints es cero.

Capítulo 3  
Números Reales

Los números reales pueden ser creados de dos maneras.

La primera es solamente escribiéndolos sin ningún prefijo. Pero este método también puede ser usado para crear bints. ¿Entonces, como hace el compilador para saber cuando tu quieres un número real o un bint? Si el número incluye un punto y/o un exponente, entonces es un número real. De lo contrario, es un bint.

Así para escribir el número real 5 puedes escribir de estas cuatro maneras:

5.

5.E0

% 5.

% 5

En el editor de Debug4x además puedes usar estas otras dos maneras:

5E0

5e0

Sin embargo, el método preferido es usar la estructura % <dec>. De esta manera, te aseguras de conseguir siempre un número real y el código será más entendible.

Como en el caso de los bints, hay también muchos números reales ya incorporados en la ROM y están listados abajo. Estos te permitirán ahorrar memoria. Por ejemplo, % 5 ocupa 10.5 bytes, pero %5 sólo ocupa 2.5 bytes.

Las operaciones básicas que usan números reales son: %+, %-, %\*, %/ y %ˆ. Pero también hay muchas otras, las cuales son listadas abajo.

También hay otro tipo de números reales, los cuales no son accesibles directamente al usuario y a los programas de User RPL. Estos son los Números Reales Extendidos (o Largos). Estos funcionan como los números reales normales, pero hay dos diferencias:

Tienen 15 dígitos de precisión, a diferencia de los 12 dígitos de los números reales normales, y sus exponentes están en el rango -49999 a 49999, a diferencia de los números reales normales cuyos exponentes están en el rango -499 a 499.

Los números reales extendidos son creados usando la estructura %% <dec>. Si tienes un número real extendido en la pila, este se muestra como un número real normal, pero siempre en notación científica. Las operaciones básicas son las mismas, excepto que usan el prefijo %% en vez de %. Aclaremos de una vez lo siguiente: en User RPL el comando + adiciona cualquier tipo de objetos, por ejemplo números reales, objetos simbólicos, matrices, agrega elementos a listas, etc. En System RPL, el comando %+ solamente funciona para dos números reales. Para sumar dos números enteros binarios debes usar #+. Para sumar dos números reales extendidos usa el comando %%+. Si llamas a un comando con los argumentos equivocados es posible que en tu sistema ocurra un crash.

Para convertir números reales a números reales extendidos usa el comando %>%%. El comando que hace lo opuesto es %%>%. Para convertir un bint a número real (normal) el comando a usar es UNCOERCE, y el comando opuesto es COERCE. Abajo hay una lista de más comandos de conversión y otros comandos relacionados a número reales y a números reales extendidos.

3.1 Referencia

3.1.1 Números reales ya incorporados en ROM

**Direcc. Nombre Descripción**

2FB0A %-MAXREAL -9.99E499

2FAB1 %-9 -9

2FA9C %-8 -8

2FA87 %-7 -7

2FA72 %-6 -6

2FA5D %-5 -5

2FA48 %-4 -4

2FA33 %-3 -3

2FA1E %-2 -2

2FA09 %-1 -1

2FB34 %-MINREAL -1E-499

2F937 %0 0

2FB1F %MINREAL 1E-499

27118 %.1 .1

339BE %.5 .5

339D3 (%-.5) -.5

2F94C %1 1

270EE (%1.8) 1.8

2F961 %2 2

339A9 %e e

2F976 %3 3

2FAC6 %PI π

2F98B %4 4

2F9A0 %5 5

2F9B5 %6 6

2F9CA %7 7

2F9DF %8 8

2F9F4 %9 9

339E8 %10 10

2FCE6 %11 11

2FCFB %12 12

2FD10 %13 13

2FD25 %14 14

2FD3A %15 15

2FD4F %16 16

2FD64 %17 17

2FD79 %18 18

2FD8E %19 19

2FDA3 %20 20

2FDB8 %21 21

2FDCD %22 22

2FDE2 %23 23

2FDF7 %24 24

2FE0C %25 25

2FE21 %26 26

2FE36 %27 27

**Direcc. Nombre Descripción**

2FE4B (%28) 28

2FE60 (%29) 29

2FE75 (%30) 30

2FE8A (%31) 31

2FE9F (%32) 32

2FEB4 (%33) 33

2FEC9 (%34) 34

2FEDE (%35) 35

27103 %80 80

27E5D %100 100

339FD %180 180

33A12 (%200) 200

33A3C (%400) 400

33A27 %360 360

2FC7D (%1200) 1200

2FC92 (%2400) 2400

2FCA7 (%4800) 4800

0CF0B5 (~%TICKSsec) 8192

2FCBC (%9600) 9600

2FCD1 (%15360) 15360

0CD0B5 (~%TICKSmin) 491520

0CB0B5 (~%HrTicks) 29491200

0C90B5 ROMPTR 0B5 0C9 707788800

0C70B5 (~%TICKSweek) 4954521600

2FAF5 %MAXREAL 9.99E499

2F180 1REV ( 🡺 6.28318530718 )

( 🡺 360. )

( 🡺 400. )

Retorna el ángulo de una revolución, correspondiente al modo angular actual.

3.1.2 Números reales extendidos ya incorporados en ROM

**Direcc. Nombre Descripción**

2FB49 %%0 0

2FBE5 %%.1 0.1

30DC8 %%.4 0.4

2FBFF %%.5 0.5

2DA11 cfF 0.555555555555556

Usado como factor de conversión de ºF a Kelvin.

2FB63 %%1 1

2DA2B cfC 1

Usado como factor de conversión de ºC a Kelvin.

2FB7D %%2 2

2FB97 %%3 3

2FADB %%PI π

30017 PI/180 π /180

2FBB1 %%4 4

2FBCB %%5 5

27A89 %%2PI 2π

30BEA %%7 7

**Direcc. Nombre Descripción**

2FC19 %%10 10

30CC7 %%12 12

30CEB %%60 60

3.1.3 Manipulación de pila combinado con Reales

**Direcc. Nombre Descripción**

282CC (DROP%0) ( ob 🡺 %0 )

3.1.4 Conversión

**Direcc. Nombre Descripción**

2FFAC %>%% ( % 🡺 %% )

35ECA %>%%SWAP ( ob % 🡺 %% ob )

2FF9B %%>% ( %% 🡺 % )

30E47 2%>%% ( % % 🡺 %% %% )

30E5B 2%%>% ( %% %%' 🡺 % %' )

262F6 UNCOERCE ( # 🡺 % )

3F495 UNCOERCE2 ( # # 🡺 % % )

36BFA UNCOERCE%% ( # 🡺 %% )

2EFCA HXS>% ( hxs 🡺 % )

05D2C C%>% ( C% 🡺 %re %im )

2B3FD %IP># ( % 🡺 #ABS(IP(%)) )

Para un número real, toma su parte entera. Luego halla su valor absoluto y finalmente lo convierte a bint.

0F6006 ˆZ>R ( Z 🡺 % )

Convierte entero a real.

18A006 ˆZ2%% ( Z 🡺 %% )

Convierte entero a real extendido.

197006 ˆOBJ2REAL ( Z/% 🡺 % )

Si el argumento es un entero, lo convierte a real.

Si el argumento es un real, no hace nada.

Para otros objetos, genera el error “Argumento incorrecto”.

3.1.5 Operaciones con Reales

**Direcc. Nombre Descripción**

3035F %+ ( % %' 🡺 %+%' )

25E69 %+SWAP ( ob % %' 🡺 %+%' ob )

26F36 %1+ ( % 🡺 %+1 )

3036C %- ( % %' 🡺 %-%' )

26F4A %1- ( % 🡺 %-1 )

30346 %>%%- ( % %' 🡺 %%-%%' )

303A7 %\* ( % %' 🡺 %\*%' )

35C18 %10\* ( % 🡺 %\*10 )

303E9 %/ ( % %' 🡺 %/%' )

3045B %ˆ ( % %' 🡺 %ˆ%' )

302EB %ABS ( % 🡺 %' )

3030B %CHS ( % 🡺 -% )

302C2 %SGN ( % 🡺 -1/0/1 )

3049A %1/ ( % 🡺 1/% )

30489 %>%%1/ ( % 🡺 1/%% )

304F4 %SQRT ( % 🡺 √% )

**Direcc. Nombre Descripción**

304E1 %>%%SQRT ( % 🡺 √%% )

3051A %EXP ( % 🡺 eˆ% )

3052D %EXPM1 ( % 🡺 eˆ%-1 )

30559 %LN ( % 🡺 LN% )

30592 %LNP1 ( % 🡺 LN(%+1) )

3056C %LOG ( % 🡺 LOG% )

305A5 %ALOG ( % 🡺 10ˆ% )

305DA %SIN ( % 🡺 SIN% )

3062B %COS ( % 🡺 COS% )

3067C %TAN ( % 🡺 TAN% )

306AC %ASIN ( % 🡺 ASIN% )

306DC %ACOS ( % 🡺 ACOS% )

3070C %ATAN ( % 🡺 ATAN% )

30799 %SINH ( % 🡺 SINH% )

307C5 %COSH ( % 🡺 COSH% )

307D8 %TANH ( % 🡺 TANH% )

307EB %ASINH ( % 🡺 ASINH% )

307FE %ACOSH ( % 🡺 ACOSH% )

30811 %ATANH ( % 🡺 ATANH% )

3031B %MANTISSA ( % 🡺 %mant )

Similar al comando **MANT** de User RPL.

30824 %EXPONENT ( % 🡺 %expn )

Similar al comando **XPON** de User RPL.

30938 %FP ( % 🡺 %frac )

Parte decimal de un número real.

3094B %IP ( % 🡺 %int )

Parte entera de un número real.

30971 %FLOOR ( % 🡺 %maxint <=% )

Máximo entero menor o igual al argumento.

3095E %CEIL ( % 🡺 %minint >=% )

Mínimo entero mayor o igual al argumento.

305C7 %MOD ( % %' 🡺 %residuo )

Residuo de una división.

30723 %ANGLE ( %x %y 🡺 %ang )

Dadas las coordenadas rectangulares de un punto, devuelve el ángulo que forma el radio vector de este punto con el eje +x usando el actual formato de ángulo (rad, deg, grad).

El ángulo devuelto estará en <-180º;180º]

Similar al comando **ARG** de User RPL.

30746 %>%%ANGLE ( %x %y 🡺 %%ang )

Similar al comando %ANGLE pero devuelve un real extendido.

30F14 RNDXY ( % %lugares 🡺 %' )

Redondea el número real %, según %lugares.

Si %lugares es positivo, entonces es número de decimales.

Si %lugares es negativo, es número de cifras significativas.

30F28 TRCXY ( % %lugares 🡺 %' )

Trunca el número real %, según %lugares.

Si %lugares es positivo, entonces es número de decimales.

Si %lugares es negativo, es número de cifras significativas.

3084D %COMB ( % %' 🡺 COMB(%,%') )

30860 %PERM ( % %' 🡺 PERM(%,%') )

**Direcc. Nombre Descripción**

30837 %NFACT ( % 🡺 %! )

Calcula el factorial de un número.

El argumento puede ser: 0, 1, 2, 3, …, 253

30AAF %FACT ( % 🡺 gamma(%+1) )

Calcula gamma(x+1).

El argumento puede ser cualquier número real, excepto los enteros negativos. R – { -1;-2;-3;-4;......}

Cuando el argumento es: 0, 1, 2, 3, …, 253, coincide con el factorial del número.

3046C %NROOT ( % %n 🡺 %' )

Calcula la raíz enésima de un número real.

Equivalente al comando **XROOT** de User RPL.

300F9 %MIN ( % %' 🡺 %menor )

300E0 %MAX ( % %' 🡺 %mayor )

35DBC %MAXorder ( % %' 🡺 %max %min )

309AD %RAN ( 🡺 %aleatorio )

Retorna el siguiente número aleatorio.

30A2F %RANDOMIZE ( %semilla 🡺 )

Establece el número dado como la semilla generadora de números aleatorios. Si el argumento es cero, la semilla generadora será la hora del sistema.

Equivalente al comando **RDZ** de User RPL.

30A66 DORANDOMIZE ( % semilla 🡺 )

Establece el número dado como la semilla generadora de números aleatorios.

054003 FLASHPTR 003 054 ( 🡺 %aleatorio )

Retorna un número aleatorio del conjunto {-9.,-8.,….,8.,9.}

Aquí, el cero tiene doble probabilidad de ocurrir.

303B4 %OF ( % %' 🡺 % \* %' / 100 )

303F6 %T ( % %' 🡺 %'/% \* 100 )

3041B %CH ( % %' 🡺 %'/% \* 100 – 100 )

3000D %D>R ( %sexagesimales 🡺 %radianes )

30040 %R>D ( %radianes 🡺 %sexagesimales )

30E79 %REC>%POL ( %x %y 🡺 %r %ángulo )

Convierte coordenadas rectangulares a polares usando el actual formato de ángulo (rad, deg, grad).

El ángulo devuelto estará en <-180º;180º]

30EA6 %POL>%REC ( %r %ángulo 🡺 %x %y )

Convierte coordenadas polares a rectangulares. El argumento %ángulo debe estar en el actual formato de ángulo (rad, deg, grad). El argumento %ángulo puede ser cualquier real.

30EDD %SPH>%REC ( %r %tetha %phi 🡺 %x %y %z )

Convierte coordenadas esféricas a rectangulares. Los argumentos %theta y %phi deben de estar en el actual formato de ángulo (rad, deg, grad).

3.1.6 Operaciones con Reales Extendidos

**Direcc. Nombre Descripción**

3032E %%+ ( %% %%' 🡺 %%+%%' )

3033A %%- ( %% %%' 🡺 %%-%%' )

30385 %%\* ( %% %%' 🡺 %%\*%%' )

3602F %%\*ROT ( ob ob' %% %%' 🡺 ob' %%+%%' ob )

**Direcc. Nombre Descripción**

35EDE %%\*SWAP ( ob %% %%' 🡺 %%+%%' ob )

36C7C %%\*UNROT ( ob ob' %% %%' 🡺 %%+%%' ob ob' )

303D3 %%/ ( %% %%' 🡺 %%/%%' )

36C22 SWAP%%/ ( %% %%' 🡺 %%'/%% )

36BE6 %%/>% ( %% %%' 🡺 % )

3044A %%ˆ ( %% %%' 🡺 %%ˆ%%' )

51D006 ˆCK%%SQRT ( %% 🡺 %%/C%% )

Halla la raíz cuadrada de un número real extendido.

Si el argumento es no negativo, el resultado es real extendido.

Si el argumento es negativo, el resultado es un complejo extendido y la calculadora cambia a modo complejo.

304D5 %%SQRT ( %% 🡺 √%% )

3047D %%1/ ( %% 🡺 1/%% )

305F1 %%SIN ( %% 🡺 %%sin )

30642 %%COS ( %% 🡺 %%cos )

30612 %%SINRAD ( %%rad 🡺 %%sin )

Halla el seno de un ángulo dado en radianes.

No importa el actual formato de ángulo.

30602 %%SINDEG ( %%deg 🡺 %%sin )

Halla el seno de un ángulo dado en sexagesimales.

No importa el actual formato de ángulo.

30663 %%COSRAD ( %%rad 🡺 %%cos )

Halla el coseno de un ángulo dado en radianes.

No importa el actual formato de ángulo.

30653 %%COSDEG ( %%deg 🡺 %%cos )

Halla el coseno de un ángulo dado en sexagesimales.

No importa el actual formato de ángulo.

30693 %%TANRAD ( %%rad 🡺 %%tan )

Halla la tangente de un ángulo dado en radianes.

No importa el actual formato de ángulo.

2D817 (%%TANDEG) ( %%deg 🡺 %%tan )

Halla la tangente de un ángulo dado en sexagesimales.

No importa el actual formato de ángulo.

306C3 %%ASINRAD ( %% 🡺 %%rad )

Halla el arco seno de un número en el intervalo [-1;1]. Devuelve el resultado en radianes.

No importa el actual formato de ángulo.

306F3 %%ACOSRAD ( %% 🡺 %%rad )

Halla el arco seno de un número en el intervalo [-1;1]. Devuelve el resultado en sexagesimales.

No importa el actual formato de ángulo.

30780 %%SINH ( %% 🡺 %%sinh )

307B2 %%COSH ( %% 🡺 %%cosh )

30E83 %%R>P ( %%x %%y 🡺 %%radio %%angulo )

Convierte coordenadas rectangulares a polares usando el actual formato de ángulo (rad, deg, grad).

El ángulo devuelto estará en <-180º;180º]

30EB0 %%P>R ( %%r %%ángulo 🡺 %%x %%y )

Convierte coordenadas polares a rectangulares. El argumento %%ángulo debe estar en el actual formato de ángulo (rad, deg, grad). El argumento %ángulo puede ser cualquier real.

**Direcc. Nombre Descripción**

3073A %%ANGLE ( %%x %%y 🡺 %%ang )

Equivalente al comando %ANGLE, pero para reales extendidos.

30767 %%ANGLERAD ( %%x %%y 🡺 %%rad )

Equivale a %%ANGLE, pero el resultado siempre en radianes.

30757 %%ANGLEDEG ( %%x %%y 🡺 %%deg )

Equivale a %%ANGLE, pero el resultado siempre en sexagesimales.

302DB %%ABS ( %% 🡺 %%abs )

302FB %%CHS ( %% 🡺 -%% )

30507 %%EXP ( %% 🡺 eˆ%% )

30546 %%LN ( %% 🡺 ln %% )

3057F %%LNP1 ( %% 🡺 %%ln(%%+1) )

30984 %%FLOOR ( %% 🡺 %%maxint )

aka: %%INT

300C7 %%MAX ( %% %%' 🡺 %%max )

3.1.7 Tests

**Direcc. Nombre Descripción**

302AC %= ( % %' 🡺 flag )

302B7 %<> ( % %' 🡺 flag )

3025C %< ( % %' 🡺 flag )

302A1 %<= ( % %' 🡺 flag )

30275 %> ( % %' 🡺 flag )

3028B %>= ( % %' 🡺 flag )

30156 %0= ( % 🡺 flag )

36C0E DUP%0= ( % 🡺 flag )

301BA %0<> ( % 🡺 flag )

Se puede usar para cambiar un flag de User RPL a flag de System RPL.

30123 %0< ( % 🡺 flag )

30184 %0> ( % 🡺 flag )

301E2 %0>= ( % 🡺 flag )

3020A %%< ( %% %%' 🡺 flag )

30296 %%<= ( %% %%' 🡺 flag )

3026A %%> ( %% %%' 🡺 flag )

30280 %%>= ( %% %%' 🡺 flag )

30145 %%0= ( %% 🡺 flag )

301A6 %%0<> ( %% 🡺 flag )

30112 %%0< ( %% 🡺 flag )

301F6 %%0<= ( %% 🡺 flag )

30173 %%0> ( %% 🡺 flag )

301CE %%0>= ( %% 🡺 flag )

Capítulo 4  
Números complejos

Los números complejos pueden ser ingresados en tus programas con la siguiente estructura: C% <real> <imag>. La parte real y la parte imaginaria son números reales en notación decimal. Si en la pila tienes la parte real y la parte imaginaria, el comando %>C% creará un número complejo a partir de esos dos números. El comando C%>% toma un número complejo y devuelve la parte real y la parte imaginaria en la pila.

También existen los Números Complejos Extendidos (llamados también largos), los cuales no son accesibles directamente en User RPL. Estos son números complejos cuyas partes real e imaginaria son números reales extendidos. Estos pueden ser ingresados en tu programa con la siguiente estructura: C%% <real> <imag>, donde las partes real e imaginaria son reales extendidos. En la pila se muestran como los números complejos normales, pero siempre en notación científica.

4.1 Referencia

4.1.1 Números Complejos ya incorporados en ROM

**Direcc. Nombre Descripción**

27DE4 C%0 (0,0)

27E09 C%1 (1,0)

27DBF C%-1 (-1,0)

27E2E C%%1 (%%1,%%0)

4.1.2 Conversión

**Direcc. Nombre Descripción**

261D9 C%%>C% ( C%% 🡺 C% )

05C27 %>C% ( %re %im 🡺 C% )

362F2 SWAP%>C% ( %im %re 🡺 C% )

261FC Re>C% ( %re 🡺 C% )

25E9C C>Re% ( C% 🡺 %re )

25E9B C>Im% ( C% 🡺 %im )

18C006 ˆE%%>C%% ( %%re %%im 🡺 C%% )

Convierte dos reales extendidos a un complejo extendido.

261CF %%>C% ( %%re %%im 🡺 C% )

25E82 C%>%% ( C% 🡺 %%re %%im )

25E83 C%>%%SWAP ( C% 🡺 %%im %%re )

05DBC C%%>%% ( C%% 🡺 %%re %%im )

188006 ˆC2C%% ( C 🡺 C%% )

Convierte entero gaussiano a complejo extendido.

189006 ˆZZ2C%%ext ( Zre Zim 🡺 C%% )

Crea un complejo extendido a partir de dos enteros.

18B006 ˆC%>C%% ( C% 🡺 C%% )

Convierte complejo a complejo extendido.

15E006 ˆRIXCext ( Zre Zim 🡺 C )

Crea un entero gausiano.

15F006 ˆIRXCext ( Zim Zre 🡺 C )

Crea un entero gausiano.

4.1.3 Operaciones con Complejos

**Direcc. Nombre Descripción**

25E8F C%CˆC ( C% C%' 🡺 C%'' )

25E90 C%CˆR ( C% % 🡺 C%' )

25E94 C%RˆC ( % C% 🡺 C%' )

25E84 C%ABS ( C% 🡺 % )

Módulo del número complejo.

50C006 ˆCZABS ( C% 🡺 C% )

( C%% 🡺 %% )

( symb 🡺 symb/Z/% )

Módulo del número complejo. También funciona para números complejos escritos en forma simbólica.

261ED C%CHS ( C% 🡺 -C% )

25E81 C%1/ ( C% 🡺 1/C% )

25E98 C%SQRT ( C% 🡺 √C% )

25E95 C%SGN ( C% 🡺 C%/C%ABS )

**Direcc. Nombre Descripción**

261F2 C%CONJ ( C% 🡺 C%' )

25E88 C%ARG ( C% 🡺 % )

Halla el argumento del complejo en el actual formato de ángulo (rad, deg, grad). El ángulo devuelto está en el intervalo <-180º;180º]

25E91 C%EXP ( C% 🡺 eˆC% )

25E92 C%LN ( C% 🡺 ln C% )

25E93 C%LOG ( C% 🡺 log C% )

25E87 C%ALOG ( C% 🡺 10ˆC% )

25E96 C%SIN ( C% 🡺 sin C% )

25E8D C%COS ( C% 🡺 cos C% )

25E99 C%TAN ( C% 🡺 tan C% )

25E89 C%ASIN ( C% 🡺 asin C% )

25E85 C%ACOS ( C% 🡺 acos C% )

25E8B C%ATAN ( C% 🡺 atan C% )

25E97 C%SINH ( C% 🡺 sinh C% )

25E8E C%COSH ( C% 🡺 cosh C% )

25E9A C%TANH ( C% 🡺 tanh C% )

25E8A C%ASINH ( C% 🡺 asinh C% )

25E86 C%ACOSH ( C% 🡺 acosh C% )

25E8C C%ATANH ( C% 🡺 atanh C% )

261DE C%%CHS ( C%% 🡺 -C%% )

261E3 C%%CONJ ( C%% 🡺 C%%' )

515006 ˆARG2 ( im re 🡺 meta )

Retorna un meta que representa al argumento.

517006 ˆQUADRANT ( re im ?re>0 ?im>0 🡺 newre newim % )

Retorna Z0 Z1 Z-2 ó Z-1 de tal manera que el argumento del correspondiente número complejo sea Z\*π/2 + theta donde Ө está en el intervalo [0, π/2].

51E006 ˆC%%SQRT ( C%% 🡺 C%%' )

Retorna una de las raíces cuadradas del número complejo.

4.1.4 Tests

**Direcc. Nombre Descripción**

261E8 C%0= ( C% 🡺 flag )

261D4 C%%0= ( C%% 🡺 flag )

Capítulo 5  
Enteros (ZINTS)

Este tipo de objeto está presente a partir de la HP 49. Los números enteros (llamados también ZINTS) son objetos que pueden representar a números muy grandes. En la mayoría de los casos, no es necesario preocuparse si el usuario ingresó numeros enteros como argumentos para un programa El mecanismo de verificación de tipos de objetos (descrito en la sección 30.2) convertirá los enteros a reales la mayoría de veces.

Si quieres manejar a los números enteros como tales (sin convertirlos a reales) hay varios comandos que manipulan números enteros. Ya que los números enteros en realidad forman parte del CAS de la HP 50g, estos comandos no son descritos aquí. En vez de eso, el capítulo 45 es el lugar donde podrás ver la documentación sobre los comandos que manejan a los ZINTs.

Para escribir un entero en la pila escribe la palabra ZINT.

Por ejemplo, el siguiente código:

:: ZINT 45 ZINT -49 ;

Coloca los enteros 45 y -49 en la pila.

Capítulo 6  
Caracteres y Cadenas

Caracteres y cadenas son dos tipos de datos que tienen texto.

Los **caracteres** no son accesibles directamente en User RPL. Los caracteres son objetos que sólo pueden tener un carácter. Puedes crearlos usando la estructura: CHR <char> o usando uno de los caracteres ya incorporados en ROM listados abajo. Por ejemplo, en el editor de Debug 4x, para escribir el carácter F puedes escribir de cualquiera de estas dos formas:

CHR F

CHR \46

Sin embargo, sólo la primera forma es posible cuando programas en la calculadora.

Para convertir un carácter a bint, usa CHR>#. El bint devuelto es el código ASCII para el carácter. El comando opuesto es #>CHR.

Las **cadenas** (o strings) son insertadas en tu programa con $ "<cadena>", o simplemente "<cadena>". Hay algunas cadenas ya incorporadas que se listan abajo.

Es posible convertir un carácter en cadena con el comando CHR>$.

Dos comandos útiles que tratan con cadenas son LEN$ y &$. El primero retorna la longitud (número de caracteres) de una cadena como un bint, y el segundo concatena dos cadenas. Para conseguir una subcadena (parte de una cadena) usa el comando SUB$. Este comando toma tres argumentos: la cadena original, la posición inicial (un bint) y la posición final (también un bint). Todo lo que esté entre la posición inicial y final (inclusive) será retornado.

Otro comando es POS$, el cual busca dentro de una cadena (en nivel 3) por un carácter o cadena (en nivel 2), empezando desde una posición especificada (en nivel 1). La posición de la primera ocurrencia de la cadena buscada en la otra cadena es retornada (como un bint) en el nivel 1. Si esta no puede ser encontrada devuelve #0.

6.1 Referencia

6.1.1 Caracteres ya incorporados en ROM

**Direcc. Nombre Descripción**

33D2B CHR\_00 '**\00**' (carácter 0d 00h)

El carácter NULO.

33F77 CHR\_Newline '**\0a**' (carácter 10d 0Ah)

El carácter NUEVA\_LÍNEA.

33D32 CHR\_... '**…**' (carácter 31d 1Fh

33F93 CHR\_Space '' (carácter 32d 20h)

El carácter ESPACIO.

33D39 CHR\_DblQuote '**"**' (carácter 34d 22h)

33D40 CHR\_# '**#**' (carácter 35d 23h)

33F70 CHR\_LeftPar '**(**' (carácter 40d 28h)

33F85 CHR\_RightPar '**)**' (carácter 41d 29h)

33D47 CHR\_\* '**\***' (carácter 42d 2Ah)

33D4E CHR\_+ '**+**' (carácter 43d 2Bh)

33D55 CHR\_, '**,**' (carácter 44d 2Ch)

33D5C CHR\_- '**-**' (carácter 45d 2Dh)

33D63 CHR\_. '**.**' (carácter 46d 2Eh)

33D6A CHR\_/ '**/**' (carácter 47d 2Fh)

33D71 CHR\_0 '**0**' (carácter 48d 30h)

33D78 CHR\_1 '**1**' (carácter 49d 31h)

33D7F CHR\_2 '**2**' (carácter 50d 32h)

33D86 CHR\_3 '**3**' (carácter 51d 33h)

33D8D CHR\_4 '**4**' (carácter 52d 34h)

33D94 CHR\_5 '**5**' (carácter 53d 35h)

33D9B CHR\_6 '**6**' (carácter 54d 36h)

33DA2 CHR\_7 '**7**' (carácter 55d 37h)

33DA9 CHR\_8 '**8**' (carácter 56d 38h)

33DB0 CHR\_9 '**9**' (carácter 57d 39h)

33DB7 CHR\_: '**:**' (carácter 58d 3Ah)

33DBE CHR\_; '**;**' (carácter 59d 3Bh)

33DC5 CHR\_< '**<**' (carácter 60d 3Ch)

33DCC CHR\_= '**=**' (carácter 61d 3Dh)

33DD3 CHR\_> '**>**' (carácter 62d 3Eh)

33DDA CHR\_A '**A**' (carácter 65d 41h)

33DE1 CHR\_B '**B**' (carácter 66d 42h)

33DE8 CHR\_C '**C**' (carácter 67d 43h)

33DEF CHR\_D '**D**' (carácter 68d 44h)

33DF6 CHR\_E '**E**' (carácter 69d 45h)

33DFD CHR\_F '**F**' (carácter 70d 46h)

33E04 CHR\_G '**G**' (carácter 71d 47h)

33E0B CHR\_H '**H**' (carácter 72d 48h)

33E12 CHR\_I '**I**' (carácter 73d 49h)

33E19 CHR\_J '**J**' (carácter 74d 4Ah)

33E20 CHR\_K '**K**' (carácter 75d 4Bh)

33E27 CHR\_L '**L**' (carácter 76d 4Ch)

33E2E CHR\_M '**M**' (carácter 77d 4Dh)

33E35 CHR\_N '**N**' (carácter 78d 4Eh)

33E3C CHR\_O '**O**' (carácter 79d 4Fh)

33E43 CHR\_P '**P**' (carácter 80d 50h)

33E4A CHR\_Q '**Q**' (carácter 81d 51h)

33E51 CHR\_R '**R**' (carácter 82d 52h)

**Direcc. Nombre Descripción**

33E58 CHR\_S '**S**' (carácter 83d 53h)

33E5F CHR\_T '**T**' (carácter 84d 54h)

33E66 CHR\_U '**U**' (carácter 85d 55h)

33E6D CHR\_V '**V**' (carácter 86d 56h)

33E74 CHR\_W '**W**' (carácter 87d 57h)

33E7B CHR\_X '**X**' (carácter 88d 58h)

33E82 CHR\_Y '**Y**' (carácter 89d 59h)

33E89 CHR\_Z '**Z**' (carácter 90d 5Ah)

33FA1 CHR\_[ '**[**' (carácter 91d 5Bh)

33FA8 CHR\_] '**]**' (carácter 93d 5Dh)

33F9A CHR\_UndScore '**\_**' (carácter 95d 5Fh)

33E90 CHR\_a '**a**' (carácter 97d 61h)

33E97 CHR\_b '**b**' (carácter 98d 62h)

33E9E CHR\_c '**c**' (carácter 99d 63h)

33EA5 CHR\_d '**d**' (carácter 100d 64h)

33EAC CHR\_e '**e**' (carácter 101d 65h)

33EB3 CHR\_f '**f**' (carácter 102d 66h)

33EBA CHR\_g '**g**' (carácter 103d 67h)

33EC1 CHR\_h '**h**' (carácter 104d 68h)

33EC8 CHR\_i '**i**' (carácter 105d 69h)

33ECF CHR\_j '**j**' (carácter 106d 6Ah)

33ED6 CHR\_k '**k**' (carácter 107d 6Bh)

33EDD CHR\_l '**l**' (carácter 108d 6Ch)

33EE4 CHR\_m '**m**' (carácter 109d 5Dh)

33EEB CHR\_n '**n**' (carácter 110d 6Eh)

33EF2 CHR\_o '**o**' (carácter 111d 6Fh)

33EF9 CHR\_p '**p**' (carácter 112d 70h)

33F00 CHR\_q '**q**' (carácter 113d 71h)

33F07 CHR\_r '**r**' (carácter 114d 72h)

33F0E CHR\_s '**s**' (carácter 115d 73h)

33F15 CHR\_t '**t**' (carácter 116d 74h)

33F1C CHR\_u '**u**' (carácter 117d 75h)

33F23 CHR\_v '**v**' (carácter 118d 76h)

33F2A CHR\_w '**w**' (carácter 119d 77h)

33F31 CHR\_x '**x**' (carácter 120d 78h)

33F38 CHR\_y '**y**' (carácter 121d 79h)

33F3F CHR\_z '**z**' (carácter 122d 7Ah)

33FAF CHR\_{ '**{**' (carácter 123d 7Bh)

33FB6 CHR\_} '**{**' (carácter 125d 7Dh)

33F5B CHR\_Angle '**<**' (carácter 128d 80h)

33F69 CHR\_Integral '**∫**' (carácter 132d 84h)

33F62 CHR\_Deriv '**∂**' (carácter 136d 88h)

33F46 CHR\_-> '**→**' (carácter 141d 8Dh)

33F4D CHR\_<< '**«**' (carácter 171d ABh)

33F54 CHR\_>> '**»**' (carácter 187d BBh)

33F7E CHR\_Pi '**π**' (carácter 135d 87h)

33F8C CHR\_Sigma '**∑**' (carácter 133d 85h)

33FBD CHR\_<= '**≤**' (carácter 137d 89h)

33FC4 CHR\_>= '**≥**' (carácter 138d 8Ah)

33FCB CHR\_<> '**≠**' (carácter 139d 8Bh)

6.1.2 Cadenas ya incorporadas en ROM

**Direcc. Nombre Descripción**

055DF NULL$ ""

Cadena vacía.

33B55 SPACE$ " "

aka: tok\_

33B39 NEWLINE$ "**\0a**"

Nueva línea.

27195 CRLF$ "**\0d\0a**"

Retorno de carro y nueva línea.

33ABF tokESC "**<ESC>**"

Carácter ESCAPE. "\1B"

33D1F ($\_...) "**…**"

33B79 tokquote "**"**"

Una comilla doble.

33A8F toksharp "**#**"

33AA7 (tok$) "**$**"

33AB3 (tok&) "**&**"

33B85 tok' "**'**"

Una comilla simple.

33BB5 (toklparen) "**(**"

33BC1 (tokrparen) "**)**"

33BD9 (tok\*) "**\***"

33BF1 (tok+) "**+**"

33B91 tok, "**,**"

33BFD tok- "**-**"

33B9D tok. "**.**"

33BE5 (tok/) "**/**"

33C4D tok0 "**0**"

33C59 tok1 "**1**"

33C65 (tok2) "**2**"

33C71 (tok3) "**3**"

33C7D (tok4) "**4**"

33C89 (tok5) "**5**"

33C95 (tok6) "**6**"

33CA1 (tok7) "**7**"

33CAD tok8 "**8**"

33CB9 tok9 "**9**"

2723F (tok:) "**:**"

33BA9 (tok;) "**;**"

33C09 tok= "**=**"

2D933 (tok?) "**?**"

2D88D (tokA) "**A**"

33AE3 tokexponent "**E**"

2D8CD (tokK) "**K**"

33A6B (tok[) "**[**"

33A51 (tok]) "**]**"

33BCD (tokˆ) "**ˆ**"

33A9B (tokuscore) "**\_**"

2724B (tok`) "**`**"

2D848 tok\_g "**g**"

2D86D tok\_m "**m**"

2D7B3 (tokr) "**r**"

2D8AD tok\_s "**s**"

33A77 tok{ "**{**"

**Direcc. Nombre Descripción**

33B07 (tokWHERE) "**|**"

33A83 (tok}) "**}**"

33AEF (tokanglesign) "**∠**"

33C15 (tokSQRT) "**√**"

33AFB (tokSIGMA) "**∑**"

33C21 (tokDER) "**∂**"

272D9 tok-> "**🡪**"

33AD7 tok<< "**«**"

33ACB (tok>>) "**»**"

34002 $\_<<>> "**«»**"

34010 $\_{} "**{}**"

3401E $\_[] "**[]**"

34048 $\_LRParens "**()**"

3402C $\_'' "**''**"

Dos comillas simples.

34056 $\_2DQ "**""**"

Dos comillas dobles.

3403A $\_:: "**::**"

33C2D (tokCTGROB) "**GROB**"

33C3F (tokCTSTR) "**C$**"

33B45 ($DER) "**der**"

33B61 (tokUNKNOWN) "**UNKNOWN**"

340A4 $\_RAD "**RAD**"

340B4 $\_GRAD "**GRAD**"

33FF2 $\_XYZ "**XYZ**"

33FE2 $\_R<Z "**R∠Z**"

33FD2 $\_R<< "**R∠∠**"

34076 $\_EXIT "**EXIT**"

34088 $\_Undefined "**Undefined**"

34064 $\_ECHO "**ECHO**"

2722F (tokDIR) "**DIR**"

272C7 (tokTHEN) "**THEN**"

27257 (tokELSE) "**ELSE**"

27269 (tokEND) "**END**"

272A3 (tokNEXT) "**NEXT**"

272B5 (tokSTEP) "**STEP**"

27279 (tokUNTIL) "**UNTIL**"

2728D (tokREPEAT) "**REPEAT**"

27221 (tokTO) "**TO**"

27F4C (<Del$) "**🡪DEL**"

27EB4 (<Skip$) "**🡪SKIP**"

27F9F (>Del$) "**DEL🡪**"

27F00 (>Skip$) "**SKIP🡪**"

37F5C (tokIF-prompt) "**IF-prompt**"

3DFB3 (tokSlope) "**Slope**"

3DF97 (tokIntercept) "**Intercept**"

2D8ED (tokcd) "**cd**"

2D7FF (tokdegR) "**ºR**"

2D8ED (tokcd) "**cd**"

2D7D3 (toksr) "**sr**"

2D90F (tokmol) "**mol**"

33B13 (14SPACES$) " "

Cadena de 14 espacios.

6.1.3 Cadenas presentes en ROM con manipulación de pila.

**Direcc. Nombre Descripción**

35D94 NULL$SWAP ( ob 🡺 $ ob )

NULL$, luego SWAP.

04D3E DROPNULL$ ( ob 🡺 NULL$ )

DROP luego NULL$.

25EEC NULL$TEMP ( 🡺 $ )

Crea cadena vacía en memoria temporal (hace NULL$, luego TOTEMPOB).

6.1.4 Conversión

**Direcc. Nombre Descripción**

25F77 #>$ ( # 🡺 $ )

Convierte un bint en cadena (en base decimal).

Por ejemplo:

El programa :: BINT83 #>$ ; retorna "83"

El programa :: # FF #>$ ; retorna "255"

El programa :: 849 #>$ ; retorna "849"

El programa :: % 849. COERCE #>$ ; retorna "849"

25F72 #:>$ ( # 🡺 "#: " )

Convierte un bint a cadena (en base decimal) y agrega dos puntos y un espacio. Por ejemplo:

El programa :: BINT8 #:>$ ; retorna "8: "

25F0F a%>$ ( % 🡺 $ )

Convierte un número real a cadena usando el actual formato de número. Por ejemplo, este programa:

:: DOSTD 12362. a%>$

BINT2 DOFIX 12362. a%>$

BINT2 DOENG 12362. a%>$

BINT2 DOSCI 12362. a%>$

;

Devuelve: "12362." "12,362.00" "12.4E3" "1.24E4"

aka: a%>$,

2F019 UNIT>$ ( u 🡺 $ )

Convierte unidad a cadena sin comillas.

05BE9 ID>$ ( id/lam 🡺 $ )

Convierte nombre global o local en cadena sin comillas.

25EB3 DOCHR ( % 🡺 $ )

Dado un número real como código ASCII, devuelve una cadena con un solo carácter.

Equivale al comando **CHR** de User RPL.

Por ejemplo el programa :: % 65. DOCHR ; retorna "A"

0F1006 ˆZ>S ( Z 🡺 $ )

Convierte entero en cadena.

2EFC1 hxs>$ ( hxs 🡺 $ )

Convierte a cadena usando base y ancho de palabra actuales.

2EFC0 HXS>$ ( hxs 🡺 $ )

Hace hxs>$ y luego agrega el carácter de la base actual.

6.1.5 Manejo de cadenas y caracteres

**Direcc. Nombre Descripción**

05A75 #>CHR ( # 🡺 chr )

Dado un bint como código ASCII, devuelve el carácter respectivo.

37AA5 CHR>$ ( chr 🡺 $ )

Convierte un carácter en cadena.

05636 LEN$ ( $ 🡺 #longitud )

Retorna el número de caracteres que contiene la cadena.

357E2 DUPLEN$ ( $ 🡺 $ # )

DUP luego LEN$.

05622 OVERLEN$ ( $ ob 🡺 $ ob #len )

OVER luego LEN$.

361DA NEWLINE$&$ ( $ 🡺 "$\0A" )

Agrega salto de línea al final de la cadena.

aka: NEWLINE&$

2F31A APNDCRLF ( $ 🡺 "$\0D\0A" )

Agrega retorno de carro y salto de línea a la cadena.

050ED CAR$ ( $ 🡺 chr )

( "" 🡺 "" )

Retorna el primer carácter de una cadena como un carácter, o

NULL$ para una cadena vacía.

0516C CDR$ ( $ 🡺 $' )

( "" 🡺 "" )

Retorna una cadena sin el primer carácter, o NULL$ para una cadena vacía.

378FA POS$ ( $base $buscada #inicio 🡺 #pos )

( $base chr #inicio 🡺 #pos )

Busca la cadena $buscada (o el carácter chr) en la cadena $base, empezando en la posición #inicio. Retorna la posición de $buscada ó #0 si no es encontrada.

aka: POSCHR

37906 POS$REV ( $base $buscada #limite 🡺 #pos )

( $base chr #limite 🡺 #pos )

Busca la cadena $buscada (o el carácter chr) hacia atrás empezando en #limite hasta #1.

aka: POSCHRREV

25EA0 COERCE$22 ( $ 🡺 $' )

Si la cadena tiene más de 22 caracteres, es truncada a 21 caracteres y se le agrega "...".

2F16D Blank$ ( #longitud 🡺 $ )

Crea una cadena con el numero especificado de espacios.

2EEF0 PromptIdUtil ( id ob 🡺 $ )

Crea una cadena de la forma "id: ob".

25EF8 SEP$NL ( $ 🡺 $' $'' )

Separa una cadena en el primer salto de línea.

$' es la subcadena después del primer salto de línea.

$'' es la subcadena antes del primer salto de línea.

**Direcc. Nombre Descripción**

09A003 (ˆStrCutNchr) ( $ #ancho 🡺 $' )

Reemplaza algunos caracteres ESPACIO con caracteres SALTO DE LINEA para ajustar el texto al ancho #ancho.

Pero si hay en la cadena una palabra con más caracteres que #ancho, esta palabra NO será cortada.

Usado por el comando ViewStrObject.

Muy rápido (comando tipo bang).

09A003 (ˆStrCutNchr2) ( $ #ancho #LineasMax 🡺 $' #LineasFinal )

Reemplaza algunos caracteres ESPACIO con caracteres SALTO DE LINEA para ajustar el texto al ancho #ancho.

Si hay en la cadena una palabra con más caracteres que #ancho, esta palabra SI será cortada.

Se puede especificar cual será el número máximo de líneas que queremos que tenga la cadena.

05733 SUB$ ( $ #inicio #fin 🡺 $' )

Retorna la subcadena entre las posiciones especificadas.

2F2C0 (XEQSUB$) ( $ %inicio %fin 🡺 $' )

Retorna la subcadena entre las posiciones especificadas.

Hace COERCE2 luego SUB$.

Equivale al comando **SUB** de User RPL cuando en la pila se hallan los argumentos indicados.

3628E #1-SUB$ ( $ #inicio #fin+#1 🡺 $' )

Hace #1- y luego SUB$.

362A2 1\_#1-SUB$ ( $ #fin+#1 🡺 $' )

Retorna subcadena desde el primer carácter hasta el carácter antes de la posición especificada.

aka: 1\_#1-SUB

362B6 LAST$ ( $ #inicio 🡺 $' )

Retorna subcadena desde la posición inicial especificada hasta el final (inclusive).

362CA #1+LAST$ ( $ #inicio-#1 🡺 $' )

Retorna subcadena desde la posición siguiente a la especificada hasta el final (inclusive).

35DA8 SUB$SWAP ( ob $ # #' 🡺 $' ob )

SUB$ luego SWAP.

2A5CA SUB$1# ( $ # 🡺 #' )

Para el carácter de la posición especificada, retorna su código ASCII como un bint.

34C82 EXPAND ( $ #nibs 🡺 $' )

Agrega caracteres nulos a la cadena. Debido a que el argumento es el número de nibbles, tu debes agregar un número que sea el doble de la cantidad de caracteres nulos que deseas agregar. Por ejemplo este programa:

:: "RATA" BINT4 EXPAND ; retorna "RATA\00\00"

05193 &$ ( $ $' 🡺 $+$' )

Concatena dos cadenas.

35346 SWAP&$ ( $ $' 🡺 $'+$ )

Concatena dos cadenas.

36FF6 &$SWAP ( ob $ $' 🡺 $+$' ob )

&$ luego SWAP.

**Direcc. Nombre Descripción**

353CD !append$ ( $ $' 🡺 $+$' )

Intenta concatenar dos cadenas con &$.

Pero si no hay suficiente memoria, intentará concatenarlas de otro modo.

3533C !insert$ ( $ $' 🡺 $'+$ )

Hace SWAP luego !append$.

35F6A !append$SWAP ( ob $ $' 🡺 $+$' ob )

Hace !append$ luego SWAP.

0525B >H$ ( $ chr 🡺 $' )

Agrega el carácter al inicio de la cadena.

Ejemplo: :: "RAIZ" CHR Ñ >H$ ; retorna "ÑRAIZ"

052EE >T$ ( $ chr 🡺 $' )

Agrega el carácter al final de la cadena.

Ejemplo: :: "RAIZ" CHR Ñ >T$ ; retorna "RAIZÑ"

35BD7 APPEND\_SPACE ( $ 🡺 $' )

Agrega un espacio al final de la cadena.

2EED3 TIMESTR ( %fecha %hora 🡺 "día\_semana fecha hora" )

Retorna una cadena que representa el tiempo actual, usando los actuales formatos de hora y fecha. Ejemplo:

"WED 06/24/98 10:00:45A"

25E7C AND$ ( $ $' 🡺 $'' )

AND lógico. Las cadenas deben tener el mismo tamaño.

25EF0 OR$ ( $ $' 🡺 $'' )

OR lógico. Las cadenas deben tener el mismo tamaño.

25F0D XOR$ ( $ $' 🡺 $'' )

XOR lógico. Las cadenas deben tener el mismo tamaño.

556001 FLASHPTR 001 556 ( $base %posicion $reemplazo 🡺 $base' )

Coloca en la cadena $base a la cadena $reemplazo en la posición indicada con un número real.

El número real debe ser mayor o igual a uno. De lo contrario, genera el error “Argumento: valor incorr”

Equivale al comando **REPL** de User RPL.

01A00F FLASHPTR 00F 01A ( $base $buscada $reemplazo 🡺 $base' %veces )

Busca en la cadena $base todas las ocurrencias de $buscada y las reemplaza por $reemplazo.

Retorna la cadena modificada y el número de ocurrencias de $buscada en $base.

Equivale al comando **SREPL** de User RPL.

6.1.6 Parseando Cadenas

**Direcc. Nombre Descripción**

25EB7 DOSTR> ( $ 🡺 ? )

Intenta convertir una cadena a objeto.

Si se tiene éxito, el objeto es evaluado.

Si no se tiene éxito, genera el error “Sintaxis incorrntos”

Llama a palparse.

Equivale al comando **STR→** de User RPL.

2EF62 palparse ( $ 🡺 ob T )

( $ 🡺 $ #pos $' F )

Intenta convertir una cadena a objeto.

Si se tiene éxito, retorna el objeto y TRUE.

Si no se tiene éxito, agrega la posición del error (posición final del primer objeto incorrecto), la parte de la cadena que es incorrecta $' y FALSE.

Si la cadena contiene varios objetos, el resultado es un objeto programa que contiene a esos objetos.

00E004 ^algparse ( $ 🡺 ob T )

( $ 🡺 $ #pos #LongCadIncorr F )

Intenta convertir una cadena (modo algebraico) a objeto.

Si se tiene éxito, retorna un objeto etiquetado y TRUE.

Si no tiene éxito, agrega la posición del error (posición final del 1º objeto incorrecto), la longitud de la parte incorrecta y FALSE.

2EF6A Parse.1 ( $ 🡺 ob T )

( $ 🡺 $ $ #pos #Long F )

Este comando es usado cuando existe una línea de edición.

Al usar este comando, desaparecerá la línea de edición.

Parse.1 intenta convertir su contenido a objeto (usando palparse o ^algparse según el estado del flag 95).

Si se tiene éxito, retorna el objeto y TRUE. Además pone la cadena que había en la línea de edición a la lista accesible a través de la tecla CMD.

Si no tiene éxito, agrega la posición del error (posición final del 1º objeto incorrecto), la longitud de la parte incorrecta y FALSE.

2EF6F DispBadToken ( ob $ #FinSeleccion #TamañoSelección 🡺 )

Edita la cadena, con una parte de ella seleccionada (en fondo oscuro). Posiciona el cursor al final de la selección.

ob es un objeto cualquiera.

2EF6E ParseFail ( ob $ #FinSeleccion #TamañoSelección 🡺 )

Usa DispBadToken para editar una cadena y muestra el mensaje de alerta “Sintaxis incorrntos”.

6.1.7 Ancho de Descompilación

**Direcc. Nombre Descripción**

2F191 !DcompWidth ( # 🡺 )

Fija un nuevo valor para DcompWidth.

DcompWidth es el ancho (en caracteres) de las cadenas descompiladas.

Este ancho es usado para cortar la cadena resultante (para mostrar en la pila a un objeto), o para dividir la cadena en varias líneas (mayormente para edición de objetos).

Observa que hay varios comandos de descompilación que fijan a DcompWidth como el valor estándar para mostrar o editar, antes de hacer la descompilación.

2F190 DcompWidth@ ( 🡺 # )

Llama al valor actual de DcompWidth.

26459 setStdWid ( 🡺 )

Fija DcompWidth al valor estándar para mostrar la pila.

Si el flag 72 está activado, fija DcompWidth a 30.

Si el flag 72 está desactivado, fija DcompWidth a 19.

2645E setStdEditWid ( 🡺 )

Fija DcompWidth al valor estándar para la edición.

Si el flag 73 está activado, fija DcompWidth a 32.

Si el flag 73 está desactivado, fija DcompWidth a 21.

6.1.8 Descompilación con ^FSTR1

**Direcc. Nombre Descripción**

001004 ˆFSTR1 ( ob 🡺 $ )

Descompila un objeto mostrandolo siempre en una sóla línea.

Si la cadena resultante es mayor a DcompWidth, será cortada un objeto más allá del ancho DcompWidth.

Si el flag 85 está activado, mostrará la representación de los objetos en System RPL.

25F13 stkdecomp$w ( ob 🡺 $ )

Hace lo mismo que FLASHPTR FSTR1.

25E6D 1stkdecomp$w ( ob 🡺 $ )

Hace: setStdWid FLASHPTR FSTR1.

2A842 Decomp1Line ( ob 🡺 $ )

Idéntico al comando 1stkdecomp$w.

2A904 RPNDecomp1Line ( ob 🡺 $ )

Similar a Decomp1Line, pero pone la calculadora en modo RPN (desactivando flag 95) durante su ejecución.

25E6F >Review$ ( # $ 🡺 $ )

( # id/lam 🡺 $ )

( # ob 🡺 $ )

Crea una cadena a partir de la variable y su contenido (descompilado con Decomp1Line). El bint debe estar entre 1 y 6 inclusive y es el número de tecla de menú.

Este comando es llamado cuando se presiona la tecla ShiftDerecho + Abajo.

2A8E4 DecompStd1Line32 ( ob 🡺 $ )

Hace: BINT32 !DcompWidth FLASHPTR FSTR1.

**Direcc. Nombre Descripción**

2A9C4 RPNDecompStd1Line32 ( ob 🡺 $ )

Similar a DecompStd1Line32, pero pone la calculadora en modo RPN (desactivando flag 95) durante su ejecución.

6.1.9 Descompilación con ^FSTR9

**Direcc. Nombre Descripción**

009004 ˆFSTR9 ( ob 🡺 $ )

Descompila un objeto mostrandolo siempre en una sóla línea.

Este comando es parecido a ^FSTR1.

La diferencia es que ^FSTR9 corta a la cadena como si el valor de DcompWidth fuera 32. Además, no muestra la representación de los objetos en System RPL.

2A8C9 DecompStd1Line ( ob 🡺 $ )

Hace: setStdWid FLASHPTR FSTR9.

2A9A4 RPNDecompStd1Line ( ob 🡺 $ )

Similar a DecompStd1Line, pero pone la calculadora en modo RPN (desactivando flag 95) durante su ejecución.

6.1.10 Descompilación con ^FSTR8 y ^FSTR6

**Direcc. Nombre Descripción**

008004 ˆFSTR8 ( ob 🡺 $ )

Descompila un objeto pudiendo mostrarlo en varias líneas.

Divide la cadenas en varías líneas según el valor DcompWidth.

Reales y complejos son mostrados en el formato actual.

Cadenas hexadecimales son mostradas con un número de cifras de acuerdo al ancho de palabra actual.

No muestra la representación de los objetos en System RPL.

006004 ˆFSTR6 ( ob #n 🡺 $ )

Descompila un objeto mostrandolo en varias líneas.

Este comando es parecido a ^FSTR8.

La diferencia es que retorna una cadena que sólo tiene las primeras #n+2 líneas.

2A893 Decomp#Disp ( ob # 🡺 $ )

Hace: setStdWid FLASHPTR FSTR6.

2A964 RPNDecomp#Disp ( ob # 🡺 $ )

Similar a Decomp#Disp, pero pone la calculadora en modo RPN (desactivando flag 95) durante su ejecución.

6.1.11 Descompilación con ^FSTR2 y ^FSTR12

**Direcc. Nombre Descripción**

002004 ˆFSTR2 ( ob 🡺 $ )

Descompila un objeto. Muestra representación en System RPL.

Agrega al final una línea adicional con el carácter @.

Si el flag 92 está desactivado, retorna 2 líneas adicional al inicio: "!NO CODE" y "!RPL"

Los comandos System RPL aparecerán con su nombre (si tienen uno y está instalada la biblioteca Extable) o de la forma PTR XXXXX, FLASHPTR XXX XXX o ROMPTR XXX XXX.

Equivale al comando **→S2** de User RPL.

00C004 ˆFSTR12 ( ob 🡺 $ )

Si el flag 85 está activado, muestra la representación de un objeto en System RPL.

Si el flag 85 está desactivado, descompila el objeto siempre en una sóla línea, mostrando reales y complejos en el formato estándar y las cadenas hexadecimales con todas sus cifras.

OBS: Si ob contiene objetos System RPL, el flag 85 deberá activarse antes.

6.1.12 Descompilación con ^FSTR4

**Direcc. Nombre Descripción**

004004 ˆFSTR4 ( ob 🡺 $ )

Descompila un objeto pudiendo mostrarlo en varias líneas.

Divide la cadenas en varías líneas según el valor DcompWidth.

Sólo descompila los componentes User RPL. Algunos comandos de System RPL como TakeOver son pasados por alto, otros son escritos como "External".

Reales y complejos son mostrados en formato estándar.

Cadenas hexadecimales son mostradas con todos sus cifras.

25F11 editdecomp$w ( ob 🡺 $ )

Hace lo mismo que FLASHPTR FSTR4.

25ECE EDITDECOMP$ ( ob 🡺 $ )

Hace: setStdEditWid FLASHPTR FSTR4.

Llamado por el comando **EDIT** de User RPL para editar cualquier objeto.

2A85D DecompEdit ( ob 🡺 $ )

Idéntico al comando EDITDECOMP$.

2A924 RPNDecompEdit ( ob 🡺 $ )

Similar a EDITDECOMP$, pero pone la calculadora en modo RPN (desactivando flag 95) durante su ejecución.

2AA43 AlgDecomp ( ob 🡺 $ )

En modo RPN, sólo llama a DecompEdit.

En modo algebraico, hace algunas verificaciones antes de llamar a DecompEdit.

00A004 ˆFSTR10 ( ob 🡺 $ )

Descompila un objeto pudiendo mostrarlo en varias líneas.

Este comando es parecido a ^FSTR4.

La diferencia es que ^FSTR10 corta a la cadena como si el valor de DcompWidth fuera 21.

**Direcc. Nombre Descripción**

00B004 ˆFSTR11 ( ob 🡺 $ )

Descompila un objeto pudiendo mostrarlo en varias líneas.

Este comando es parecido a ^FSTR4.

La diferencia es que ^FSTR11 corta a la cadena como si el valor de DcompWidth fuera 32.

6.1.13 Descompilación con ^FSTR5

**Direcc. Nombre Descripción**

005004 ˆFSTR5 ( ob 🡺 $ )

Descompila un objeto mostrandolo siempre en una sóla línea.

La totalidad del objeto es retornada como cadena sin importar el valor de DcompWidth.

Reales y complejos son mostrados en formato estándar.

Cadenas hexadecimales son mostradas con todos sus cifras.

2A8AE DecompEcho ( ob 🡺 $ )

Hace: setStdEditWid FLASHPTR FSTR5.

2A984 RPNDecompEcho ( ob 🡺 $ )

Similar a DecompEcho, pero pone la calculadora en modo RPN (desactivando flag 95) durante su ejecución.

6.1.14 Descompilación con ^FSTR13

**Direcc. Nombre Descripción**

00D004 ˆFSTR13 ( ob 🡺 $ )

Descompila un objeto y rompe la cadena en varias líneas usando DcompWidth como ancho de línea.

Reales y complejos son mostrados en formato actual.

Cadenas hexadecimales son mostradas con todos sus cifras.

25EAA DECOMP$ ( ob 🡺 $ )

Hace: setStdWid FLASHPTR FSTR13.

39CB3 (Ob,$>$') ( ob $ 🡺 "ob$" )

Aplica DECOMP$ a ob y lo concatena con la cadena.

39C9F ($,Ob>$') ( $ ob 🡺 "$ob" )

Aplica DECOMP$ a ob y lo concatena con la cadena.

6.1.15 Descompilación con ^FSTR7

**Direcc. Nombre Descripción**

007004 ˆFSTR7 ( ob 🡺 $ )

Descompila un objeto en una sóla línea.

Sin embargo, algunos objetos como matrices y programas son mostrados en más de una línea pero no según el valor de DcompWidth.

Reales y complejos son mostrados en formato actual.

Cadenas hexadecimales son mostradas con todos sus cifras.

25EB1 DO>STR ( $ 🡺 $ )

( ob 🡺 $ )

Para objetos que no sean cadenas hace:

setStdWid FLASHPTR FSTR7.

Equivale al comando **→STR** de User RPL.

1A7006 ˆDO>STRID ( id/ob 🡺 $ )

Como DO>STR pero sin comillas para un id.

6.1.16 Descompilación con ^FSTR3

**Direcc. Nombre Descripción**

003004 ˆFSTR3 ( ob #n 🡺 $ )

Parecido a ^FSTR6, pero la cadena retornada es la representación interna de las diferentes líneas a ser mostradas.

DcompWidth deberá ser fijado antes.

2A878 Decomp#Line ( ob # 🡺 $ )

Hace: setStdWid FLASHPTR FSTR3.

2A944 RPNDecomp#Line ( ob # 🡺 $ )

Similar a Decomp#Line, pero pone la calculadora en modo RPN (desactivando flag 95) durante su ejecución.

6.1.17 Más Descompilación

**Direcc. Nombre Descripción**

2F1BF Decomp%Short ( % #width 🡺 $ )

Descompila un número real en una cadena de un ancho dado.

Borrará cifras menos significativas, pero también excederá ancho cuando sea necesario.

Por ejemplo "-1.E-33" no puede ser escrito con menos de 7 decimales, de tal manera que si #width es menor que siete, siete caracteres serán usados de todos modos.

%0 es siempres descompilado como "0"

Cuidado: Este comando no siempre muestra correctamente algunos números menores que 1. Por ejemplo, el número 1.41421356237E-2 con un #width de 6 es mostrado erroneamente como "1.4142".

35B82 palrompdcmp ( ROMPTR 🡺 $ T )

Si es un comando de User RPL, convierte el rompointer en una cadena con el mismo nombre. De lo contrario, lo convierte en una cadena de la forma “XLIB #lib #cmd” (en base decimal).

6.1.18 Tests con cadenas

**Direcc. Nombre Descripción**

0556F NULL$? ( ob 🡺 flag )

36252 DUPNULL$? ( ob 🡺 ob flag )

2F321 CkChr00 ( $ 🡺 $ flag )

Retorna FALSE si la cadena contiene algún carácter nulo.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | DESCOMP SYS RPL | FORMATO NUMÉRICO | ANCHO DE PALABRA | COMILLAS ID ÚNICO | COMILLAS UNIDAD | ANCHO | matrices y programas  varias lineas |
| FSTR1 | FLAG 85 | ACTUAL | ACTUAL | SI | NO | Un objeto más allá de DcompWidth (1L) | NO |
| FSTR2 | SIEMPRE |  |  |  |  |  |  |
| FSTR4 |  | ESTANDAR | TODO | SI | SI | DcompWidth | SI |
| FSTR5 |  | ESTANDAR | TODO | NO | SI | Ilimitado (1L) | NO |
| FSTR6 |  | ACTUAL | ACTUAL | SI | NO | DcompWidth | SI |
| FSTR7 |  | ACTUAL | TODO | SI | SI | Ilimitado (1L)  Prog y matr (nL) | SI |
| FSTR8 |  | ACTUAL | ACTUAL | SI | NO | DcompWidth | SI |
| FSTR9 |  | ACTUAL | ACTUAL | SI | NO | Un objeto más allá de 32 (1L) | NO |
| FSTR10 |  | ESTANDAR | TODO | SI | SI | 21 | SI |
| FSTR11 |  | ESTANDAR | TODO | SI | SI | 32 | SI |
| FSTR12 | FLAG 85 | ESTANDAR | TODO | SI | SI | Ilimitado (1L) | NO |
| FSTR13 |  | ACTUAL | TODO | NO | SI | DcompWidth | SI |

6.2 Ejemplos

Ejemplo 1 Cadenas

Forzando a una cadena a no sobrepasar un tamaño determinado

El siguiente NULLNAME fuerza a una cadena a que tenga como máximo un número de caracteres indicado en la pila como un bint.

\* Fuerza a la cadena a tener como máximo #max caracteres

\* Si la cadena tiene más de #max caracteres, trunca la cadena a

\* #max-1 caracteres y le agrega el caracter ... al final.

\* #max debe ser 1 o mayor.

NULLNAME COERCE$ ( $ #max -> $' )

:: ( $ #max )

OVERLEN$ ( $ #max #n )

OVER ( $ #max #n #max )

#> ( $ #max flag )

NOTcaseDROP

( $ #max )

1\_#1-SUB$ ( $' )

CHR\_... ( $' chr )

>T$ ( $'' )

;

Ejemplo 2 Cadenas

Poniendo estilo a una cadena

Los siguientes NULLNAME ponen una cadena en negrita, cursiva, subrayado o inversa.

Las cadenas retornadas estarán en la forma: "\13\0X\13………\13\0X\13".

\* Pone una cadena de texto en negrita

NULLNAME PONE\_NEGRITA ( $ -> $' )

:: ( $ )

"\13\01\13" ( $ "\13\01\13" )

NULL$ ( $ "\13\01\13" "" )

FLASHPTR 00F 01A ( $' #veces ) ( comando SREPL de User RPL )

DROP ( $' )

"\13\01\13" ( $ "\13\01\13" )

SWAPOVER ( "\13\01\13" $ "\13\01\13" )

&$ &$ ( $' )

;

\* Pone una cadena de texto en cursiva

NULLNAME PONE\_CURSIVA ( $ -> $' )

:: ( $ )

"\13\02\13" ( $ "\13\01\13" )

NULL$ ( $ "\13\01\13" "" )

FLASHPTR 00F 01A ( $' #veces ) ( comando SREPL de User RPL )

DROP ( $' )

"\13\02\13" ( $ "\13\01\13" )

SWAPOVER ( "\13\01\13" $ "\13\01\13" )

&$ &$ ( $' )

;

\* Pone una cadena de texto en subrayado

NULLNAME PONE\_SUBRAYADA ( $ -> $' )

:: ( $ )

"\13\03\13" ( $ "\13\01\13" )

NULL$ ( $ "\13\01\13" "" )

FLASHPTR 00F 01A ( $' #veces ) ( comando SREPL de User RPL )

DROP ( $' )

"\13\03\13" ( $ "\13\01\13" )

SWAPOVER ( "\13\01\13" $ "\13\01\13" )

&$ &$ ( $' )

;

\* Pone una cadena de texto en inversa

NULLNAME PONE\_INVERSA ( $ -> $' )

:: ( $ )

"\13\04\13" ( $ "\13\01\13" )

NULL$ ( $ "\13\01\13" "" )

FLASHPTR 00F 01A ( $' #veces ) ( comando SREPL de User RPL )

DROP ( $' )

"\13\04\13" ( $ "\13\01\13" )

SWAPOVER ( "\13\01\13" $ "\13\01\13" )

&$ &$ ( $' )

;

Ejemplo 3 Cadenas

Quitando estilo a una cadena

Los siguientes NULLNAME remueven totalmente en una cadena los estilos negrita, cursiva, subrayado o inversa.

\* Hace que ningún carácter en la cadena de texto esté en negrita

NULLNAME REMUEVE\_NEGRITA ( $ -> $' )

:: ( $ )

"\13\01\13" ( $ "\13\01\13" )

NULL$ ( $ "\13\01\13" "" )

FLASHPTR 00F 01A ( $' #veces ) ( comando SREPL de User RPL )

DROP ( $' )

;

\* Hace que ningún carácter en la cadena de texto esté en cursiva

NULLNAME REMUEVE\_CURSIVA ( $ -> $' )

:: ( $ )

"\13\02\13" ( $ "\13\01\13" )

NULL$ ( $ "\13\01\13" "" )

FLASHPTR 00F 01A ( $' #veces ) ( comando SREPL de User RPL )

DROP ( $' )

;

\* Hace que ningún carácter en la cadena de texto esté en subrayado

NULLNAME REMUEVE\_SUBRAYADA ( $ -> $' )

:: ( $ )

"\13\03\13" ( $ "\13\01\13" )

NULL$ ( $ "\13\01\13" "" )

FLASHPTR 00F 01A ( $' #veces ) ( comando SREPL de User RPL )

DROP ( $' )

;

\* Hace que ningún carácter en la cadena de texto esté en inversa

NULLNAME REMUEVE\_INVERSA ( $ -> $' )

:: ( $ )

"\13\04\13" ( $ "\13\01\13" )

NULL$ ( $ "\13\01\13" "" )

FLASHPTR 00F 01A ( $' #veces ) ( comando SREPL de User RPL )

DROP ( $' )

;

Ejemplo 4 Cadenas

Alternando una cadena respecto al estilo negrita

El siguiente xNAME alterna una cadena respecto al estilo negrita.

Pone en estilo negrita a una cadena o remueve el estilo negrita de acuerdo a la cadena que se encuentra en la pila.

\* Si una cadena de texto está en negrita (en la forma "\13\01\13....\13\01\13"),

\* entonces ejecuta REMUEVE\_NEGRITA.

\* De lo contrario ejecuta PONE\_NEGRITA

ASSEMBLE

CON(1) 8 ( Le dice al parseador que el comando es 'No algebraico' )

RPL

xNAME NEGRITA ( $ -> $' )

:: ( $ )

CK1&Dispatch

str

:: ( $ )

DUP ( $ $ )

BINT4 ( $ $ #4 )

1\_#1-SUB$ ( $ $' )

"\13\01\13" ( $ $' "\13\01\13" )

EQUALNOTcase

PONE\_NEGRITA

( $ )

DUP ( $ $ )

BINT4 ( $ $ #4 )

OVERLEN$ ( $ $ #4 #n )

SUB$ ( $ $' )

"\13\01\13" ( $ $' "\13\01\13" )

BINT1 ( $ $' "\13\01\13" #1 )

POS$ ( $ #pos/#0 )

DUP#0= ( $ #pos/#0 flag )

casedrop

PONE\_NEGRITA

( $ #pos )

OVERLEN$ ( $ #pos #n )

#5- ( $ #pos #n-2 )

#=case

REMUEVE\_NEGRITA

( $ )

PONE\_NEGRITA ( $' )

;

;

Capítulo 7  
Cadenas Hexadecimales (HXS)

Cadenas hexadecimales son los numeros que son llamados Enteros Binarios en el manual oficial de Hewlett Packard, los cuales pueden ser representados en varias bases. En System RPL estos son llamados Cadenas Hexadecimales. Son creados usando la estructura HXS <long> <cuerpo\_hex>. Donde long es la longitud de la cadena (número de nibbles o digitos hexadecimales), en forma hexadecimal, y cuerpo\_hex es el contenido de este. La parte complicada de esto es que, debido a la arquitectura interna de la HP, debes ingresar el contenido en orden inverso. Por ejemplo, para conseguir la cadena #12AD7h, debes ingresar HXS 5 7DA21. Para conseguir #12345678h usa HXS 8 87654321.

En System RPL, las cadenas hexadecimales pueden ser de cualquier longitud, no como en User RPL, donde estas estaban limitadas a 16 nibbles o 64 bits.

Para convertir de cadena hexadecimal a entero binario y viceversa, usa los comandos HXS># y #>HXS respectivamente. Para convertir un HXS a número real y viceversa, usa #>% (o HXS>%) y %># respectivamente.

Ver abajo para más comandos relacionados a cadenas hexadecimales.

7.1 Referencia

7.1.1 Conversión

**Direcc. Nombre Descripción**

059CC #>HXS ( # 🡺 hxs )

Convierte bint a hxs. La longitud del hxs será cinco.

2EFCB %># ( % 🡺 hxs )

Convierte número real a hxs. Debería llamarse %>HXS.

7.1.2 General Functions

**Direcc. Nombre Descripción**

2EFBE WORDSIZE ( 🡺 # )

Retorna el actual tamaño de palabra (wordsize) como un bint.

2EFAA dostws ( # 🡺 )

Fija el actual tamaño de palabra (wordsize). Este debe estar entre BINT0 y BINT64.

055D5 NULLHXS HXS 0

Pone un hxs nulo en la pila.

0518A &HXS ( hxs hxs' 🡺 hxs'' )

Agrega hxs' a hxs. Las longitudes se suman y los cuerpos se escriben uno a continuación del otro.

Por ejemplo, el programa de abajo retorna la cadena hexadecimal HXS 5 ABCDE

:: HXS 3 ABC HXS 2 DE &HXS ;

34C82 EXPAND ( hxs #nibs 🡺 hxs' )

La longitud del hxs aumenta en #nibs y al cuerpo del hxs se agregan #nibs ceros. Por ejemplo, el programa de abajo retorna la cadena hexadecimal HXS 7 ABCDE00

:: HXS 5 ABCDE BINT2 EXPAND ;

05616 LENHXS ( hxs 🡺 #nibs )

Retorna longitud en nibbles.

05815 SUBHXS ( hxs #m #n 🡺 hxs' )

Retorna sub hxs. Por ejemplo, el programa de abajo retorna la cadena hexadecimal HXS 3 BCD

:: HXS 6 ABCDEF BINT2 BINT4 SUBHXS ;

2EFB9 bit+ ( hxs hxs' 🡺 hxs'' )

Suma dos hxs.

2EFC8 bit%#+ ( % hxs 🡺 hxs' )

Suma real a hxs, retorna hxs.

2EFC9 bit#%+ ( hxs % 🡺 hxs' )

Suma real a hxs, retorna hxs.

2EFBA bit- ( hxs hxs' 🡺 hxs'' )

Resta hxs2 de hxs1.

2EFC6 bit%#- ( % hxs 🡺 hxs' )

Resta hxs de real, retorna hxs.

2EFC7 bit#%- ( hxs % 🡺 hxs' )

Resta real de hxs, retorna hxs.

2EFBC bit\* ( hxs hxs' 🡺 hxs'' )

Multiplica dos hxs.

2EFC4 bit%#\* ( % hxs 🡺 hxs' )

Multiplica real por hxs, retorna hxs.

**Direcc. Nombre Descripción**

2EFC5 bit#%\* ( hxs % 🡺 hxs' )

Multiplica hxs por real, retorna hxs.

2EFBD bit/ ( hxs hxs' 🡺 hxs'' )

Divide hxs1 entre hxs2.

2EFC2 bit%#/ ( % hxs 🡺 hxs' )

Divide real entre hxs, retorna hxs.

2EFC3 bit#%/ ( hxs % 🡺 hxs' )

Divide hxs entre real, retorna hxs.

2EFAC bitAND ( hxs hxs' 🡺 hxs'' )

AND bit a bit.

2EFAD bitOR ( hxs hxs' 🡺 hxs'' )

OR bit a bit.

2EFAE bitXOR ( hxs hxs' 🡺 hxs'' )

XOR bit a bit.

2EFAF bitNOT ( hxs 🡺 hxs' )

NOT bit a bit.

2EFB8 bitASR ( hxs 🡺 hxs' )

Cambio de puesto aritmético de un bit a la derecha. El bit más significativo no cambia.

2EFB6 bitRL ( hxs 🡺 hxs' )

Rota un bit a la izquierda.

2EFB7 bitRLB ( hxs 🡺 hxs' )

Rota un byte a la izquierda.

2EFB4 bitRR ( hxs 🡺 hxs' )

Rota un bit a la derecha.

2EFB5 bitRRB ( hxs 🡺 hxs' )

Rota un byte a la derecha.

2EFB0 bitSL ( hxs 🡺 hxs' )

Mueve un bit a la izquierda.

2EFB1 bitSLB ( hxs 🡺 hxs' )

Mueve un byte a la izquierda.

2EFB2 bitSR ( hxs 🡺 hxs' )

Mueve un bit a la derecha.

2EFB3 bitSRB ( hxs 🡺 hxs' )

Mueve un byte a la derecha.

7.1.3 Tests

Estos tests son para comparar dos cadenas hexadecimales.

Estos comandos primero truncan o amplian los argumentos de acuerdo a la actual longitud de palabra y luego hacen la comparación.

**Direcc. Nombre Descripción**

2EFCC HXS==HXS ( hxs hxs' 🡺 %flag )

== test

2F0EE HXS#HXS ( hxs hxs' 🡺 %flag )

≠ test

2EFCF HXS<HXS ( hxs hxs' 🡺 %flag )

< test

2EFCD HXS>HXS ( hxs hxs' 🡺 %flag )

> test

2EFCE HXS>=HXS ( hxs hxs' 🡺 %flag )

≥ test

2F0EF HXS<=HXS ( hxs hxs' 🡺 %flag )

≤ test

Capítulo 8  
Identificadores

Los identificadores son usados para representar los nombres de objetos guardados en memoria (es decir, variables). En User RPL, estos aparecen en la pila entre comillas simples, esto es, ''. En System RPL, son creados con ID <name>. Cuando usas esta estructura, no siempre consigues tener el identificador en la pila. Este es siempre evaluado.

Por ejemplo, si la variable anumber contiene 123.45 y tu pones en alguna parte de tu programa ID anumber, el identificador es evaluado, llamando al contenido de la variable. De esta manera, en la pila aparecerá 123.45.

Para poner un id en la pila, usa ' ID <name>. Como verás en el capítulo 20, el comando ' pone el objeto que está a continuación en la pila. Esto es llamado citación.

Sin embargo, ID <name> (sin el ') pondrá también el id en la pila si aún no existe la variable llamada <name>. Este comportamiento es similar al observado cuando ingresas una variable sin las comillas simples en la línea de comandos de la calculadora.

Puedes convertir una cadena en un id usando $>ID. La transformación opuesta se consigue con el comando ID>$.

También hay otro tipo de identificadores: los identificadores temporales, también llamados lams. Estos son usados cuando creas variables locales, y tu aprenderás sobre estos más tarde en el capítulo 19. Son creados con LAM <name>, y trabajan de manera bastante parecida a los ids normales (variables globales).

Puesto que los ids están estrechamente relacionados al acceso a la memoria, los comandos relacionados con estos son tratados en el capítulo 25.

Capítulo 9  
Objetos Etiquetados (Tags)

Para insertar un objeto etiquetado en tu programa, puedes usar la siguiente estructura TAG <tag> <object>. Donde <tag> es una cadena sin comillas, y el objeto puede ser cualquiera. Por ejemplo, para crear 0: 1790, usarías TAG 0 % 1790.

Un objeto puede tener muchas etiquetas, pero no se usa esto con frecuencia.

El comando >TAG crea un objeto etiquetado, dado el objeto (en el nivel dos) y una cadena que representa la etiqueta (en el nivel uno). %>TAG funciona de manera similar, pero etiqueta un objeto con un número real. ID>TAG etiqueta un objeto con un identificador.

Para remover todas las etiquetas de un objeto usa STRIPTAGS.

Algunos comandos adicionales relacionados a los objetos etiquetados son listados abajo.

Nota que el programador rara vez necesita preocuparse de los objetos etiquetados, pues el mecanismo de despachar tipos (que es descrito en la sección 30.2) puede remover las etiquetas de los argumentos de tu programa de manera automática.

9.1 Referencia

**Direcc. Nombre Descripción**

05E81 >TAG ( ob $tag 🡺 tagged )

Etiqueta un objeto.

2F266 USER$>TAG ( ob $tag 🡺 tagged )

Etiqueta un objeto con un máximo de 255 caracteres en $tag.

2F223 %>TAG ( ob % 🡺 tagged )

Convierte real a cadena usando el actual formato de número y etiqueta objeto.

05F2E ID>TAG ( ob id/lam 🡺 tagged )

Etiqueta objeto con nombre global (id) o nombre local (lam).

05E9F {}>TAG\_ ( { id ob } 🡺 tagged )

( { lam ob } 🡺 tagged )

Etiqueta objeto con nombre global (id) o nombre local (lam).

37B04 TAGOBS ( ob $tag 🡺 tagged )

( ob.. { $.. } 🡺 tagged... )

Etiqueta uno o más objetos.

37ABE STRIPTAGS ( tagged 🡺 ob )

( ob 🡺 ob )

Quita todas las etiquetas del objeto.

37AEB STRIPTAGSl2 ( tagged ob' 🡺 ob ob' )

( ob ob' 🡺 ob ob' )

Quita todas las etiquetas del objeto del nivel 2.

05EC9 TAG>\_ ( tagged 🡺 ob $tag )

Quita solamente la etiqueta más externa.

Capítulo 10  
Arreglos (Arrays)

Actualmente hay dos grupos de objetos que representan formaciones en la calculadora HP.

El primer grupo (que se describe en este capítulo) ha existido desde la HP48: son los arreglos normales (aquellos que en User RPL pueden ser sólo de números reales o números complejos), y los arreglos vinculados (linked arrays), los cuales no son accesibles usando User RPL.

La HP49 introduce un nuevo tipo de objeto para representar formaciones: las matrices simbólicas. Debido a que estas son parte del HP49 CAS, son descritas en el capítulo 46.

En User RPL, los arreglos pueden ser solamente de reales o números complejos. En System RPL, puedes tener arreglos de cualquier cosa, hasta arreglos de arreglos. Nota que un arreglo no es un objeto compuesto (ver capítulo 11), aunque parezca un compuesto. También nota que un arreglo puede contener objetos solamente de un tipo.

En el editor de Debug4x, los arreglos son ingresados de esta manera:

|  |  |
| --- | --- |
| **Entrada** | **Resultado** |
| ARRY 2 3 [ % 1. % 2. % 3. % 4. % 5. % 6. ] | [ [ 1. 2. 3. ] [ 4. 5. 6. ] ] |
| ARRY 3 [ % 11. % 12. % 13. ] | [ 11. 12. 13. ] |
| ARRY [ % 11. % 12. % 13. ] | [ 11. 12. 13. ] |
| ARRY 2 3 [ “a” “b” “c” “d” “e” “f” ] | [ [ “a” “b” “c” ] [ “d” “e” “f” ] ] |

Donde el primer número es el número de filas y el siguiente es el número de columnas. Si el arreglo es un vector sólo será necesario el número de elementos e incluso este número se puede omitir.

En User RPL, sólo puedes usar arreglos de una o de dos dimensiones. En cambio, en System RPL, también puedes usar arreglos de más de dos dimensiones. Sólo debes escribir a continuación de la palabra ARRY las longitudes de cada dimensión. En el siguiente ejemplo, el resultado es un arreglo de tres dimensiones.

|  |  |
| --- | --- |
| **Entrada** | ARRY 2 2 4 [ “a” “b” “c” “d” “e” “f” “g” “h” “i” “j” “k” “l” “m” “n” “o” “p” ] |
| **Resultado** | [ [ [ “a” “b” “c” “d” ] [ “e” “f” “g” “h” ] ] [ [ “i” “j” “k” “l” ] [ “m” “n” “o” “p” ] ] ] |

La estructura de un arreglo de ‘n’ dimensiones es la siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| Prólogo | DOARRY # 29E8 | 5 | “8E920” |
| Cuerpo | Tamaño del cuerpo | 5 |  |
| Prólogo de los objetos | 5 |  |
| Nº de dimensiones | 5 |  |
| Longitud de la dimensión 1 | 5 |  |
| Longitud de la dimensión 2 | 5 |  |
| ………. |  |  |
| Longitud de la dimensión n | 5 |  |
| Objetos (sin su prólogo) |  |  |

También hay otro tipo de arreglos: los arreglos vinculados (linked arrays). Estos son como los arreglos normales (también pueden contener objetos de un sólo tipo). La diferencia es que tienen una tabla con punteros a todos los objetos en el arreglo. Con esto, tendremos un acceso mucho más rapido a los elementos del arreglo, pues cuando necesites acceder a un objeto en el arreglo vinculado, sólo será necesario leer el puntero a la posición de ese objeto en la tabla e ir ahí directamente. En cambio, en los arreglos normales, el comando que consigue un elemento hace una búsqueda para encontrar al elemento.

Como un arreglo vinculado tiene una tabla con punteros, entonces tendrá un tamaño mayor a un arreglo normal.

Para ingresar arreglos vinculados en el editor de Debug 4x, sólo debes cambiar la palabra ARRY por LNKARRY.

LNKARRY 2 3 [ % 1. % 2. % 3. % 4. % 5. % 6. ]

LNKARRY 3 [ % 11. % 12. % 13. ]

LNKARRY [ % 11. % 12. % 13. ]

LNKARRY 2 3 [ “a” “b” “c” “d” “e” “f” ]

LNKARRY 2 2 2 [ “ab1” “ab2” “ab3” “ab4” “ab5” “ab6” “ab7” “ab8” ]

La estructura de un arreglo vinculado de ‘n’ dimensiones y ‘N’ objetos es la siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| Prólogo | DOLNKARRY # 2A0A5 | 5 | “A0A20” |
| Cuerpo | Tamaño del cuerpo | 5 |  |
| Prólogo de los objetos | 5 |  |
| Nº de dimensiones | 5 |  |
| Longitud de la dimensión 1 | 5 |  |
| Longitud de la dimensión 2 | 5 |  |
| ………. |  |  |
| Longitud de la dimensión n | 5 |  |
| Puntero al objeto 1 |  |  |
| Puntero al objeto 2 |  |  |
| ………. |  |  |
| Puntero al objeto N |  |  |
| Objetos (sin su prólogo) |  |  |

En un arreglo vinculado pueden no estar presentes todos los objetos. En este caso el puntero a los objetos ausentes debe ser cero. Por ejemplo en el siguiente arreglo vinculado:

LNKARRY 5 [ % 11. % 12. % 13. ]

Faltan los elementos cuarto y quinto. Por lo tanto, los punteros correspondientes al cuarto objeto y quinto objeto deben ser ceros.

También puedes crear un arreglo de números reales o complejos (normales, no extendidos) poniéndolos en la pila e ingresando una lista con las dimensiones del arreglo (números reales, no bints) en el nivel 1. Luego ejecuta ˆXEQ>ARRY. Este comando hace verificaciones para asegurarse que hay suficientes argumentos y estos son de los tipos permitidos.

El comando ˆARSIZE retorna el número de elementos en un arreglo. Puedes conseguir las dimensiones de un arreglo con ˆDIMLIMITS, el cual retorna una lista de bints que representan las dimensiones del arreglo.

Para conseguir un elemento de un arreglo, pon el número del elemento en el nivel 2, el arreglo en el nivel 1 y ejecuta GETATELN. Conseguirás el elemento y TRUE.

Todas las entradas de aquí, tratán con los arreglos normales (aún cuando algunos de estos funcionan también para arreglos vinculados y Matrices Simbólicas). Para entradas específicas de Matrices Simbólicas ver el capítulo 46.

10.1 Referencia

10.1.1 Dimensiones de un arreglo

**Direcc. Nombre Descripción**

035A9 DIMLIMITS\_ ( ARRY 🡺 {#L1,#L2,…,#Ln} )

( LNKARRY 🡺 {#L1,#L2,…,#Ln} )

Retorna una lista con tantos bints como dimensiones tiene el arreglo.

Cada bint representa la longitud de una dimensión del arreglo.

Para arreglos de una dimensión, retorna una lista con un bint que es el número de elementos.

Para arreglos de dos dimensiones, retorna una lista con dos bints: el número de filas y de columnas.

Para arreglos de tres dimensiones retorna una lista con tres bints, etc.

16E006 ˆDIMLIMITS ( ARRY 🡺 {#L1,#L2,…#Ln} )

( LNKARRY 🡺 {#L1,#L2,…#Ln} )

( 2DMATRIX 🡺 {#m #n} )

( 1DMATRIX 🡺 {#elem} )

Similar al comando DIMLIMITS\_, pero también funciona para matrices simbólicas.

03562 ARSIZE\_ ( ARRY 🡺 # )

( LNKARRY 🡺 # )

Retorna el número de elementos como un bint.

35E006 ˆARSIZE ( ARRY 🡺 # )

( LNKARRY 🡺 # )

( MATRIX 🡺 # )

Similar al comando ARSIZE\_, pero también funciona para matrices simbólicas.

36183 OVERARSIZE ( ARRY ob 🡺 ARRY ob # )

( LNKARRY ob 🡺 LNKARRY ob # )

( MATRIX ob 🡺 MATRIX ob # )

Hace OVER luego ^ARSIZE.

3BA001 FLASHPTR 001 3BA ( 2DARRY 🡺 #filas #cols T )

( 1DARRY 🡺 #elem F )

( 2DLNKARRY 🡺 #filas #cols T )

( 1DLNKARRY 🡺 #elem F )

Retorna el tamaño de un arreglo o arreglo vinculado de una o de dos dimensiones.

Para arreglos o arreglos vinculados de tres o más dimensiones, retorna las longitudes de las dos primeras dimensiones y TRUE.

16D006 ˆMDIMS ( 2DARRY 🡺 #filas #cols T )

( 1DARRY 🡺 #elem F )

( 2DMATRIX 🡺 #filas #cols T )

( 1DMATRIX 🡺 #elem F )

Similar a FLASHPTR 001 3BA, pero no funciona para arreglos vinculados y si funciona para matrices simbólicas.

**Direcc. Nombre Descripción**

35FD8 MDIMSDROP ( 2DARRY 🡺 #filas #cols )

( 1DARRY 🡺 #elem )

( 2DMATRIX 🡺 #filas #cols )

( 1DMATRIX 🡺 #elem )

Hace ˆMDIMS luego DROP.

10.1.2 Obtener un elemento de un arreglo

**Direcc. Nombre Descripción**

0371D GETATELN ( #ubic ARRY 🡺 ob T )

( #ubic LNKARRY 🡺 ob T )

( #ubic LNKARRY 🡺 F )

Retorna un elemento en la ubicación especificada del arreglo o arreglo vinculado y TRUE.

El bint no debe ser mayor al tamaño del arreglo o arreglo vinculado.

Si el elemento no se encuentra en el arreglo vinculado (cuando el puntero del elemento es cero), retorna FALSE.

260F8 PULLREALEL ( RealARRY #ubic 🡺 RealARRY % )

( CmpARRY #ubic 🡺 CmpARRY % )(1≤#ubic≤2N)

Consigue un elemento de un arreglo real que tiene una o dos dimensiones.

Para un arreglo complejo de 1 o 2 dimensiones, consigue la parte real o imaginaria de uno de sus elementos.

Por ejemplo:

ARRY [ C% 11. 31. C% 12. 32. ] BINT3 PULLREALEL

Devuelve el número real: 12.

260F3 PULLCMPEL ( CmpARRY #ubic 🡺 CmpARRY C% )

( RealARRY #ubic 🡺 RealARRY C% ) (1≤#ubic≤floor(N/2))

Consigue un elemento de un arreglo complejo que tiene una o dos dimensiones.

Para un arreglo real de 1 o 2 dimensiones, retorna un número complejo cuyas componentes son los números de las posiciones #2\*ubic-1 y #2\*ubic. Por ejemplo:

ARRY [ 11. 12. 13. 14. 15. 16. ] BINT3 PULLCMPEL

Devuelve el número complejo: (15.,16.)

35B006 ˆPULLEL[S] ( RealARRY #ubic 🡺 RealARRY % )

( CmpARRY #ubic 🡺 CmpARRY C% )

( MATRIX #ubic 🡺 MATRIX ob )

Retorna el elemento de la posición especificada.

Funciona también para matrices simbólicas.

10.1.3 Reemplazar un elemento de un arreglo

**Direcc. Nombre Descripción**

26102 PUTEL ( RealARRY % #ubic 🡺 RealARRY' )

( CmpARRY C% #ubic 🡺 CmpARRY' )

Pone el elemento en la posición indicada. Si el número es real extendido o complejo extendido lo convierte a normal primero.

(comando tipo bang)

26107 PUTREALEL ( RealARRY % #ubic 🡺 RealARRY' )

Pone el elemento real en la posición indicada.

(comando tipo bang)

260FD PUTCMPEL ( CmpARRY C% #ubic 🡺 CmpARRY' )

Pone el elemento complejo en la posición indicada.

(comando tipo bang)

10.1.4 Número de orden de un elemento

**Direcc. Nombre Descripción**

03685 ARRYEL?\_ ( {#P1,#P2,…#Pn} ARRY/LNKARRY 🡺 #ubic T )

( {#P1,#P2,…#Pn} ARRY/LNKARRY 🡺 F )

Dada la posición de un elemento como una lista retorna un bint que representa la ubicación de ese elemento en el arreglo y TRUE.

Si la posición no existe en el arreglo o arreglo vinculado entonces solo retorna FALSE.

Por ejemplo, si en la pila se encuentra la lista

{ BINT2 BINT1 } y luego un arreglo de dimensiones 3x5, entonces el comando retorna #6 y TRUE, es decir, la ubicación del elemento de la segunda fila y la primera columna.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Col 1** | **Col 2** | **Col 3** | **Col 4** | **Col 5** |
| **Fila 1** | ubic 1 | ubic 2 | ubic 3 | ubic 4 | ubic 5 |
| **Fila 2** | ubic 6 | ubic 7 | ubic 8 | ubic 9 | ubic 10 |
| **Fila 3** | ubic 11 | ubic 12 | ubic 13 | ubic 14 | ubic 15 |

aka: FINDELN\_

35A006 ˆFINDELN ( {#P1,#P2……#Pn} ARRY/LNKARRY/MATRIX 🡺 #ubic T )

( {#P1,#P2……#Pn} ARRY/LNKARRY/MATRIX 🡺 F )

Similar al comando ARRYEL?\_, pero también funciona para matrices simbólicas.

0290E8 ROMPTR 0E8 029 ( #ubic 2DARRY/2DMATRIX 🡺 #f #c )

( #ubic 1DARRY/1DMATRIX 🡺 #1 #ubic )

Dada la ubicación de un elemento retorna dos bints que representan la fila y la columna correspondientes.

10.1.5 Formar y desintegrar un compuesto

**Direcc. Nombre Descripción**

17F006 ˆXEQ>ARRY ( %...%' {%el}/{%f,%c} 🡺 RealARRY )

( C%...C%' {%el}/{%f,%c} 🡺 CmpARRY )

( ob1...obn {%el}/{%f,%c} 🡺 MATRIX )

Crea una formación a partir de los elementos de la pila.

Si todos son reales, crea un arreglo real.

Si todos son complejos, crea un arreglo complejo.

De no cumplirse esas condiciones, crea una matriz simbólica.

Pero los objetos sólo puden ser reales, complejos y objetos de clase simbólica (enteros, symb, id o lam).

Usado por el comando **→ARRY** de User RPL.

180006 ˆXEQ>ARRAY1 ( %...%' {#el}/{#f,#c} #el/#fxc 🡺 RealArry )

( C%...C%' {#el}/{#f,#c} #el/#fxc 🡺 CmpArry )

( ob1...obn {#el}/{#f,#c} #el/#fxc 🡺 MATRIX )

Similar al comando ˆXEQ>ARRY, pero el número de elementos se indica con bints.

03442 MAKEARRY\_ ( {#L1,#L2,…#Ln} ob 🡺 ARRY )

Crea un arreglo con todos sus elementos iguales a ob.

Las dimensiones del arreglo serán las que indica la lista.

373006 ^MAKEARRY ( {#L1,#L2,…#Ln} % 🡺 RealARRY )

( {#L1,#L2,…#Ln} C% 🡺 CmpARRY )

( {#L1,#L2,…#Ln} %% 🡺 ExtRealARRY )

( {#L1,#L2,…#Ln} C%% 🡺 ExtCmpARRY )

( {#elem}/{#f,#c} Z 🡺 MATRIX )

( {#elem}/{#f,#c} symb 🡺 MATRIX )

( {#elem}/{#f,#c} id 🡺 MATRIX )

( {#elem}/{#f,#c} lam 🡺 MATRIX )

( {#elem}/{#f,#c} {} 🡺 MATRIX )

Parecido al comando MAKEARRY\_, pero solo funciona para crear formaciones con objetos de los tipos mostrados, pudiendo crear también matrices simbólicas.

Si en la pila hay un lam, crea una matriz cuyos elementos son iguales al contenido del lam; si el lam no existe, manda un mensaje de error.

17C006 ˆXEQARRY> ( RealARRY 🡺 %...%' {%el}/{%f,%c} )

( CmpARRY 🡺 %C...%C' {%el}/{%f,%c} )

( MATRIX 🡺 ob1...obn {%el}/{%f,%c} )

Desintegra un arreglo con elementos reales o complejos, y con una o dos dimensiones. La lista devuelta tiene uno o dos números reales que indican las dimensiones del arreglo. También funciona para matrices simbólicas.

Equivale al comando **ARRY→** de User RPL.

10.1.6 Conversión

**Direcc. Nombre Descripción**

003007 ˆArryToList ( [] 🡺 {} )

( [[]] 🡺 {{}} )

Convierte un arreglo a lista.

Funciona para arreglos con objetos de cualquier tipo.

Sólo funciona para arreglos de una o de dos dimensiones.

001007 ˆListToArry ( {}/{{}} 🡺 []/[[]] TRUE )

( {}/{{}} 🡺 FALSE )

( 1DMATRIX/2DMATRIX 🡺 []/[[]] TRUE )

( 1DMATRIX/2DMATRIX 🡺 FALSE )

( []/[[]] 🡺 FALSE )

( ob 🡺 FALSE )

Convierte la lista (o lista de listas) a arreglo real o complejo y pone TRUE en la pila.

Si no es posible hacer la conversión, retorna FALSE.

También funciona para matrices simbólicas.

La conversión sólo es posible cuando todos los elementos son reales o todos son complejos.

169006 ˆBESTMATRIXTYPE ( {}/{{}} 🡺 []/[[]] )

( {}/{{}} 🡺 {}/{{}} )

( 1DMATRIX/2DMATRIX 🡺 []/[[]] )

( 1DMATRIX/2DMATRIX 🡺 1DMATRIX/2DMATRIX )

( []/[[]] 🡺 []/[[]] )

( ob 🡺 ob )

Hace lo mismo que ˆListToArry pero no retorna un flag.

178006 ˆMATRIX2ARRAY ( 1DMATRIX 🡺 []/1DMATRIX )

( 2DMATRIX 🡺 [[]]/2DMATRIX )

( []/[[]] 🡺 []/[[]] )

Intenta convertir matriz a arreglo real o arreglo complejo incluso cambiando elementos (evaluando los objetos simbólicos y si es necesario, llamando al contenido de los nombres globales o locales o evaluandolos. Genera el error “Argumento incorrecto” si hay elementos no permitidos para formar matriz).

Si no es posible lo deja como matriz convirtiendo los elementos que se puedan a reales o complejos.

Este comando demora para formaciones grandes.

172006 ˆCKNUMARRY ( 1DMATRIX 🡺 [] )

( 2DMATRIX 🡺 [[]] )

( []/[[]] 🡺 []/[[]] )

Si en la pila hay un arreglo, no hace nada.

Si en la pila hay una matriz, llama a ˆMATRIX2ARRAY. De no ser posible la conversión a arreglo real o complejo, manda el mensaje de error “Argumento incorrecto”.

10.1.7 Crear Arreglos con el editor de matrices (MTRW)

Los comandos de esta sección tienen un comportamiento diferente según el estado del flag del sistema número 91.

Cuando el flag del sistema número 91 está activado, se acepta la entrada de objetos de cualquier tipo. Además, si uno sale del editor presionando ENTER, en la pila aparecerá una lista de listas y TRUE.

Cuando el flag del sistema número 91 está desactivado, se acepta sólo la entrada de objetos de algunos tipos como reales, complejos, ids, lams, symbs y enteros. Además, si uno sale del editor presionando ENTER, en la pila aparecerá un arreglo o una matriz simbólica y TRUE.

Si uno se retira del editor presionando CANCL, en la pila sólo aparecerá FALSE.

**Direcc. Nombre Descripción**

007007 ˆDoNewMatrixReal ( 🡺 RealArry T ) (flag 91 clear)

( 🡺 {{}} T ) (flag 91 set)

( 🡺 F )

Crea un arreglo real con el editor MTRW.

Devuelve FALSE cuando uno se retira del MTRW.

008007 ˆDoNewMatrixCplx ( 🡺 CmpArry T ) (flag 91 clear)

( 🡺 {{}} T ) (flag 91 set)

( 🡺 F )

Crea un arreglo complejo con el editor MTRW.

00B007 ˆDoNewMatrixRealOrCplx ( 🡺 RealArry T ) (flag 91 clear)

( 🡺 CmpArry T ) (flag 91 clear)

( 🡺 {{}} T ) (flag 91 set)

( 🡺 F )

Crea un arreglo real o complejo con el editor MTRW.

Si el primer objeto escrito es real, sólo se aceptarán reales.

Si el primer objeto escrito es complejo, sólo se aceptarán números complejos.

006007 ˆRunDoNewMatrix ( 🡺 RealArry T ) (flag 91 clear)

( 🡺 CmpArry T ) (flag 91 clear)

( 🡺 MATRIX T ) (flag 91 clear)

( 🡺 {{}} T ) (flag 91 set)

( 🡺 F )

Con el flag 91 desactivado hace lo siguiente:

Crea un arreglo real, arreglo complejo, o matriz simbólica con el editor MTRW.

Si todos los objetos son reales, crea un arreglo real.

Si todos son complejos, crea un arreglo complejo.

En otro caso, crea una matriz simbólica.

Sólo se permite el ingreso de reales, complejos, ids, lams, simbólicos y enteros.

2F142 DoNewMatrix ( 🡺 RealArry ) (flag 91 clear)

( 🡺 CmpArry ) (flag 91 clear)

( 🡺 MATRIX ) (flag 91 clear)

( 🡺 {{}} ) (flag 91 set )

( 🡺 )

Este comando es parecido a ˆRunDoNewMatrix, pero no retorna un flag. Este comando llama a ˆRunDoNewMatrix.

Este comando es llamado cuando uno presiona la tecla MTRW.

10.1.8 Editar Arreglos con el editor de matrices (MTRW)

Los comandos de esta sección tienen un comportamiento diferente según el estado del flag del sistema número 91.

Cuando el flag del sistema número 91 está activado, se acepta la entrada de objetos de cualquier tipo. Además, si uno sale del editor presionando ENTER, en la pila aparecerá una lista de listas y TRUE.

Cuando el flag del sistema número 91 está desactivado, se acepta sólo la entrada de objetos de algunos tipos como reales, complejos, ids, lams, symbs y enteros. Además, si uno sale del editor presionando ENTER, en la pila aparecerá un arreglo o una matriz simbólica y TRUE.

Si uno se retira del editor presionando CANCL, en la pila sólo aparecerá FALSE.

**Direcc. Nombre Descripción**

009007 ˆDoOldMatrixReal ( Arry/MATRIX 🡺 RealArry T )(flag 91 clear)

( Arry/MATRIX 🡺 {{}} T )(flag 91 set)

( Arry/MATRIX 🡺 F )

Abre el editor MTRW para editar una formación.

Con el flag 91 desactivado, sólo se aceptará la entrada de números reales y en la pila se devuelve un arreglo real.

Devuelve FALSE cuando uno se retira del MTRW.

00A007 ˆDoOldMatrixCplx ( Arry/MATRIX 🡺 CmpArry T ) (flag 91 clear)

( Arry/MATRIX 🡺 {{}} T ) (flag 91 set)

( Arry/MATRIX 🡺 F )

Abre el editor MTRW para editar una formación.

Con el flag 91 desactivado, sólo se aceptará la entrada de números complejos y en la pila se devuelve un arreglo complejo.

Devuelve FALSE cuando uno se retira del MTRW.

005007 ˆRunDoOldMatrix ( Arry/Matrix/{{}} 🡺 RealArry T )(flag 91 clr)

( Arry/Matrix/{{}} 🡺 CmpArry T )(flag 91 clr)

( Arry/Matrix/{{}} 🡺 MATRIX T )(flag 91 clr)

( Arry/Matrix/{{}} 🡺 {{}} T )(flag 91 set)

( Arry/Matrix/{{}} 🡺 F )

Abre el editor MTRW para editar una formación o una lista de listas.

Con el flag 91 desactivado, retorna un arreglo real, arreglo complejo, o matriz simbólica. Si todos los objetos son reales, crea un arreglo real. Si todos son complejos, crea un arreglo complejo. En otro caso, crea una matriz simbólica. Sólo se permite el ingreso de reales, complejos, ids, lams, simbólicos y enteros.

Devuelve FALSE cuando uno se retira del MTRW.

2F13C DoOldMatrix ( Arry/Matrix/{{}} 🡺 RealArry T )(flag 91 clr)

( Arry/Matrix/{{}} 🡺 CmpArry T )(flag 91 clr)

( Arry/Matrix/{{}} 🡺 MATRIX T )(flag 91 clr)

( Arry/Matrix/{{}} 🡺 {{}} T )(flag 91 set)

( Arry/Matrix/{{}} 🡺 F )

Este comando hace exactamente lo mismo que el anterior comando, ˆRunDoOldMatrix.

DoOldMatrix llama a ˆRunDoOldMatrix.

10.1.7 Copiar varios elementos de un arreglo a otra posición o a otro arreglo.

**Direcc. Nombre Descripción**

05F003 FLASHPTR 003 05F ( #fila [[%]] [[%]]' 🡺 [[%]]'' )

( #fila [[C%]] [[C%]]' 🡺 [[C%]]'' )

Reemplaza el arreglo del nivel 1 (o parte de el, si es muy grande) en el arreglo del nivel 2 a partir de la fila #fila. Ambos arreglos deben tener el mismo número de columnas.

(comando tipo bang)

06E003 ˆlaGPROW ( [[%]] [%] #fila T 🡺 [[%]]' [%] )

( [[C%]] [C%] #fila T 🡺 [[C%]]' [C%] )

( [[%]] [%] #fila F 🡺 [[%]] [%]' )

( [[C%]] [C%] #fila F 🡺 [[C%]] [C%]' )

Si el flag es TRUE, reemplaza el vector en la fila especificada del arreglo de dos dimensiones.

Si el flag es FALSE, reemplaza los elementos de la fila especificada del arreglo de dos dimensiones en el vector.

El número de columnas del arreglo de dos dimensiones debe ser igual al número de elementos del arreglo de una dimensión.

(comando tipo bang)

06D003 ˆlaINSROW ( #pos [%] 🡺 #pos [%]' )

( #pos [C%] 🡺 #pos [C%]' )

( #fila [[%]] 🡺 #fila [[%]]' )

( #fila [[C%]] 🡺 #fila [[C%]]' )

Para arreglos de una dimensión, copia cada objetos de las posiciones #pos...#n-1, a la celda que está una posición a la derecha.

#pos está entre #1 y #n inclusive.

Si #pos=#n, entonces no hace nada.

Para arreglos de dos dimensiones, copia las filas #fila...#m-1, a la fila que está debajo.

#fila está entre #1 y #m inclusive (m es el número de filas del arreglo de dos dimensiones).

Si #fila=#m, entonces no hace nada.

(comando tipo bang)

06C003 ˆlaDELROW ( [%] #pos 🡺 [[%]]' )

( [C%] #pos 🡺 [[C%]]' )

( [[%]] #fila 🡺 [[%]]' )

( [[C%]] #fila 🡺 [[C%]]' )

Para arreglos de una dimensión, copia los objetos de las posiciones #pos+1...#n, a la celda que está una posición a la izquierda.

#pos está entre #1 y #n inclusive.

Si #pos=#n, entonces no hace nada.

Para arreglos de dos dimensiones, copia las filas #fila+1...#m, a la fila que está arriba.

#fila está entre #1 y #m inclusive (m es el número de filas del arreglo de dos dimensiones).

Si #fila=#m, entonces no hace nada.

(comando tipo bang)

10.1.10 Estadística

**Direcc. Nombre Descripción**

2EEDA STATCLST ( 🡺 )

Borra la variable ∑DAT del directorio actual.

Equivale al comando **CL∑** de User RPL.

2EEDB STATSADD% ( % 🡺 )

Si ∑DAT tiene una columna, agrega el número real del nivel uno de la pila a ∑DAT debajo de la última fila.

Equivale al comando **∑+** de User RPL cuando en la pila hay un número real.

2EEDC STATN ( 🡺 %filas )

Retorna el número de filas de ∑DAT.

Equivale al comando **N∑** de User RPL.

2EEDF STATSMIN ( 🡺 % )

( 🡺 [%] )

Si ∑DAT tiene una columna, retorna el número mínimo.

Si ∑DAT tiene dos o más columnas, retorna un arreglo real con los números mínimos de cada columna.

Equivale al comando **MIN∑** de User RPL.

2EEDD STATSMAX ( 🡺 % )

( 🡺 [%] )

Si ∑DAT tiene una columna, retorna el número máximo.

Si ∑DAT tiene dos o más columnas, retorna un arreglo real con los números máximos de cada columna.

Equivale al comando **MAX∑** de User RPL.

2EEE1 STATTOT ( 🡺 % )

( 🡺 [%] )

Si ∑DAT tiene una columna, retorna la suma de sus elementos.

Si ∑DAT tiene dos o más columnas, retorna un arreglo real con la suma de los elementos de cada columna.

Equivale al comando **TOT** de User RPL.

2EEDE STATMEAN ( 🡺 % )

( 🡺 [%] )

Si ∑DAT tiene una columna, retorna la media aritmética.

Si ∑DAT tiene dos o más columnas, retorna un arreglo real con la media aritmética de los elementos de cada columna.

Equivale al comando **MEAN** de User RPL.

2EEE2 STATVAR ( 🡺 % )

( 🡺 [%] )

Si ∑DAT tiene una columna, retorna la varianza de muestra.

Si ∑DAT tiene dos o más columnas, retorna un arreglo real con la varianza de muestra de los elementos de cada columna.

∑DAT debe tener al menos dos filas.

Equivale al comando **VAR** de User RPL.

**Direcc. Nombre Descripción**

2EEE0 STATSTDEV ( 🡺 % )

( 🡺 [%] )

Si ∑DAT tiene una columna, retorna la desviación estándar de muestra de sus elementos.

Si ∑DAT tiene dos o más columnas, retorna un arreglo real con la desviación estándar de muestra de los elementos de cada columna.

∑DAT debe tener al menos dos filas.

Equivale al comando **SDEV** de User RPL.

098003 FLASHPTR 003 098 ( 🡺 % )

( 🡺 [%] )

Si ∑DAT tiene una columna, retorna la varianza de población.

Si ∑DAT tiene dos o más columnas, retorna un arreglo real con la varianza de población de los elementos de cada columna.

∑DAT debe tener al menos dos filas.

Equivale al comando **PVAR** de User RPL.

097003 FLASHPTR 003 097 ( 🡺 % )

( 🡺 [%] )

Si ∑DAT tiene una columna, retorna la desviación estándar de población de sus elementos.

Si ∑DAT tiene dos o más columnas, retorna un arreglo real con la desviación estándar de población de los elementos de cada columna.

∑DAT debe tener al menos dos filas.

Equivale al comando **PSDEV** de User RPL.

10.2 Ejemplos

Ejemplo 1 Arreglos Vinculados

**¿Cómo saber si un arreglo vinculado está completo o incompleto?**

Para saber si un arreglo vinculado está completo o incompleto, puedes usar lo siguiente.

\* Retorna TRUE si el arreglo vinculado está completo

NULLNAME LNKARRY\_FULL? ( LNKARRY -> flag )

:: ( LNKARRY )

TRUE ( LNKARRY TRUE )

1LAMBIND ( LNKARRY )

DUP ( LNKARRY LNKARRY )

ARSIZE\_ ( LNKARRY #elem )

#1+\_ONE\_DO ( LNKARRY )

INDEX@ ( LNKARRY #i )

OVER ( LNKARRY #i LNKARRY )

GETATELN ( LNKARRY obi T // LNKARRY F )

ITE

DROP

:: ExitAtLOOP FALSE 1PUTLAM ;

LOOP

( LNKARRY )

DROP ( )

1GETABND ( flag )

;

El siguiente NULLNAME hace lo mismo (sólo verifica que exista el último elemento).

\* Retorna TRUE si el arreglo vinculado está completo

\* Sólo verifica que exista el último elemento

NULLNAME LNKARRY\_FULL? ( LNKARRY -> flag )

:: ( LNKARRY )

DUP ( LNKARRY LNKARRY )

ARSIZE\_ ( LNKARRY #elem )

SWAP ( #elem LNKARRY )

GETATELN ( ob T // F )

NOTcaseFALSE ( sale con FALSE en la pila )

( ob )

DROPTRUE ( T )

;

Ejemplo 2 Arreglos

**Formar un arreglo con los objetos de la pila**

El siguiente NULLNAME es similar al comado ^XEQ>ARRY pero puede crear un arreglo de varias dimensiones y que contenga objetos de cualquier tipo.

\* Forma un arreglo a partir de N objetos de la pila.

\* Estos N objetos deben ser del mismo tipo.

\* Las dimensiones del arreglo están indicadas en la lista.

NULLNAME ob>ARRY ( ob1...obN {#L1,#L2...#Ln} -> ARRY )

:: ( ob1...obN {#} )

DUP ( ob1...obN {#} {#} )

>R ( ob1...obN {#} )

DUPLENCOMP ( ob1...obN {#} #n )

3PICK ( ob1...obN {#} #n obn )

FLASHPTR 002 0A5 ( ob1...obN {#} #n "Prolo..." )

BINT1 BINT5 SUB$ ( ob1...obN {#} #n "Prolo" )

OVER ( ob1...obN {#} #n "Prolo" #n )

#>HXS ( ob1...obN {#} #n "Prolo" HXSn )

FLASHPTR 002 0A7 ( ob1...obN {#} #n "Prolo" "Ndims" )

&$ ( ob1...obN {#} #n "ProloNdims" )

SWAP ( ob1...obN {#} "ProloNdims" #n )

ZERO\_DO (DO) ( ob1...obN {#} "ProloNdims" )

RSWAP 'R RSWAP ( ob1...obN {#} "ProloNdims" #Li )

#>HXS

FLASHPTR 002 0A7

&$ ( ob1...obN {#} "ProloNdims...Ldimi" )

LOOP

( ob1...obN {#} "ProloNdimsLdims" )

1LAMBIND ( ob1...obN {#} )

INNERCOMP ( ob1...obN #L1...#Ln #dims )

DUP#1= ( ob1...obN #L1...#Ln #dims flag )

ITE

DROP

:: ONE\_DO (DO) #\* LOOP ;

( ob1...obN #N )

DUPDUP ( ob1...obN #N #N #N )

#2+UNROLL ( #N ob1...obN #N )

::N

>R ( #N )

1GETABND

SWAP ( "ProloNdimsLdims" #N )

ZERO\_DO (DO) ( "ProloNdimsLdims" )

RSWAP 'R RSWAP ( "ProloNdimsLdims" obi )

FLASHPTR 002 0A5

BINT6

OVERLEN$

SUB$

&$ ( "ProloNdimsLdimsObjetosSinPrólogo" )

LOOP

( "ProloNdimsLdimsObjetosSinPrólogo" )

DUPLEN$ ( "ProloNdimsLdimsObjetosSinPrólogo" $tamaño )

#5+ ( "ProloNdimsLdimsObjetosSinPrólogo" $tamaño+5 )

#>HXS ( "ProloNdimsLdimsObjetosSinPrólogo" HXStamaño+5 )

FLASHPTR 002 0A7 ( "ProloNdimsLdimsObjetosSinPrólogo" "Tamañ" )

SWAP&$ ( "TamañProloNdimsLdimsObjetosSinPrólogo" )

"8E920" ( "TamañProloNdimsLdimsObjetosSinPrólogo" "8E920" )

SWAP&$ ( "8E920TamañProloNdimsLdimsObjetosSinPrólogo" )

FLASHPTR 002 0A4 ( ARRY )

;

Ejemplo 3 Arreglos y Arreglos Vinculados

**Desintegrar un arreglo o arreglo vinculado**

El siguiente NULLNAME es similar al comado ^XEQARRY> pero no funciona para matrices simbólicas y si funciona para cualquier arreglo (con varias dimensiones y que contengan objetos de cualquier tipo). También funciona para arreglos vinculados que estén completos.

\* Desintegra un arreglo o arreglo vinculado

\* Funciona para todo tipo de arreglos y para arreglos vinculados completos

\* ARRY -> ob1...obN {#L1,#L2...#Ln}

\* LNKARRY -> ob1...obN {#L1,#L2...#Ln}

NULLNAME ARRY>ob ( ARRY/LNKARRY -> ob1...obN {#L1,#L2...#Ln} )

:: ( ARRY )

DUP ( ARRY ARRY )

ARSIZE\_ ( ARRY #elem )

#1+\_ONE\_DO (DO)

( ... ARRY )

INDEX@ ( ... ARRY #i )

OVER ( ... ARRY #i ARRY )

GETATELN ( ... ARRY obi T )

DROPSWAP ( ... obi ARRY )

LOOP

( ob1...obN ARRY )

DIMLIMITS\_ ( ob1...obN {#L1,#L2...#Ln} )

;

El siguiente NULLNAME es similar al anterior pero también funciona para arreglos vinculados incompletos. Pone xNOVAL en el lugar de los elementos ausentes.

\* Desintegra un arreglo o arreglo vinculado

\* Funciona para todo tipo de arreglos y todo tipo de arreglos vinculados

\* ARRY -> ob1...obN {#L1,#L2...#Ln}

\* LNKARRY -> ob1...obN {#L1,#L2...#Ln}

NULLNAME ARRY>ob ( ARRY/LNKARRY -> ob1...obN {#L1,#L2...#Ln} )

:: ( LNKARRY )

DUP ( LNKARRY LNKARRY )

ARSIZE\_ ( LNKARRY #elem )

#1+\_ONE\_DO

( ... LNKARRY )

INDEX@ ( ... LNKARRY #i )

OVER ( ... LNKARRY #i ARRY )

GETATELN ( ... LNKARRY obi T // ... LNKARRY F )

NOT\_IT :: ' xNOVAL ;

( ... LNKARRY obi' )

SWAPLOOP

( ob1...obN LNKARRY )

DIMLIMITS\_ ( ob1...obN {#L1,#L2...#Ln} )

;

Ejemplo 4 Arreglos y Arreglos Vinculados

**Convertir un arreglo vinculado en arreglo**

El siguiente NULLNAME convierte arreglos vinculados “**completos**” a arreglos.

\* Convierte un arreglo vinculado **completo** a arreglo

\* Usa dos NULLNAME anteriores: **ARRY>ob** y **ob>ARRY**

NULLNAME LNKARRY>ARRY ( LNKARRY -> ARRY )

:: ( LNKARRY )

ARRY>ob ( ob1...obN {#L1,#L2...#Ln} )

ob>ARRY ( ARRY )

;

Ejemplo 5 Arreglos Reales y Arreglos Complejos

**Formar un arreglo real o complejo con los objetos de la pila**

El siguiente NULLNAME es similar al comado ^XEQ>ARRY pero puede crear un arreglo de varias dimensiones y que contenga objetos de cualquier tipo.

\* Crea un arreglo real o complejo a partir de los elementos de la pila

\* El arreglo creado puede tener cualquier número de dimensiones

\* a diferencia de ^XEQ>ARRY, el cual crea arreglos de 1 o 2 dimensiones

\* Este NULLNAME también es más rápido que ^XEQ>ARRY

NULLNAME ob>RealoCmpArry ( %/C%...%'/C%' {#L1...#Ln} -> RealArry/CmpArry )

:: ( ob1...obn {#L1,...,#Ln} )

DUPINCOMP ( ob1...obn {#L1,...,#Ln} #L1,...,#Ln #n )

:: DUP#1=

caseDROP

ONE\_DO (DO)

#\*

LOOP

;

( ob1...obn {#L1,...,#Ln} #elem )

SWAP ( ob1...obn #elem {#L1,...,#Ln} )

3PICK ( ob1...obn #elem {#L1,...,#Ln} %/C% )

MAKEARRY\_ ( ob1...obn #elem RealArry/CmpArry )

ReplaceMetainARRY ( RealArry'/CmpArry' )

;

NULLNAME ReplaceMetainARRY

\* ( ob1...obn #elem RealArry/CmpArry -> RealArry'/CmpArry' )

::

SWAP

#1+\_ONE\_DO (DO)

ISTOP-INDEX

#1+ROLL

INDEX@

PUTEL

LOOP

;

Ejemplo 6 Arreglos

**Obtener un elemento de un arreglo ubicado en la posición indicada por una lista.**

El siguiente NULLNAME permite obtener un elemento de un arreglo. La ubicación del elemento es indicado mediante una lista de bints.

\* Retorna un objeto de un arreglo o arreglo vinculado completo

\* Nivel 2: Arreglo o arreglo vinculado completo con objetos de

\* cualquier tipo.

\* Nivel 1: Lista que representa la posición del objeto.

\* Salida: El objeto.

NULLNAME GetFromARRY ( ARRY {#P1,#P2,…,#Pn} -> ob )

:: ( ARRY {#P1,#P2,…,#Pn} )

OVER ( ARRY {#P1,#P2,…,#Pn} ARRY )

FINDELN\_ ( ARRY # T )

DROPSWAP ( # ARRY )

GETATELN ( ob T )

DROP ( ob )

;

Por ejemplo:

:: { BINT5 BINT3 } GetFromARRY ;

Permite obtener el elemento ubicado en la fila 5 y en la columna 3 de un arreglo de dos dimensiones ubicado en la pila.

Ejemplo 7 Arreglos

**Reemplazar un elemento por otro objeto dentro de un arreglo**

El siguiente NULLNAME reemplaza un elemento de un arreglo. La ubicación del elemento es indicado mediante una lista de bints.

\* Reemplaza en el arreglo del nivel 3, el objeto que se encuentre en el

\* nivel 1 en la posición indicada en el nivel 2.

\* Funciona para cualquier tipo de arreglos, de cualquier número de

\* dimensiones y que contenga cualquier tipo de objetos.

\* Llama a tres subprogramas: ARRY>ob, MetaRepl\_Arriba y ob>ARRY

\* El subprograma MetaRepl\_Arriba también se encuentra al final del capítulo 12.

NULLNAME ReplaceInARRY ( ARRY {#P1,#P2,…#Pn} ob -> ARRY' )

:: ( ARRY {#P1,#P2,…#Pn} ob )

SWAP3PICK ( ARRY ob {#P1,#P2,…#Pn} ARRY )

ARRY>ob ( ARRY ob {#P1,#P2,…#Pn} ob1...obN {#L1,#L2,…#Ln} )

DUP ( ARRY ob {#P1,#P2,…#Pn} ob1...obN {#L1,#L2,…#Ln} {#L1,#L2,…#Ln} )

INNERCOMP ( ARRY ob {#P1,#P2,…#Pn} ob1...obN {#L1,#L2,…#Ln} #L1,#L2,…#Ln #n )

:: DUP#1= caseDROP

ONE\_DO (DO)

#\*

LOOP

;

( ARRY ob {#P1,#P2,…#Pn} ob1...obN {#L1,#L2,…,#Ln} #N )

SWAPOVER ( ARRY ob {#P1,#P2,…#Pn} ob1...obN #N {#L1,#L2,…,#Ln} #N )

#5+ ( ARRY ob {#P1,#P2,…#Pn} ob1...obN #N {#L1,#L2,…,#Ln} #N+5 )

UNROLL ( {#L1,#L2,…,#Ln} ARRY ob {#P1,#P2,…,#Pn} ob1...obN #N )

DUP #3+ ( {#L1,#L2,…,#Ln} ARRY ob {#P1,#P2,…,#Pn} ob1...obN #N #N+3 )

ROLL ( {#L1,#L2,…,#Ln} ARRY {#P1,#P2,…,#Pn} ob1...obN #N ob )

OVER #3+ ( {#L1,#L2,…,#Ln} ARRY {#P1,#P2,…,#Pn} ob1...obN #N ob #N+3 )

ROLL ( {#L1,#L2,…,#Ln} ARRY ob1...obN #N ob {#P1,#P2,…,#Pn} )

3PICK #4+ ( {#L1,#L2,…,#Ln} ARRY ob1...obN #N ob {#P1,#P2,…,#Pn} #N+4 )

ROLL ( {#L1,#L2,…,#Ln} ob1...obN #N ob {#P1,#P2,…,#Pn} ARRY )

ARRYEL?\_ ( {#L1,#L2,…,#Ln} ob1...obN #N ob #pos T )

DROP ( {#L1,#L2,…,#Ln} ob1...obN #N ob #pos )

MetaRepl\_Arriba ( {#L1,#L2,…,#Ln} ob1...obN #N )

#1+ROLL ( ob1...obN {#L1,#L2,…,#Ln} )

ob>ARRY ( ARRY' )

;

\* Reemplaza el objeto de la posición i del meta por el del nivel 2

\* (cuenta desde ARRIBA)

\* i: 1,2,...,n

NULLNAME MetaRepl\_Arriba ( meta ob #i -> meta' )

:: ( meta ob #i )

3PICK ( meta ob #i #n )

SWAP#- ( meta ob #n-i )

#2+ ( meta ob #n-i+3 )

UNPICK\_ ( meta' )

;

Ejemplo 8 Arreglos

**Aplicar un programa o comando a cada elemento de un arreglo**

**Caso 1: General**

El arreglo inicial y el arreglo final tienen cualquier dimensión y tienen objetos de cualquier tipo.

\* Evalúa un programa o comando sobre cada elemento de un arreglo

\* Entrada:

\* Nivel 2: Un arreglo

\* Nivel 1: Un programa o comando que tome un argumento y devuelva un objeto

\* Salida: Un arreglo con las mismas dimensiones que el arreglo original

\*

\* Para llamar a la posición del elemento usar **INDEX@**

\* Para llamar al número de elementos del arreglo usa **ISTOP@ #1-**

\* Este NULLNAME usa un NULLNAME anterior: **ob>ARRY**

NULLNAME Prog1a1ARRY ( ARRY Prog1a1 -> ARRY' )

:: ( ARRY Prog1a1 )

OVER ( ARRY Prog1a1 ARRY )

ARSIZE\_ ( ARRY Prog1a1 #elem )

#1+\_ONE\_DO (DO)

( ... ARRY Prog1a1 )

INDEX@ ( ... ARRY Prog1a1 #i )

3PICK ( ... ARRY Prog1a1 #i ARRY )

GETATELN ( ... ARRY Prog1a1 obi T )

DROP ( ... ARRY Prog1a1 obi )

OVER ( ... ARRY Prog1a1 obi Prog1a1 )

EVAL ( ... ARRY Prog1a1 obi' )

UNROT ( ... obi' ARRY Prog1a1 )

LOOP

( ob1...obN ARRY Prog1a1 )

DROP

DIMLIMITS\_ ( ob1...obN {#L1,#L2...#Ln} )

ob>ARRY ( ARRY' )

;

Por ejemplo:

ARRY 4 2 [ # B # C # D # E # F # 10 # 11 # 12 ]

' :: INDEX@ #>$ "/" &$ ISTOP@ #1- #>$ &$ ": " &$ SWAP #>$ &$ ;

Prog1a1ARRY

Devuelve:

[[ "1/8: 11" "2/8: 12" ]

[ "3/8: 13" "4/8: 14" ]

[ "5/8: 15" "6/8: 16" ]

[ "7/8: 17" "8/8: 18" ]]

Ejemplo 9 Arreglos

**Aplicar un programa o comando a cada elemento de un arreglo**

**Caso 2: Arreglo real de 1 o de 2 dimensiones**

El arreglo inicial y el arreglo final son arreglos reales y sólo tienen una o dos dimensiones.

Este NULLNAME es más rápido que el del caso general, usarlo cuando sea posible.

\* Evalúa un programa o comando sobre cada elemento de un arreglo real

\* Este arreglo debe tener 1 o 2 dimensiones

\* Entrada:

\* Nivel 2: Un arreglo real

\* Nivel 1: Un programa o comando de la forma: ( % -> %' )

\* Salida: Un arreglo real con las mismas dimensiones que el

\* arreglo original

\* Para llamar a la posición del elemento usar INDEX@

\* Para llamar al número de elementos del arreglo usa ISTOP@ #1-

NULLNAME Prog1a1RealArry ( RARRY Prog1a1 -> RARRY' )

:: ( RARRY Prog1a1 )

SWAP ( Prog1a1 RARRY )

CKREF ( Prog1a1 RARRY )

DUP ( Prog1a1 RARRY RARRY )

ARSIZE\_ ( Prog1a1 RARRY #elem )

#1+\_ONE\_DO (DO)

( Prog1a1 RARRY )

INDEX@ ( Prog1a1 RARRY #i )

PULLREALEL ( Prog1a1 RARRY % )

3PICK ( Prog1a1 RARRY % Prog1a1 )

EVAL ( Prog1a1 RARRY %' )

INDEX@ ( Prog1a1 RARRY %' #i )

PUTREALEL ( Prog1a1 RARRY' )

LOOP

( Prog1a1 RARRY' )

SWAPDROP ( RARRY' )

;

Por ejemplo:

ARRY 4 3 [ 11. 12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. ]

' :: %2 %\* ;

Prog1a1RealArry

Devuelve:

[[ 22. 24. 26. ]

[ 28. 30. 32. ]

[ 34. 36. 38. ]

[ 40. 42. 44. ]]

Ejemplo 10 Arreglos

**Aplicar un programa o comando a cada elemento de un arreglo**

**Caso 3: Arreglo complejo de 1 o de 2 dimensiones**

El arreglo inicial y el arreglo final son arreglos complejos y sólo tienen una o dos dimensiones.

Este NULLNAME es más rápido que el del caso general, usarlo cuando sea posible.

\* Evalúa un programa o comando sobre cada elemento de un arreglo complejo

\* Este arreglo debe tener 1 o 2 dimensiones

\* Entrada:

\* Nivel 2: Un arreglo complejo

\* Nivel 1: Un programa o comando de la forma: ( C% -> C%' )

\* Salida: Un arreglo real con las mismas dimensiones que el

\* arreglo original

\* Para llamar a la posición del elemento usar INDEX@

\* Para llamar al número de elementos del arreglo usa ISTOP@ #1-

NULLNAME Prog1a1CmpArry ( CARRY Prog1a1 -> CARRY' )

:: ( CARRY Prog1a1 )

SWAP ( Prog1a1 CARRY )

CKREF ( Prog1a1 CARRY )

DUP ( Prog1a1 CARRY CARRY )

ARSIZE\_ ( Prog1a1 CARRY #elem )

#1+\_ONE\_DO (DO)

( Prog1a1 CARRY )

INDEX@ ( Prog1a1 CARRY #i )

PULLCMPEL ( Prog1a1 CARRY % )

3PICK ( Prog1a1 CARRY % Prog1a1 )

EVAL ( Prog1a1 CARRY %' )

INDEX@ ( Prog1a1 CARRY %' #i )

PUTCMPEL ( Prog1a1 CARRY' )

LOOP

( Prog1a1 CARRY' )

SWAPDROP ( CARRY' )

;

Ejemplo 11 Arreglos

**Aplicar un programa o comando a cada elemento de un arreglo**

**Caso 4: Arreglo final real o complejo**

El arreglo inicial y el arreglo final tienen una o dos dimensiones.

El arreglo inicial tiene objetos de cualquier tipo y el arreglo final tiene reales o complejos.

Este NULLNAME sólo es más rápido que el del caso general.

\* Evalúa un programa o comando sobre cada elemento de un arreglo

\* Entrada:

\* Nivel 2: Un arreglo (con objetos cualesquiera) de una o dos dimensiones

\* Nivel 1: Un programa o comando que tome un argumento y devuelva un número

\* real o complejo

\* Salida:

\* Un arreglo real o complejo con las mismas dimensiones que el arreglo

\* original

\* Para llamar a la posición del elemento usar INDEX@

\* Para llamar al número de elementos del arreglo usa ISTOP@ #1-

NULLNAME Prog1a1ARRY>NumArry ( Arry Prog1a1 -> RealArry/CmpArry )

:: ( Arry Prog1a1 )

1LAMBIND ( Arry )

DUP ( Arry Arry )

ARSIZE\_ ( Arry #el )

#1+\_ONE\_DO (DO)

( ... Arry )

INDEX@ ( ... Arry #i )

OVER ( ... Arry #i Arry )

GETATELN ( ... Arry obi T )

DROP ( ... Arry obi )

1GETLAM EVAL ( ... Arry obi' )

SWAP ( ... obi' Arry )

LOOP

( ob1...obN Arry )

DIMLIMITS\_ ( ob1...obN {#el} // ob1...obN {#f #c} )

INNER#1=

ITE

:: UNCOERCE ONE{}N ;

:: UNCOERCE2 TWO{}N ;

( ob1...obN {%el} // ob1...obN {%f %c} )

FLASHPTR XEQ>ARRY ( RealArry // CmpArry )

ABND ( RealArry // CmpArry )

;

Por ejemplo:

ARRY 2 3 [ "ZZ" "TEXTO SWE" "ABC" "XYZ" "A1" "B5" ]

' :: LEN$ INDEX@ UNCOERCE2 %>C% ;

MAP\_Arry\_SysRPL

Devuelve:

[[ (2.,1.) (9.,2.) (3.,3.) ]

[ (3.,4.) (2.,5.) (2.,6.) ]]

Ejemplo 12 Arreglos

**Aplicar un programa o comando a los elementos de 2 arreglos del mismo tamaño**

**Caso 1: General**

Tanto los 2 arreglos iniciales como el arreglo final tienen objetos de cualquier tipo y pueden tener cualquier número de dimensiones.

\* Evalúa un programa o comando a los elementos de 2 arreglos

\* Entrada:

\* Nivel 3: Primer arreglo

\* Nivel 2: Segundo arreglo

\* Nivel 1: Un programa o comando que tome 2 argumentos y devuelva 1 objeto

\* Salida: Un arreglo con las mismas dimensiones que los arreglos originales

\*

\* Para llamar a la posición del elemento usa: INDEX@

\* Para llamar al número de elementos de cada arreglo usa: ISTOP@ #1-

\* Este NULLNAME usa un NULLNAME anterior: ob>ARRY

NULLNAME Prog2a1ARRY ( ARRY1 ARRY2 Prog2a1 -> ARRY' )

:: ( ARRY1 ARRY2 Prog2a1 )

3PICK ( ARRY1 ARRY2 Prog2a1 ARRY1 )

FLASHPTR 3LAMBIND ( ARRY1 )

ARSIZE\_ ( #elem )

#1+\_ONE\_DO (DO)

( ... )

INDEX@ ( ... #i )

1GETLAM ( ... #i ARRY1 )

GETATELN ( ... ob1i T )

DROP ( ... ob1i )

INDEX@ ( ... ob1i #i )

3GETLAM ( ... ob1i #i ARRY2 )

GETATELN ( ... ob1i ob2i T )

DROP ( ... ob1i ob2i )

2GETEVAL ( ... obi )

LOOP

( ob1...obN )

1GETABND ( ob1...obN ARRY1 )

DIMLIMITS\_ ( ob1...obN {#L1,#L2...#Ln} )

ob>ARRY ( ARRY' )

;

Por ejemplo, el siguiente programa retorna un arreglo donde cada elemento es la suma de los cuadrados de los elementos de los 2 arreglos.

ARRY 4 2 [ 11. 12. 13. 14. 15. 16. 17. 18. ]

ARRY 4 2 [ 1. 2. 3. 4. 5. 6. 7. 8. ]

' :: %SQ\_ SWAP %SQ\_ %+ ;

Prog2a1ARRY

Retorna:

[[ 122. 148. ]

[ 178. 212. ]

[ 250. 292. ]

[ 338. 388. ]]

Ejemplo 13 Arreglos

**Aplicar un programa o comando a los elementos de 2 arreglos del mismo tamaño**

**Caso 2: Arreglos reales de 1 o de 2 dimesiones**

Tanto los 2 arreglos iniciales como el arreglo final son arreglos reales de 1 o de 2 dimensiones.

Este NULLNAME es más rápido que el del caso general, usarlo cuando sea posible.

\* Evalúa un programa o comando a los elementos de 2 arreglos reales

\* del mismo tamaño.

\* Estos arreglos deben tener 1 o 2 dimensiones

\* Entrada:

\* Nivel 3: Primer arreglo real

\* Nivel 2: Segundo arreglo real

\* Nivel 1: Un programa o comando de la forma ( % %' -> %'' )

\* Salida: Arreglo real con las mismas dimensiones que los

\* arreglos originales

\*

\* Para llamar a la posición del elemento usa: INDEX@

\* Para llamar al número de elementos de cada arreglo usa: ISTOP@ #1-

NULLNAME Prog2a1RealArry ( RARRY1 RARRY2 Prog2a1 -> RARRY' )

:: ( RARRY1 RARRY2 Prog2a1 )

UNROTDUP ( Prog2a1 RARRY1 RARRY2 RARRY2 )

TOTEMPSWAP ( Prog2a1 RARRY1 RARRY2 RARRY2 )

ARSIZE\_ ( Prog2a1 RARRY1 RARRY2 #elem )

#1+\_ONE\_DO (DO)

( Prog2a1 RARRY1 RARRY2 )

SWAPINDEX@ ( Prog2a1 RARRY2 RARRY1 #i )

PULLREALEL ( Prog2a1 RARRY2 RARRY1 % )

ROT ( Prog2a1 RARRY1 % RARRY2 )

INDEX@ ( Prog2a1 RARRY1 % RARRY2 #i )

PULLREALEL ( Prog2a1 RARRY1 % RARRY2 %' )

ROTSWAP ( Prog2a1 RARRY1 RARRY2 % %' )

5PICK ( Prog2a1 RARRY1 RARRY2 % %' prog2a1 )

EVAL ( Prog2a1 RARRY1 RARRY2 %'' )

INDEX@ ( Prog2a1 RARRY1 RARRY2 %'' #i )

PUTREALEL ( Prog1a1 RARRY1 RARRY2' )

LOOP

( Prog1a1 RARRY1 RARRY' )

UNROT2DROP ( RARRY' )

;

Por ejemplo, el siguiente programa retorna un arreglo donde cada elemento es la suma de los cuadrados de los elementos de los 2 arreglos.

ARRY 4 2 [ 11. 12. 13. 14. 15. 16. 17. 18. ]

ARRY 4 2 [ 1. 2. 3. 4. 5. 6. 7. 8. ]

' :: %SQ\_ SWAP %SQ\_ %+ ;

Prog2a1RealArry

Retorna:

[[ 122. 148. ]

[ 178. 212. ]

[ 250. 292. ]

[ 338. 388. ]]

Ejemplo 14 Arreglos

**Aplicar un programa o comando a los elementos de 2 arreglos del mismo tamaño**

**Caso 3: Arreglos complejos de 1 o de 2 dimesiones**

Tanto los 2 arreglos iniciales como el arreglo final son arreglos complejos de 1 o de 2 dimensiones.

Este NULLNAME es más rápido que el del caso general, usarlo cuando sea posible.

\* Evalúa un programa o comando a los elementos de 2 arreglos complejos

\* del mismo tamaño.

\* Estos arreglos deben tener 1 o 2 dimensiones

\* Entrada:

\* Nivel 3: Primer arreglo complejo

\* Nivel 2: Segundo arreglo complejo

\* Nivel 1: Un programa o comando de la forma ( C% C%' -> C%'' )

\* Salida: Arreglo complejo con las mismas dimensiones que los

\* arreglos originales

\*

\* Para llamar a la posición del elemento usa: INDEX@

\* Para llamar al número de elementos de cada arreglo usa: ISTOP@ #1-

NULLNAME Prog2a1CmpArry ( CARRY1 CARRY2 Prog2a1 -> CARRY' )

:: ( CARRY1 CARRY2 Prog2a1 )

UNROTDUP ( Prog2a1 CARRY1 CARRY2 CARRY2 )

TOTEMPSWAP ( Prog2a1 CARRY1 CARRY2 CARRY2 )

ARSIZE\_ ( Prog2a1 CARRY1 CARRY2 #elem )

#1+\_ONE\_DO (DO)

( Prog2a1 CARRY1 CARRY2 )

SWAPINDEX@ ( Prog2a1 CARRY2 CARRY1 #i )

PULLCMPEL ( Prog2a1 CARRY2 CARRY1 % )

ROT ( Prog2a1 CARRY1 % CARRY2 )

INDEX@ ( Prog2a1 CARRY1 % CARRY2 #i )

PULLCMPEL ( Prog2a1 CARRY1 % CARRY2 %' )

ROTSWAP ( Prog2a1 CARRY1 CARRY2 % %' )

5PICK ( Prog2a1 CARRY1 CARRY2 % %' prog2a1 )

EVAL ( Prog2a1 CARRY1 CARRY2 %'' )

INDEX@ ( Prog2a1 CARRY1 CARRY2 %'' #i )

PUTCMPEL ( Prog1a1 CARRY1 CARRY2' )

LOOP

( Prog1a1 CARRY1 CARRY' )

UNROT2DROP ( CARRY' )

;

Por ejemplo, el siguiente programa retorna un arreglo complejo donde cada elemento es la suma de los elementos de los 2 arreglos complejos.

ARRY 2 2 [ C% 11. 12. C% 13. 14. C% 15. 16. C% 17. 18. ]

ARRY 2 2 [ C% 1. 2. C% 3. 4. C% 5. 6. C% 7. 8. ]

' :: C%>% ROT C%>% ROT %+ 3UNROLL %+SWAP %>C% ;

Prog2a1CmpArry

Retorna:

[[ (12.,14.) (16.,18.) ]

[ (20.,22.) (24.,26.) ]]

Ejemplo 15 Arreglos

**Aplicar un programa o comando a los elementos de varios arreglos**

**Caso 1: General**

Tanto los arreglos iniciales como el arreglo final tienen objetos de cualquier tipo y pueden tener cualquier número de dimensiones.

\* Evalúa un programa o comando a los elementos de varios arreglos

\* Entrada:

\* Nivel N+2,...,5,4,3: N Arreglos, todos de las mismas dimensiones.

\* De cualquier dimens y con cualquier tipo de objeto

\* Nivel 2: Un bint (#N) Indica el número de arreglos

\* Nivel 1: Programa o comando que tome N argumentos y devuelva un objeto

\* Salida:

\* Nivel 1: Un arreglo con las mismas dimensiones que los arreglos

\* originales. Puede contener objetos de cualquier tipo

\* Para llamar a la posición del elemento usar INDEX@ #1+

\* Para llamar al número de elementos de los arreglos usa ISTOP@

\* Este NULLNAME usa un NULLNAME anterior: ob>ARRY

NULLNAME ProgNa1ARRY ( ARRY1...ARRYN #N ProgNa1 -> ARRY' )

:: ( ARRY1...ARRYN #N Prog\_Nargum\_1salida )

SWAP ( ARRY1...ARRYN ProgNa1 #N )

FLASHPTR 2LAMBIND ( ARRY1...ARRYN )

DUP ( ARRY1...ARRYN ARRYN )

ARSIZE\_ ( ARRY1...ARRYN #elem )

#1+\_ONE\_DO (DO) ( ...ARRY1...ARRYN )

1GETLAM ( ...ARRY1...ARRYN #N )

#1+\_ONE\_DO (DO) ( ...ARRY1...ARRYN ... )

1GETLAM ( ...ARRY1...ARRYN ... #N )

PICK ( ...ARRY1...ARRYN ... ARRYi )

JINDEX@ ( ...ARRY1...ARRYN ... ARRYi #j )

SWAP ( ...ARRY1...ARRYN ... #j ARRYi )

GETATELN ( ...ARRY1...ARRYN ... obij T )

DROP ( ...ARRY1...ARRYN ... obij )

LOOP

( ...ARRY1...ARRYN ob1j,ob2j...obNj )

2GETEVAL ( ...ARRY1...ARRYN obj' )

1GETLAM ( ...ARRY1...ARRYN obj' #N )

#1+UNROLL ( ...ARRY1...ARRYN )

LOOP

( ob1',ob2'...obelem' ARRY1...ARRYN )

1GETABND ( ob1',ob2'...obelem' ARRY1...ARRYN #N )

#1- ( ob1',ob2'...obelem' ARRY1...ARRYN #N-1 )

NDROP ( ob1',ob2'...obelem' ARRYN )

DIMLIMITS\_ ( ob1',ob2'...obelem' {#L1,#L2...#Ln} )

ob>ARRY ( ARRY' )

;

Por ejemplo:

ARRY [ 12. 13. 14. ]

ARRY [ 100. 200. 300. ]

BINT2

' :: %+ % 1000. %\* INDEX@ UNCOERCE %+ ;

Opera\_N\_ARRY

Devuelve:

[ 112001. 213002. 314003. ]

Ejemplo 16 Arreglos

**Aplicar un programa o comando a los elementos de varios arreglos**

**Caso 2: Arreglos reales de 1 o de 2 dimensiones**

Tanto los arreglos iniciales como el arreglo final son arreglos reales de 1 o de 2 dimensiones.

Este NULLNAME es más rápido que el del caso general, usarlo cuando sea posible.

\* Evalúa un programa o comando a los elementos de varios arreglos reales

\* Estos arreglos deben tener 1 o 2 dimensiones

\* Entrada:

\* Nivel N+2,...,5,4,3: N Arreglos reales, todos de las mismas dimensiones.

\* De cualquier dimens

\* Nivel 2: Un bint (#N) Indica el número de arreglos

\* Nivel 1: Programa o comando que tome N argumentos reales

\* y devuelva un real

\* Salida:

\* Nivel 1: Un arreglo con las mismas dimensiones que los arreglos

\* originales. Puede contener objetos de cualquier tipo

\* Para llamar a la posición del elemento usar INDEX@ #1+

\* Para llamar al número de elementos de los arreglos usa ISTOP@

NULLNAME ProgNa1RealArry ( RARRY1...RARRYN #N ProgNa1 -> RARRY' )

:: ( RARRY1...RARRYN #N ProgNa1 )

SWAP ( RARRY1...RARRYN ProgNa1 #N )

FLASHPTR 2LAMBIND ( RARRY1...RARRYN )

TOTEMPOB ( RARRY1...RARRYN )

DUP ( RARRY1...RARRYN RARRYN )

ARSIZE\_ ( RARRY1...RARRYN #elem )

#1+\_ONE\_DO (DO)

( RARRY1...RARRYN )

1GETLAM ( RARRY1...RARRYN #N )

#1+\_ONE\_DO (DO)

( ...RARRYN... ... )

1GETLAM ( ...RARRYN... ... #N )

INDEX@ #+-1

ROLL ( ...RARRYN... ... RARRYi )

JINDEX@ ( ...RARRYN... ... RARRYi #elem )

PULLREALEL ( ...RARRYN... ... RARRYi % )

SWAPINDEX@ ( ...RARRYN... ... % RARRYi #i )

#1+UNROLL ( ...RARRYN...RARRYi ... % )

LOOP

( RARRY1...RARRYN %1...%N )

2GETEVAL ( RARRY1...RARRYN %' )

INDEX@ ( RARRY1...RARRYN %' #elem )

PUTREALEL ( RARRY1...RARRYN' )

LOOP

( RARRY1...RARRYN )

1GETLAM ( RARRY1...RARRYN #N )

UNROLL ( RARRYN RARRY1...RARRYN-1 )

1GETABND ( RARRYN RARRY1...RARRYN-1 #N )

#1- ( RARRYN RARRY1...RARRYN-1 #N-1 )

NDROP ( RARRYN )

;

Ejemplo 17 Arreglos

**Aplicar un programa o comando a los elementos de varios arreglos**

**Caso 3: Arreglos complejos de 1 o de 2 dimesiones**

Tanto los arreglos iniciales como el arreglo final son arreglos complejos de 1 o de 2 dimensiones.

Este NULLNAME es más rápido que el del caso general, usarlo cuando sea posible.

\* Evalúa un programa o comando a los elementos de varios arreglos complejos

\* Estos arreglos deben tener 1 o 2 dimensiones

\* Entrada:

\* Nivel N+2,...,5,4,3: N Arreglos complejos, todos de las mismas dimens

\* De cualquier dimens

\* Nivel 2: Un bint (#N) Indica el número de arreglos

\* Nivel 1: Programa o comando que tome N argumentos complejos

\* y devuelva un complejo

\* Salida:

\* Nivel 1: Un arreglo con las mismas dimensiones que los arreglos

\* originales. Puede contener objetos de cualquier tipo

\* Para llamar a la posición del elemento usar INDEX@ #1+

\* Para llamar al número de elementos de los arreglos usa ISTOP@

NULLNAME ProgNa1CmpArry ( CARRY1...CARRYN #N ProgNa1 -> CARRY' )

:: ( CARRY1...CARRYN #N ProgNa1 )

SWAP ( CARRY1...CARRYN ProgNa1 #N )

FLASHPTR 2LAMBIND ( CARRY1...CARRYN )

TOTEMPOB ( CARRY1...CARRYN )

DUP ( CARRY1...CARRYN CARRYN )

ARSIZE\_ ( CARRY1...CARRYN #elem )

#1+\_ONE\_DO (DO)

( CARRY1...CARRYN )

1GETLAM ( CARRY1...CARRYN #N )

#1+\_ONE\_DO (DO)

( ...CARRYN... ... )

1GETLAM ( ...CARRYN... ... #N )

INDEX@ #+-1

ROLL ( ...CARRYN... ... CARRYi )

JINDEX@ ( ...CARRYN... ... CARRYi #elem )

PULLCMPEL ( ...CARRYN... ... CARRYi % )

SWAPINDEX@ ( ...CARRYN... ... % CARRYi #i )

#1+UNROLL ( ...CARRYN...CARRYi ... % )

LOOP

( CARRY1...CARRYN %1...%N )

2GETEVAL ( CARRY1...CARRYN %' )

INDEX@ ( CARRY1...CARRYN %' #elem )

PUTCMPEL ( CARRY1...CARRYN' )

LOOP

( CARRY1...CARRYN )

1GETLAM ( CARRY1...CARRYN #N )

UNROLL ( CARRYN CARRY1...CARRYN-1 )

1GETABND ( CARRYN CARRY1...CARRYN-1 #N )

#1- ( CARRYN CARRY1...CARRYN-1 #N-1 )

NDROP ( CARRYN )

;

Ejemplo 18 Arreglos

**Obtener una fila de un arreglo de dos dimensiones**

**Caso 1: General**

El arreglo tiene objetos de cualquier tipo.

\* Nivel 2: Arreglo de dos dimensiones, con objetos de cualquier tipo.

\* Nivel 1: Bint que representa la fila que queremos obtener.

\* Salida: La fila deseada como un arreglo de una dimensión.

\* Este NULLNAME usa un NULLNAME anterior: ob>ARRY

NULLNAME Get\_Row\_Arry ( [[]] #fila -> [] )

:: ( [[]] #fila )

SWAPDUP ( #fila [[]] [[]] )

DIMLIMITS\_ ( #fila [[]] {#F #C} )

TWONTHCOMPDROP\_ ( #fila [[]] #C )

FLASHPTR 2LAMBIND ( #fila )

1GETLAM #\* ( #fila•C )

#1+ ( #fila•C+1 )

DUP ( #fila•C+1 #fila•C+1 )

1GETLAM #- ( #fila•C+1 #fila•C+1-C )

DO

INDEX@ ( ...#i )

2GETLAM ( ...#i [[]] )

GETATELN ( ...obi T )

DROP ( ...obi )

LOOP

( ob1...obC )

1GETABND ( ob1...obC #C )

ONE{}N ( ob1...obC {#C} )

ob>ARRY ( [ob1...obC] )

;

Ejemplo 19 Arreglos

**Obtener una fila de un arreglo de dos dimensiones**

**Caso 2: Arreglo numérico**

Este NULLNAME es mucho más rápido que el anterior, pero sólo funciona para arreglos reales o complejos.

\* Nivel 2: Arreglo de dos dimensiones, debe ser real o complejo.

\* Nivel 1: Bint que representa la fila que queremos obtener.

\* Salida: La fila deseada como un arreglo de una dimensión.

NULLNAME Get\_Row\_NumArry ( [[]] #fila -> [] )

:: ( [[]] #fila )

OVER ( [[]] #fila [[]] )

DIMLIMITS\_ ( [[]] #fila {#F #C} )

TWONTHCOMPDROP\_ ( [[]] #fila #C )

ONE{}N ( [[]] #fila {#C} )

3PICK ( [[]] #fila {#C} [[]] )

laMGET0 ( [[]] #fila {#C} %0/C%0 )

MAKEARRY\_ ( [[]] #fila [] )

SWAPFALSE\_ ( [[]] [] #fila F )

FLASHPTR laGPROW ( [[]] []' )

SWAPDROP ( []' )

;

Ejemplo 20 Arreglos

**Obtener una columna de un arreglo de dos dimensiones**

**Caso 1: General**

El arreglo tiene objetos de cualquier tipo.

\* Nivel 2: Arreglo de dos dimensiones, con objetos de cualquier tipo.

\* Nivel 1: Bint que representa la columna que queremos obtener.

\* Salida: La columna deseada como un arreglo de una dimensión.

\* Este NULLNAME usa un NULLNAME anterior: ob>ARRY

NULLNAME Get\_Col\_Arry ( [[]] #col -> [] )

:: ( [[]] #col )

OVER ( [[]] #col [[]] )

MDIMSDROP ( [[]] #col #Nfil #Ncol )

2DUP ( [[]] #col #Nfil #Ncol #Nfil #Ncol )

#\* ( [[]] #col #Nfil #Ncol #Nelem )

#1+ ( [[]] #col #Nfil #Ncol #Nelem+1 )

4ROLL ( [[]] #Nfil #Ncol #Nelem+1 #col )

DO

( ... [[]] #Nfil #Ncol )

INDEX@ ( ... [[]] #Nfil #Ncol #i )

4PICK ( ... [[]] #Nfil #Ncol #i [[]] )

GETATELN ( ... [[]] #Nfil #Ncol ob T )

DROP ( ... [[]] #Nfil #Ncol ob )

4UNROLL ( ... ob [[]] #Nfil #Ncol )

DUP

+LOOP

( ob1...obn [[]] #Nfil #Ncol )

DROPSWAPDROP ( ob1...obn #Nfil )

ONE{}N ( ob1...obn {#Nfil} )

ob>ARRY ( [ob1...obn] )

;

Ejemplo 21 Arreglos

**Obtener una columna de un arreglo de dos dimensiones**

**Caso 2: Arreglo numérico**

Este NULLNAME es mucho más rápido que el anterior, pero sólo funciona para arreglos reales o complejos.

\* Nivel 2: Arreglo de dos dimensiones, debe ser real o complejo.

\* Nivel 1: Bint que representa la columna que queremos obtener.

\* Salida: La columna deseada como un arreglo de una dimensión.

NULLNAME Get\_Col\_NumArry ( [[]] #col -> [] )

:: ( [[]] #col )

SWAP ( #col [[]] )

CKREF ( #col [[]] )

!MATTRNnc ( #col [[]] )

SWAPOVER ( [[]] #fila [[]] )

DIMLIMITS\_ ( [[]] #fila {#F #C} )

TWONTHCOMPDROP\_ ( [[]] #fila #C )

ONE{}N ( [[]] #fila {#C} )

3PICK ( [[]] #fila {#C} [[]] )

laMGET0 ( [[]] #fila {#C} %0/C%0 )

MAKEARRY\_ ( [[]] #fila [] )

SWAPFALSE\_ ( [[]] [] #fila F )

FLASHPTR laGPROW ( [[]] []' )

SWAPDROP ( []' )

;

Capítulo 11  
Objetos Compuestos

Los objetos compuestos tienen otros objetos dentro de ellos. A diferencia de los arreglos, diferentes tipos de objetos pueden ser parte de los objetos compuestos. Ya hemos visto un objeto compuesto en la introducción, cuando usamos un objeto programa para agrupar varios objetos dentro de un objeto único.

Todos los objetos compuestos son similares en su estructura: empiezan con una palabra la cual depende del tipo de objeto compuesto y terminan con la palabra SEMI.

Además de los objetos programa, otros objetos compuestos son las listas, los objetos simbólicos (descritos en el capítulo 14), los objetos unidad (descritos en el capítulo 13) y las matrices (descritas en el capítulo 46).

Puedes crear una lista empezando con {, y terminando con }. Dentro de esta puedes poner tantos objetos como desees, de cualquier tipo.

Objetos programa son delimitados por :: y ;.

Objetos unidad son delimitados por UNIT y ;.

Objetos simbólicos son delimitados por SYMBOL y ;.

Para concatenar dos objetos compuestos, pon estos en la pila y usa &COMP. Para agregar un objeto al comienzo o al final de un compuesto, primero pon el compuesto en la pila, luego el objeto, y usa >HCOMP o >TCOMP, respectivamente.

Para conseguir la longitud de un compuesto (el número de objetos, como un bint), sólo pon el compuesto en el nivel uno de la pila y usa el comando LENCOMP.

Para descomponer un compuesto obteniendo en la pila todos sus objetos y el número de elementos (como con el comando **OBJ→** de User RPL) usa INNERCOMP. La única diferencia es que el número de objetos es retornada como un bint. Para conseguir algún objeto de un compuesto, pon el compuesto en el nivel 2, la posición del objeto en el nivel1 (como un bint, por supuesto), y ejecuta NTHELCOMP. Si el número estuvo fuera de rango, obtendrás FALSE, de otra manera obtendrás el objeto deseado y TRUE. NTHCOMPDROP es la entrada de arriba, seguida por DROP.

Y para conseguir parte de un compuesto, usar SUBCOMP. Poner en el nivel 3 el compuesto, en el nivel 2 la posición inicial y en el nivel 1 la posición final, ambos como bints (desde ahora todos los argumentos numéricos serán bints, a menos que se indique lo contrario). SUBCOMP verifica si los números no están fuera de rango,. Si lo están un compuesto nulo (vacío) es retornado.

Otros comandos son listados en la sección de referencia de abajo.

La estructura de un objeto programa es la siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| Prólogo | DOCOL #2D9D | 5 | “D9D20” |
| Cuerpo | Objetos (con sus prólogos) |  |  |
| SEMI | 5 | “B2130” |

La estructura de una lista es la siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| Prólogo | DOLIST #2A74 | 5 | “47A20” |
| Cuerpo | Objetos (con sus prólogos) |  |  |
| SEMI | 5 | “B2130” |

La estructura de un objeto simbólico es la siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| Prólogo | DOSYMB #2AB8 | 5 | “8BA20” |
| Cuerpo | Objetos (con sus prólogos) |  |  |
| SEMI | 5 | “B2130” |

La estructura de un objeto unidad es la siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| Prólogo | DOEXT #2ADA | 5 | “ADA20” |
| Cuerpo | Objetos (con sus prólogos) |  |  |
| SEMI | 5 | “B2130” |

La estructura de una matriz simbólica es la siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| Prólogo | DOMATRIX #2686 | 5 | “68620” |
| Cuerpo | Objetos (con sus prólogos) |  |  |
| SEMI | 5 | “B2130” |

11.1 Referencia

11.1.1 Operaciones Generales

**Direcc. Nombre Descripción**

0521F &COMP ( comp comp' 🡺 comp'' )

Concatena dos compuestos.

052FA >TCOMP ( comp ob 🡺 comp+ob )

Agrega objeto al final (tail) de un compuesto.

052C6 >HCOMP ( comp ob 🡺 ob+comp )

Agrega objeto al comienzo (head) de un compuesto.

39C8B SWAP>HCOMP\_ ( ob comp 🡺 ob+comp )

Hace SWAP luego >HCOMP.

05089 CARCOMP ( comp 🡺 ob\_inicial )

( comp\_nulo 🡺 comp\_nulo )

Retorna el primer objeto de un compuesto, o un compuesto nulo si el argumento es un compuesto nulo.

361C6 ?CARCOMP ( comp T 🡺 ob )

( comp F 🡺 comp )

Si el flag es TRUE, hace CARCOMP.

05153 CDRCOMP ( comp 🡺 comp-ob\_inicial )

( comp\_nulo 🡺 comp\_nulo )

Retorna el compuesto menos su primer objeto, o un compuesto nulo si el argumento es un compuesto nulo.

2825E (TWONTHCOMPDROP) ( comp 🡺 ob2 )

Retorna el segundo elemento de un compuesto.

2BC006 ˆLASTCOMP ( comp 🡺 ob )

Retorna el último elemento de un compuesto.

Hace DUPLENCOMP luego NTHCOMPDROP.

0567B LENCOMP ( comp 🡺 #n )

Retorna la longitud del compuesto (número de objetos).

3627A DUPLENCOMP ( comp 🡺 comp #n )

Hace DUP luego LENCOMP.

055B7 NULLCOMP? ( comp 🡺 flag )

Si el compuesto está vacío, retorna TRUE.

36266 DUPNULLCOMP? ( comp 🡺 comp flag )

Hace DUP luego NULLCOMP?.

056B6 NTHELCOMP ( comp #i 🡺 ob T )

( comp #i 🡺 F )

Retorna elemento especificado del compuesto y TRUE, o sólo FALSE si este no pudo ser hallado.

35BC3 NTHCOMPDROP ( comp #i 🡺 ob )

Hace NTHELCOMP luego DROP.

35D58 NTHCOMDDUP ( comp #i 🡺 ob ob )

Hace NTHCOMPDROP luego DUP.

376EE POSCOMP ( comp ob test\_de2argumentos 🡺 #i )

( comp ob test\_de2argumentos 🡺 #0 )

(ejemplo de test: ' %<)

Evalúa el test para todos los elementos del compuesto y ob, y retorna el índice del primer objeto para el cual el test es TRUE. Si para ninguno es TRUE, returna #0. Por ejemplo, el programa de abajo retorna #4:

:: { %1 %2 %3 %-4 %-5 %6 %7 } %0 ' %< POSCOMP ;

**Direcc. Nombre Descripción**

3776B EQUALPOSCOMP ( comp ob 🡺 #pos )

( comp ob 🡺 #0 )

POSCOMP con EQUAL como test.

37784 NTHOF ( ob comp 🡺 #i )

( ob comp 🡺 #0 )

Hace SWAP luego EQUALPOSCOMP.

0FD006 ˆListPos ( ob {} 🡺 #i/#0 )

Equivale a NTHOF, pero más rápido. Sólo funciona para listas.

37752 #=POSCOMP ( comp # 🡺 #i )

( comp # 🡺 #0 )

POSCOMP con #= como test.

05821 SUBCOMP ( comp #m #n 🡺 comp' )

Retorna una parte del compuesto (subcompuesto). Hace verificaciones de índices.

376B7 matchob? ( ob comp 🡺 T )

( ob comp 🡺 ob F )

Retorna TRUE si ob es igual (EQUAL) a algún elemento del compuesto.

371B3 Embedded? ( ob1 ob2 🡺 flag )

Retorna TRUE si ob2 está dentro de ob1 o es el mismo ob1. De lo contario retorna FALSE. (La comparación es con EQ)

Ejemplos:

:: ' :: # 8 # 9 ; # 8 Embedded? ;

retorna FALSE.

:: ' :: # 8 # 9 ; DUP CARCOMP Embedded? ;

retorna TRUE.

:: ' :: # 8 # 9 ; DUP CARCOMP TOTEMPOB Embedded? ;

retorna FALSE.

37798 Find1stTrue ( comp test\_de1argumento 🡺 ob T )

( comp test\_de1argumento 🡺 F )

Evalúa el test para cada elemento del compuesto. El primer elemento para el que retorna TRUE es puesto en la pila seguido de TRUE. Si para ningún objeto retorna TRUE, FALSE es puesto en la pila. Por ejemplo el programa de abajo retorna %-4 y TRUE.

:: { %1 %2 %2 %-4 %-5 %6 } ' %0< Find1stTrue ;

377C5 Lookup ( ob test\_de2argumentos comp 🡺 ob\_siguiente T )

( ob test\_de2argumentos comp 🡺 ob F )

Evalúa el test para cada elemento de lugar impar (1,3,...) en el compuesto y ob (en ese orden). Si el test retorna TRUE para alguna de las evaluaciones, el objeto del compuesto después del evaluado (lugar par) es retornado seguido de TRUE. Si para ningún objeto el test retorna TRUE, FALSE es retornado.

El número de elementos del compuesto debe ser par.

Por ejemplo, el programa de abajo retorna %6 y TRUE.

:: %0 ' %< { %1 %2 %3 %-4 %-5 %6 } Lookup ;

**Direcc. Nombre Descripción**

377DE Lookup.1 ( ob test\_de2argumentos 🡺 ob\_siguiente T )

( ob test\_de2argumentos 🡺 ob F )

Pila de retornos:

( comp 🡺 )

Lookup con el compuesto ya puesto sobre la pila de retornos (por ejemplo, con >R). Llamado por Lookup.

37829 EQLookup ( ob comp 🡺 ob\_siguiente T )

( ob comp 🡺 ob F )

Lookup con EQ como test.

37B54 NEXTCOMPOB ( comp #ofs 🡺 comp #ofs' ob T )

( comp #ofs 🡺 comp F )

Retorna el objeto que comienza en la posición #ofs del compuesto (posición en nibbles).

Para conseguir el primer objeto, #ofs debe ser #5 (para pasarse el prólogo del compuesto). También funciona #0

Si el objeto de la posición #ofs es SEMI (al final de un compuesto), retorna FALSE.

11.1.2 Formar un compuesto

Hay también comandos para construir listas y objetos programa, con el número de elementos especificado, descritos en esta sección.

**Direcc. Nombre Descripción**

05459 {}N ( obn..ob1 #n 🡺 { obn..ob1 } )

Crea una lista.

05445 ::N ( ob1..obn #n 🡺 :: ob1..obn ; )

Crea un objeto programa.

0546D SYMBN ( ob1..obn #n 🡺 symb )

Crea un objeto simbólico.

05481 EXTN ( ob1..obn #n 🡺 unit )

Crea un objeto unidad.

293F8 P{}N ( ob1..obn #n 🡺 {} )

Crea una lista con posible recolección de basura.

05331 COMPN\_ ( ob1..obn #n #prólogo 🡺 comp )

Crea un objeto compuesto. Por ejemplo:

:: 1. 2. 3. BINT3 # 2686 COMPN ;

Crea una matriz simbólica de una dimensión.

11.1.3 Descomponer un compuesto

**Direcc. Nombre Descripción**

054AF INNERCOMP ( comp 🡺 obn..ob1 #n )

3622A DUPINCOMP ( comp 🡺 comp obn..ob1 #n )

3623E SWAPINCOMP ( comp obj 🡺 obj obn..ob1 #n )

35BAF INCOMPDROP ( comp 🡺 obn..ob1 )

35C68 INNERDUP ( comp 🡺 obn..ob1 #n #n )

2F0EC ICMPDRPRTDRP ( comp 🡺 obn...ob4 ob2 ob1 )

Hace INCOMPDROP luego ROTDROP.

3BADA XEQLIST>\_ ( comp 🡺 obn..ob1 %n )

Hace INNERCOMP luego UNCOERCE.

366E9 INNER#1= ( comp 🡺 obn..ob1 flag )

**Direcc. Nombre Descripción**

157006 ˆSYMBINCOMP ( symb 🡺 ob1 .. obN #n )

( ob 🡺 ob #1 )

( {} 🡺 {} #1 )

Descompone un objeto simbólico en un objeto meta. Otros objetos son convertidos en metas de 1 objeto poniendo #1 en la pila.

12A006 ˆ2SYMBINCOMP ( ob1 ob2 🡺 meta1 meta2 )

Hace ˆSYMBINCOMP para 2 objetos.

158006 ˆCKINNERCOMP ( {} 🡺 ob1 .. obN #n )

( ob 🡺 ob #1 )

Descompone una lista en un objeto meta. Otros objetos son convertidos en metas de 1 objeto poniendo #1 en la pila.

11.1.4 Listas

**Direcc. Nombre Descripción**

055E9 NULL{} ( 🡺 {} )

Pone una lista vacía en la pila.

36ABD DUPNULL{}? ( {} 🡺 {} flag )

159006 ˆDUPCKLEN{} ( {} 🡺 {} #n )

( ob 🡺 ob #1 )

Retorna la longitud de la lista o 1 para otro tipo de objetos.

29D18 ONE{}N ( ob 🡺 { ob } )

36202 TWO{}N ( ob1 ob2 🡺 { ob1 ob2 } )

36216 THREE{}N ( ob1 ob2 ob3 🡺 { ob1 ob2 ob3 } )

361EE #1-{}N ( ob1..obn #n+1 🡺 {} )

2B42A PUTLIST ( ob #i {} 🡺 {}' )

Reemplaza objeto en la posición indicada. Asume un #i válido.

2FC006 ˆINSERT{}N ( {} ob # 🡺 {}' )

Inserta objeto en la lista de tal manera que tendrá la posición # en la lista final.

# debe ser menor o igual que la longitud de lista final.

Si # es cero, >TCOMP es usado.

2FB006 ˆNEXTPext ( lista 🡺 lista1 lista2 )

Extrae en lista2 todas las ocurrencias del primer objeto de lista, los objetos restantes van a la lista1.

lista1 = lista-lista2.

2FD006 ˆCOMPRIMext ( {} 🡺 {}' )

Suprime múltiples ocurrencias en una lista.

15A006 ˆCKCARCOMP ( {} 🡺 ob1 )

( ob 🡺 ob )

Retorna el primer elemento para listas, o el objeto mismo.

2EF5A apndvarlst ( {} ob 🡺 {}' )

Agrega el objeto al final de la lista, si ob aún no está en la lista.

0FE006 ˆAppendList ( {} ob 🡺 {}' )

Equivale a apndvarlst, pero más rápido.

**Direcc. Nombre Descripción**

4EB006 ˆprepvarlist ( {} ob 🡺 {}' )

Agrega el objeto al inicio de la lista si ob aún no está en la lista.

Si ob está en la lista, lo mueve al inicio de la lista.

100006 ˆSortList ( L test\_de2argumentos 🡺 L' )

Ordena la lista según el test.

En la lista final L', el test retornaría FALSE al evaluar cada par de elementos consecutivos.

Para ordenar reales ascendentemente el test puede ser el siguiente: ' %>.

28A006 ˆPIext ( {} 🡺 ob )

Retorna el producto de todos los elementos de una lista.

25ED3 EqList? ( {} 🡺 flag )

( ob 🡺 F )

Si la lista tiene menos de dos elementos, retorna TRUE.

Si el segundo elemento de la lista no es lista, retorna TRUE.

En otro caso, retorna FALSE.

11.1.5 Objetos Programa

**Direcc. Nombre Descripción**

055FD NULL:: ( 🡺 :: ; )

Retorna un programa vacío.

37073 Ob>Seco ( ob 🡺 :: ob ; )

Hace ONE luego ::N.

3705A ?Ob>Seco ( ob 🡺 :: ob ; )

Si el objeto no es un programa, hace Ob>Seco.

37087 2Ob>Seco ( ob1 ob2 🡺 :: ob1 ob2 ; )

Hace TWO luego ::N.

3631A ::NEVAL ( ob1..obn #n 🡺 ? )

Hace ::N luego EVAL.

11.2 Ejemplos

Ejemplo 1 Compuestos

**Aplicando un programa o comando a cada uno de los elementos de un compuesto**

¿Cómo aplicar un programa a cada elemento de un compuesto?

En User RPL, puedes aplicar un programa a cada elemento de una lista con el comando **MAP** o con **DOSUBS**. Veamos tres ejemplos en System RPL:

\* Este NULLNAME eleva cada número real de la lista al cuadrado

\* Argumento: una lista no vacía con números reales

NULLNAME 1{%}\_Cuadrado ( {%} -> {%} )

:: ( {} )

INNERDUP ( ob1...obn #n #n )

ZERO\_DO (DO) ( ... #n )

ROLL ( ... obi )

%SQ\_ ( ... obi' )

ISTOP@ ( ... obi' #n )

LOOP

( ob1'...obn' #n )

{}N ( {} )

;

\* Este NULLNAME eleva cada número real de la lista al cubo

\* Argumento: una lista no vacía con números reales

NULLNAME 1{%}\_Cubo ( {%} -> {%} )

:: ( {} )

INNERDUP ( ob1...obn #n #n )

ZERO\_DO (DO) ( ... #n )

ROLL ( ... obi )

%3 %^ ( ... obi' )

ISTOP@ ( ... obi' #n )

LOOP

( ob1'...obn' #n )

{}N ( {} )

;

\* Este NULLNAME convierte cada bint de la lista a hxs de long 16 nibbles

\* Argumento: una lista no vacía con bints

NULLNAME 1{#}\_#>HXS16 ( {#} -> {HXS} )

:: ( {} )

INNERDUP ( ob1...obn #n #n )

ZERO\_DO (DO) ( ... #n )

ROLL ( ... obi )

#>HXS BINT11 EXPAND ( ... obi' )

ISTOP@ ( ... obi' #n )

LOOP

( ob1'...obn' #n )

{}N ( {} )

;

Ejemplo 2 Compuestos

**Aplicando un programa o comando a cada uno de los elementos de un compuesto. Caso General.**

El siguiente NULLNAME aplica un programa o comando (que tome un argumento y devuelva sólo un objeto) a cada elemento de un compuesto no vacío.

De esta manera, los tres ejemplos anteriores serían equivalentes a usar:

' %SQ\_ Prog1a1Comp

' :: %3 %^ ; Prog1a1Comp

' :: #>HXS BINT11 EXPAND ; Prog1a1Comp

\* Evalúa un programa o comando a los elementos de 1 compuesto

\* Entrada:

\* Nivel 2: Objeto compuesto no vacío

\* Nivel 1: Programa o comando que tome 1 argumento y devuelva un objeto

\* Salida:

\* Un compuesto con el mismo número de elementos que el original

\*

\* Para llamar a la posición del elemento usar INDEX@ #1+

\* Para llamar al tamaño del compuesto usa ISTOP@

NULLNAME Prog1a1Comp ( comp prog1a1 -> comp' )

:: ( comp prog1a1 )

OVER ( comp prog1a1 comp )

TYPE ( comp prog1a1 #tipo )

FLASHPTR 2LAMBIND ( comp )

INNERDUP ( ob1...obn #n #n )

ZERO\_DO (DO)

( ... #n )

ROLL ( ... obi )

2GETEVAL ( ... obi' )

ISTOP@ ( ... obi' #n )

LOOP

( ob1'...obn' #n )

1GETABND ( ob1'...obn' #n #tipo )

COMPN\_ ( comp' )

;

Por ejemplo:

{ 11. 12. 13. }

' %SQ\_

Prog1a1Comp

Devuelve:

{ 121. 144. 169. }

Otro ejemplo:

{ "TEXTO A" "TEXTO B" "TEXTO C" }

' :: INDEX@ #1+ #>$ tok/\_ &$ ISTOP@ #>$ &$ ") " &$ SWAP DO>STR &$ ;

Prog1a1Comp

Devuelve:

{ "1/3) TEXTO A" "2/3) TEXTO B" "3/3) TEXTO C" }

Ejemplo 3 Compuestos

**Aplicando un programa o comando a los elementos de dos compuestos**

Para evaluar un programa o comando a los elemento de dos compuestos, puedes usar el siguiente NULLNAME

\* Evalúa un programa o comando a los elementos de 2 compuestos

\* Entrada:

\* Niveles 3,2: Objetos compuestos no vacíos

\* Nivel 1: Programa o comando que tome 2 argumentos y devuelva un objeto

\* Salida:

\* Un compuesto con el mismo número de elementos que el original

\*

\* Para llamar a la posición del elemento usar INDEX@ #1+

\* Para llamar al tamaño del compuesto usa ISTOP@

NULLNAME Prog2a1Comp ( comp1 comp2 prog2a1 -> comp' )

:: ( comp1 comp2 prog2a1 )

OVER ( comp1 comp2 prog2a1 comp2 )

TYPE ( comp1 comp2 prog2a1 #tipo )

FLASHPTR 2LAMBIND ( comp1 comp2 )

>R ( comp1 )

INNERDUP ( ob11...ob1n #n #n )

ZERO\_DO (DO)

( ... #n )

ROLL ( ... ob1i )

RSWAP

'R ( ... ob1i ob2i )

RSWAP

2GETEVAL ( ... obi' )

ISTOP@ ( ... obi' #n )

LOOP

( ob1'...obn' #n )

1GETABND ( ob1'...obn' #n #tipo )

COMPN\_ ( comp' )

;

Por ejemplo:

{ 11. 12. 13. }

{ 100. 200. 300. }

' %+

Prog2a1Comp

Devuelve:

{ 111. 212. 313. }

Ejemplo 4 Compuestos

**Aplicando un programa o comando a los elementos de varios compuestos**

Para evaluar un programa o comando a los elemento de varios compuestos, puedes usar el siguiente NULLNAME

\* Evalúa un programa o comando a los elementos de varios compuestos

\* Entrada:

\* Niveles N+2,...,5,4,3: Compuestos no vacíos, todas del mismo tamaño

\* Nivel 2: Un bint (#N) Indica el número de compuestos

\* Nivel 1: Programa o comando que tome #N argumentos y devuelva un objeto

\* Salida: Un compuesto con el mismo número de elementos que los originales

\* Para llamar a la posición del elemento usar INDEX@ #1+

\* Para llamar al tamaño de los compuestos usa ISTOP@

NULLNAME ProgNa1Comp ( comp1...compN #N progNa1 -> comp' )

:: ( comp1...compN #N progNa1 )

3PICK ( comp1...compN #N progNa1 compN )

LENCOMP ( comp1...compN #N progNa1 #el )

SWAP4PICK ( comp1...compN #N #el progNa1 compN )

TYPE ( comp1...compN #N #el progNa1 #tipo )

' NULLLAM BINT4 NDUPN DOBIND

( comp1...compN )

\* 4LAM: #N 3LAM: #elem 2LAM: PROG 1LAM: #tipo

4GETLAM ( comp1...compN #N )

ZERO\_DO (DO)

( ... )

ISTOP-INDEX ( ... #N-i )

ROLL ( ... compi )

>R ( ... )

RSWAP ( ... )

LOOP

( )

3GETLAM ( #el )

ZERO\_DO (DO)

( ... )

4GETLAM ( ... #N )

ZERO\_DO (DO)

( ... )

4GETLAM ( ... #N )

#2+ ( ... #N+2 )

RROLL\_ ( ... )

'R ( ... obij )

RSWAP ( ... obij )

LOOP

( ... ob1j...obNj )

4GETLAM ( ... ob1j...obNj #N )

#1+ ( ... ob1j...obNj #N+1 )

RROLL\_ ( ... ob1j...obNj )

2GETEVAL ( ... obj' )

LOOP

( ob1'...obN' )

3GETLAM ( ob1'...obN' #N )

1GETABND ( ob1'...obN' #N #tipo )

COMPN\_ ( comp' )

;

Por ejemplo:

{ 11. 12. 13. }

{ 100. 200. 300. }

BINT2

' %+

Prog2a1Comp

Devuelve:

{ 111. 212. 313. }

Ejemplo 5 Compuestos

**Aplicando un programa o comando a grupos de dos elementos dentro de un compuesto**

Para evaluar un programa o comando a grupos de dos elementos dentro de un compuesto, puedes usar el siguiente NULLNAME

\* Evalúa un programa o comando a grupos de 2 elemento en un compuesto

\* Entrada:

\* Nivel 2: Un compuesto. Su número mínimo de elementos debe ser 2.

\* Nivel 1: Programa o comando que tome 2 argumentos y devuelva 1 objeto.

\* Salida:

\* Un compuesto cuyo número de elementos será 1 menos que en el compuesto original

\*

\* Para llamar al nº del procedim. usar INDEX@ #1+ (como NSUB en User RPL)

\* Para llamar al nº total de procedim. usa ISTOP@ (como ENDSUB en User RPL)

NULLNAME DoSubs2a1Comp ( comp prog2a1 -> comp' )

:: ( comp prog2a1 )

OVER ( comp prog2a1 comp )

LENCOMP ( comp prog2a1 #n )

#1-SWAP ( comp #n-1 prog2a1 )

ROTDUP ( #n-1 prog2a1 comp comp )

>R ( #n-1 prog2a1 comp )

TYPE ( #n-1 prog2a1 #tipo )

FLASHPTR 3LAMBIND ( )

3GETLAM ( #n-1 )

ZERO\_DO (DO)

RSWAP ( ... )

RDUP ( ... )

'R'R ( ... ob1 ob2 )

RDROP ( ... ob1 ob2 )

'R ( ... ob1 ob2 ob1 )

DROP ( ... ob1 ob2 )

RSWAP ( ... ob1 ob2 )

2GETEVAL ( ... ob1' )

LOOP

( ob1',ob2'...obn-1' )

3GETLAM ( ob1',ob2'...obn-1' #n-1 )

1GETABND ( ob1',ob2'...obn-1' #n-1 #tipo )

COMPN\_ ( comp' )

RDROP ( comp' )

;

Por ejemplo:

{ 11. 13. 23. 26. }

' :: SWAP %- ;

DoSubs2a1Comp

Devuelve:

{ 2. 10. 3. }

Ejemplo 6 Compuestos

**Aplicando un programa o comando a grupos de varios elementos dentro de un compuesto**

Para evaluar un programa o comando a grupos de ‘S’ elementos dentro de un compuesto, puedes usar el siguiente NULLNAME

\* Evalúa un programa o comando a grupos de 's' elemento en un compuesto

\* Entrada:

\* Nivel 3: Un compuesto no vacío. Su tamaño (n) debe ser mayor o igual a 's'

\* Nivel 2: Un bint (#s) Indica el tamaño de los grupos dentro del compuesto

\* Nivel 1: Programa o comando que tome 's' argumentos y devuelva 1 objeto

\* Salida:

\* Una lista cuyo número de elementos será 'n-s+1'

\*

\* Para llamar al nº del procedim. usar INDEX@ #1+ (como NSUB en User RPL)

\* Para llamar al nº total de procedim. usa ISTOP@ (como ENDSUB en User RPL)

NULLNAME DoSubsSa1Comp ( comp #s progSa1 -> comp' )

:: ( comp #s progSa1 )

3PICK ( comp #s progSa1 {} )

LENCOMP ( comp #s progSa1 #n )

3PICK ( comp #s progSa1 #n #s )

#-+1 ( comp #s progSa1 #n-s+1 )

SWAP4PICK ( comp #s #n-s+1 progSa1 comp )

TYPE ( comp #s #n-s+1 progSa1 #tipo )

' NULLLAM BINT4 NDUPN DOBIND

( comp )

>R ( )

3GETLAM ( #n-s+1 )

ZERO\_DO (DO)

RSWAP ( ... )

RDUP ( ... )

4GETLAM ( ... #s )

ZERO\_DO (DO)

RSWAP 'R RSWAP

LOOP ( ... ob1...obs )

RDROP ( ... ob1...obs )

'R ( ... ob1...obs ob1 )

DROP ( ... ob1...obs )

RSWAP ( ... ob1...obs )

2GETEVAL ( ... ob1' )

LOOP

( ob1',ob2'...ob[n-s+1]' )

3GETLAM ( ob1',ob2'...ob[n-s+1] #n-s+1 )

1GETABND ( ob1',ob2'...ob[n-s+1] #n-s+1 #tipo )

COMPN\_ ( comp' )

RDROP ( comp' )

;

Por ejemplo:

{ 11. 13. 23. 26. }

BINT2

' :: SWAP %- ;

DoSubsSa1Comp

Devuelve:

{ 2. 10. 3. }

Ejemplo 7 Compuestos

**Aplicar un programa o comando a todos los elementos de un compuesto hasta agotar el compuesto y retornar un solo objeto como resultado**

En User RPL se podia usar el comando **STREAM** para listas.

Veamos algunos ejemplos en System RPL:

\* Halla la suma de todos los elementos de un compuesto.

\* El compuesto debe ser no vacío y debe contener números reales.

NULLNAME StreamComp\_%+ ( comp -> %suma )

:: ( comp )

INNERCOMP ( %...% #n )

DUP#1= ( %...% #n flag )

caseDROP

( %...% #n )

ONE\_DO (DO)

%+

LOOP

( %' )

;

\* Halla el producto de todos los elementos de un compuesto.

\* El compuesto debe ser no vacío y debe contener números reales.

NULLNAME StreamComp\_%\* ( comp -> %producto )

:: ( comp )

INNERCOMP ( %...% #n )

DUP#1= ( %...% #n flag )

caseDROP

( %...% #n )

ONE\_DO (DO)

%\*

LOOP

( %' )

;

\* Halla el mayor valor de todos los elementos de un compuesto.

\* El compuesto debe ser no vacío y debe contener números reales.

NULLNAME StreamComp\_%MAX ( comp -> %maximo )

:: ( comp )

INNERCOMP ( %...% #n )

DUP#1= ( %...% #n flag )

caseDROP

( %...% #n )

ONE\_DO (DO)

%MAX

LOOP

( %' )

;

\* Halla el menor valor de todos los elementos de un compuesto.

\* El compuesto debe ser no vacío y debe contener números reales.

NULLNAME StreamComp\_%MIN ( comp -> %minimo )

:: ( comp )

INNERCOMP ( %...% #n )

DUP#1= ( %...% #n flag )

caseDROP

( %...% #n )

ONE\_DO (DO)

%MIN

LOOP

( %' )

;

Ejemplo 8 Compuestos

**Aplicar un programa o comando a todos los elementos de un compuesto hasta agotar el compuesto y retornar un solo objeto como resultado. Caso general.**

Dos procedimientos más generales son los siguientes NULLNAME:

\* Ejecuta un programa o comando de la forma ( ob ob' -> ob'' ) repetidamente

\* en los dos primeros elementos en una lista hasta que ésta queda agotada.

\* Indica el resultado final.

NULLNAME StreamComp ( comp prog2a1 -> ob )

:: ( comp prog2a1 )

OVER ( comp prog2a1 comp )

LENCOMP ( comp prog2a1 #n )

DUP#1= ( comp prog2a1 #n flag )

case2drop

CARCOMP

( comp prog2a1 #n )

FLASHPTR 2LAMBIND

( comp )

>R ( )

'R ( ob1 )

1GETLAM ( ob1 #n )

ONE\_DO (DO)

( ob )

RSWAP

'R ( ob obi+1 )

RSWAP

2GETEVAL ( ob' )

LOOP

( ob' )

ABND ( ob' )

;

\* Ejecuta un programa o comando de la forma ( ob ob' -> ob'' ) repetidamente

\* en los dos últimos elementos en una lista hasta que ésta queda agotada.

\* Indica el resultado final.

NULLNAME StreamRevComp ( comp prog2a1 -> ob )

:: ( comp prog2a1 )

1LAMBIND ( comp )

:: INNERCOMP ( ob1...obn #n )

DUP#1= ( ob1...obn #n flag )

caseDROP

( ob1...obn #n )

ONE\_DO (DO)

( ob1...ob' ob'' )

1GETLAM ( ob1...ob' ob'' prog2a1 )

EVAL ( ob1...ob''' )

LOOP

( ob )

;

ABND ( ob )

;

- Los 2 NULLNAME anteriores producen el mismo resultado para operaciones conmutativas como %+, %\*, %MAX y %MIN.

- Para operaciones que no son conmutativas pueden producir resultados diferentes.

- El NULLNAME StreamRevComp es más rápido que el NULLNAME StreamComp.

Los siguientes cuatro programas tienen el mismo efecto que los NULLNAME de la página anterior.

' %+ StreamRevComp

' %\* StreamRevComp

' %MAX StreamRevComp

' %MIN StreamRevComp

Ejemplo 9 Compuestos

**¿Cómo saber si al evaluar un test a cada elemento de un compuesto, el resultado es siempre TRUE?**

Puedes usar el siguiente NULLNAME

\* Evalúa el TEST para los elementos de un compuesto no vacío.

\* Si para todos los elementos es TRUE, devuelve TRUE.

\* Si para algún elemento es FALSE, devuelve FALSE.

NULLNAME TodosTrue?Comp ( comp Test1Arg -> flag )

:: ( comp Test1Arg )

SWAPDUP ( Test1Arg comp comp )

>R ( Test1Arg comp )

LENCOMP ( Test1Arg #n )

TRUESWAP\_ ( Test1Arg T #n )

ZERO\_DO (DO)

( Test1Arg T )

RSWAP

'R ( Test1Arg T obi )

RSWAP

3PICK ( Test1Arg T obi Test1Arg )

EVAL ( Test1Arg T flag )

?SKIP

:: ExitAtLOOP DROPFALSE ;

( Test1Arg flag' )

LOOP

( Test1Arg flag )

RDROP ( Test1Arg flag )

SWAPDROP ( flag )

;

A continuación algunos ejemplos que muestran el uso de ese NULLNAME:

\* Retorna TRUE si todos los elementos del compuesto son reales

NULLNAME TodosTrue?Comp\_%? ( comp -> flag )

::

' TYPEREAL?

TodosTrue?Comp

;

\* Retorna TRUE si todos los elementos del compuesto son nombres globales

NULLNAME TodosTrue?Comp\_id? ( comp -> flag )

::

' TYPEIDNT?

TodosTrue?Comp

;

\* Retorna TRUE si todos los elementos del compuesto son reales o enteros

NULLNAME TodosTrue?Comp\_%?\_or\_Z? ( comp -> flag )

::

' :: ( ob )

DUPTYPEREAL? ( ob flag )

SWAP ( flag ob )

TYPEZINT? ( flag flag' )

OR ( flag'' )

;

TodosTrue?Comp

;

Ejemplo 10 Compuestos

**¿Cómo saber si al evaluar un test a cada elemento de un compuesto, al menos una vez resulta TRUE?**

Puedes usar el siguiente NULLNAME

\* Evalúa el TEST para los elementos de un compuesto no vacío.

\* Si para alguno de los elementos es TRUE, devuelve TRUE.

\* Si para todos los elementos es FALSE, devuelve FALSE.

NULLNAME AlgunoTrue?Comp ( comp Test1Arg -> flag )

:: ( comp Test1Arg )

Find1stTrue ( ob T // F )

DUP ( ob T T // F F )

NOT?SEMI

( ob T )

SWAPDROP ( T )

;

Ejemplo 11 Compuestos

**Reemplazar un objeto dentro de un compuesto**

Puedes usar el siguiente NULLNAME

\* Reemplaza un objeto en un compuesto:

\* Entrada:

\* NIVEL 3: Objeto compuesto

\* NIVEL 2: Posición de objeto que será reemplazado

\* NIVEL 1: Objeto que entrará en el compuesto.

\* Salida:

\* NIVEL 1: Objeto compuesto modificado

NULLNAME ReplaceInComp ( comp #i ob -> comp' )

:: ( comp #i ob )

3PICK ( comp #i ob comp )

TYPE ( comp #i ob #tipo )

FLASHPTR 3LAMBIND

( )

INNERDUP ( ob1... obn #n #n )

3GETLAM ( ob1... obn #n #n #i )

#- ( ob1... obn #n #n-i )

#2+ROLL ( ob1... obn #n obi )

DROP ( ob1... obn #n )

2GETLAM ( ob1... obn #n ob )

OVER ( ob1... obn #n ob #n )

3GETLAM ( ob1... obn #n ob #n #i )

#- ( ob1... obn #n ob #n-i )

#2+UNROLL ( ob1... obn #n )

1GETABND ( ob1... obn #n #tipo )

COMPN\_ ( comp' )

;

Recuerda que en el caso de una lista se puede usar el comando PUTLIST.

**Direcc. Nombre Descripción**

2B42A PUTLIST ( ob #i {} 🡺 {}' )

Reemplaza objeto en la posición indicada. Asume un #i válido.

Ejemplo 12 Compuestos

**Insertar un objeto dentro de un compuesto**

Puedes usar el siguiente NULLNAME

\* Inserta un objeto dentro de un compuesto:

\* Entrada:

\* NIVEL 3: Objeto compuesto

\* NIVEL 2: Posición donde entrará el objeto.

\* NIVEL 1: Objeto que entrará en el compuesto.

\* Salida:

\* NIVEL 1: Objeto compuesto modificado

NULLNAME InsertInComp ( comp #i ob -> comp' )

:: ( comp #i ob )

\*{ 11. 12. 13. 14. 15. 16. } 0 777.

3PICK ( comp #i ob comp )

TYPE ( comp #i ob #tipo )

FLASHPTR 3LAMBIND

( )

INNERDUP ( ob1...obn #n #n )

2GETLAMSWAP\_ ( ob1...obn #n ob #n )

3GETLAM ( ob1...obn #n ob #n #i )

#-+1 ( ob1...obn #n ob #n-i+1 )

#2+UNROLL ( ob1...ob...obn #n )

#1+ ( ob1...ob...obn #n+1 )

1GETABND ( ob1...ob...obn #n+1 #tipo )

COMPN\_ ( comp' )

;

Recuerda que en el caso de una lista se puede usar el comando ˆINSERT{}N.

**Direcc. Nombre Descripción**

2FC006 ˆINSERT{}N ( {} ob # 🡺 {}' )

Inserta objeto en la lista de tal manera que tendrá la posición # en la lista final.

# debe ser menor o igual que la longitud de lista final.

Si # es cero, >TCOMP es usado.

Ejemplo 13 Compuestos

**Quitar un objeto de un compuesto**

Puedes usar el siguiente NULLNAME

\* QUITA EL ELEMEMENTO DE ORDEN 'i' EN UN COMPUESTO

\* 'i' debe estar entre 1 y el tamaño del compuesto.

NULLNAME RemoveInComp ( comp #i -> comp' )

:: ( comp #i )

SWAPDUP ( #i comp comp )

TYPE ( #i comp #tipo )

UNROT ( #tipo #i comp )

INNERCOMP ( #tipo #i ob1...obn #n )

get1 ( #tipo ob1...obn #n #i )

OVERSWAP ( #tipo ob1...obn #n #n #i )

#- ( #tipo ob1...obn #n #n-i )

#2+ROLL ( #tipo ob1..obn #n ob )

DROP#1- ( #tipo ob1..obn #n-1 )

get1 ( ob1..obn #n-1 #tipo )

COMPN\_ ( comp' )

;

Ejemplo 14 Compuestos

**Sustituir un objeto por otro dentro de un compuesto.**

Puedes usar el siguiente NULLNAME

\* Reemplaza ob2 por ob1 en cada elemento del compuesto

NULLNAME Subst\_Comp ( comp ob2 ob1 -> symb' )

:: ( comp ob2 ob1 )

3PICK ( comp ob2 ob1 comp )

TYPE ( comp ob2 ob1 #tipo )

FLASHPTR 3LAMBIND

( comp )

INNERDUP ( ob1...obn #n #n )

ZERO\_DO (DO)

( ... #n )

ROLL ( ... obi )

:: DUP

3GETLAM

EQUAL

casedrop

2GETLAM

;

( ... obi )

ISTOP@ ( ... obi #n )

LOOP

( ob1'...obn' #n )

1GETABND ( ob1'...obn' #n #tipo )

COMPN\_ ( comp' )

;

Capítulo 12  
Objetos Meta

Un objeto meta (o sólo meta, para ser breves) es un conjunto de n objetos y su cantidad (como un bint) en la pila. Un objeto meta puede ser considerado como otra representación de un objeto compuesto. El comando INNERCOMP descompone cualquier compuesto, expresandolo como un objeto meta en la pila. La transformación opuesta es hecha por varios comandos diferentes, de acuerdo al tipo de objeto compuesto que se desea formar (explicados en la sección 11.12).

Nota que un bint cero es un objeto meta (meta vacío), el objeto meta nulo.

Existen comandos para hacer operaciones con objetos de la pila, que tratan a objetos meta como si fueran un único objeto. Generalmente, los nombres de estos comandos están en letras minúsculas. Sin embargo, algunos comandos han sido nombrados sin seguir esa regla, debido a que fueron creados con otros propósitos en mente, como por ejemplo manipular objetos de otro tipo.

Existen también objetos meta user, los cuales son como los objetos meta, pero su cantidad se representa como número real y no como bint. Los meta user no son objetos muy comunes.

12.1 Referencia

12.1.1 Funciones de la pila

**Direcc. Nombre Descripción**

0326E NDROP ( meta 🡺 )

Debería llamarse drop.

37032 DROPNDROP ( meta ob 🡺 )

Debería llamarse DROPdrop.

35FB0 #1+NDROP ( ob meta 🡺 )

Debería llamarse dropDROP.

aka: N+1DROP

28211 NDROPFALSE ( meta 🡺 F )

Debería llamarse dropFALSE.

391006 ˆNDROPZERO ( obn..ob1 #n 🡺 #0 )

Reemplaza un objeto meta con un meta vacío.

Debería llamarse dropZERO.

29A5D (dup) ( meta 🡺 meta meta )

29A5D psh ( meta1 meta2 🡺 meta2 meta1 )

Debería llamarse swap.

29A8F roll2ND ( meta1 meta2 meta3 🡺 meta2 meta3 meta1 )

Debería llamarse rot.

29B12 unroll2ND ( meta1 meta2 meta3 🡺 meta3 meta1 meta2 )

Debería llamarse unrot.

3695A SWAPUnNDROP ( meta1 meta2 🡺 meta2 )

Debería llamarse swapdrop.

36FA6 metaROTDUP ( meta1 meta2 meta3 🡺 meta2 meta3 meta1 meta1 )

Debería llamarse rotdup.

12.1.2 Combinando Objetos Meta

**Direcc. Nombre Descripción**

296A7 top& ( meta1 meta2 🡺 meta1&meta2 )

2973B pshtop& ( meta1 meta2 🡺 meta2&meta1 )

36FBA ROTUntop& ( meta1 meta2 meta3 🡺 meta2 meta3&meta1 )

36FCE roll2top& ( meta1 meta2 meta3 🡺 meta3 meta1&meta2 )

aka: rolltwotop&

2963E psh& ( meta1 meta2 meta3 🡺 meta1&meta3 meta2 )

12.1.3 Operaciones con Objetos Meta y Otros Objetos

**Direcc. Nombre Descripción**

3592B SWAP#1+ ( # ob 🡺 ob #+1 )

( meta ob 🡺 meta&ob )

aka: SWP1+

34431 DUP#1+PICK ( obn..ob1 #n 🡺 obn..ob1 #n obn )

34504 get1 ( ob meta 🡺 meta ob )

36147 OVER#2+UNROL ( meta ob 🡺 ob meta )

29693 psh1top& ( meta ob 🡺 ob&meta )

28071 pull ( meta&ob 🡺 meta ob )

aka: #1-SWAP

**Direcc. Nombre Descripción**

28085 pullrev ( ob&meta 🡺 meta ob )

29821 psh1& ( meta1 meta2 ob 🡺 ob&meta1 meta2 )

298C0 psh1&rev ( meta1 meta2 ob 🡺 ob&meta1 meta2 )

2F193 UobROT ( ob meta1 meta2 🡺 meta1 meta2 ob )

29754 pullpsh1& ( meta1 meta2&ob 🡺 ob&meta1 meta2 )

406006 ˆaddt0meta ( meta1&ob meta2 🡺 meta1 meta2 )

Quita el último objeto de meta1.

29972 pshzer ( meta 🡺 #0 meta )

36946 SWAPUnDROP ( meta1 meta2 🡺 meta2 ob1...obn )

Intercambia 2 metas y borra el contador.

Debería llamarse swapDROP.

2F38E xnsgeneral ( meta 🡺 LAM3&meta&LAM1 )

Usa el contenido de LAM1 y LAM3.

2F38F xsngeneral ( meta 🡺 meta&LAM3&LAM1 )

Usa el contenido de LAM1 y LAM3.

12.1.4 Otras operaciones

**Direcc. Nombre Descripción**

3760D SubMetaOb ( meta #inicio #fin 🡺 meta' )

Consigue un submeta.

#1 ≤ #inicio ≤ #fin

37685 SubMetaOb1 ( ob1..obi..obn ob' ob'' #n #i 🡺 ob1..obi ob' ob'' )

Este comando puede ser usado para conseguir los primeros i objetos de una meta.

ob' y ob'' son dos objetos cualesquiera.

33F006 ˆsubmeta ( meta #inicio #fin 🡺 meta' )

Consigue un submeta.

#1 ≤ #inicio ≤ #fin

Hace lo mismo que SubMetaOb, pero un poco más rápido.

2F356 metatail ( ob1..obn #i #n+1 🡺 ob1..obn-i #n-i obn-i+1..obn #i )

Parte un meta en dos, en la posición n-i

meta1 contendrá #n-i elementos.

meta2 contendrá #i elementos.

#0 ≤ #i ≤ #n

385006 ˆmetasplit ( meta #i 🡺 meta1 meta2 )

Parte un meta en dos, en la posición i.

meta1 contendrá #i elementos.

meta2 contendrá #n-i elementos.

#0 ≤ #i ≤ #n

De no cumplirse la desigualdad, genera el error

“Dimensión inválida”

39F006 ˆmetaEQUAL? ( meta2 meta1 🡺 meta2 meta1 flag )

Retorna TRUE si los metas son iguales (EQUAL).

3BF006 ˆEQUALPOSMETA ( Meta ob 🡺 Meta ob #pos/#0 )

Retorna la última ocurrencia de ob en el meta. Si un componente del meta es lista, simbólico o matriz, entonces busca si ob es un elemento de este compuesto del meta.

**Direcc. Nombre Descripción**

3C0006 ˆEQUALPOS2META ( Meta2 Meta1 ob 🡺 Meta2 Meta1 ob #pos/#0 )

Retorna la última ocurrencia de ob en meta1 (empieza a buscar aquí) o en meta2.

Si un componente del meta es lista, simbólico o matriz, entonces busca si ob es un elemento de este compuesto del meta.

Si está en meta1, retorna MINUSONE-#posmeta1

Si no está en meta1, pero si en meta2, retorna #posmeta2

363006 ˆinsertrow[] ( ob #i metan 🡺 metan+1 )

Inserta ob en el meta en la posición #i.

Si no cumple 1 ≤ #i ≤ #n+1, genera el error “Argumento: valor incorr”. No verifica el número de objetos en la pila.

36C006 ˆMETAMAT-ROW ( metan #i 🡺 metan-1 obi )

Extrae el objeto de la posición i del meta.

Si no cumple 1 ≤ #i ≤ #n, genera el error “Argumento: valor incorr”. No verifica el número de objetos en la pila.

36F006 ˆMETAMATRSWAP ( meta #i #j 🡺 meta' )

Intercambia los elementos de las posiciones i y j del meta.

#1 ≤ #i ≤ #n

#1 ≤ #j ≤ #n

No verifica los argumentos.

12.2 Ejemplos

Ejemplo 1 Metas

**Insertar un objeto en un meta.**

Puedes usar alguno uno de los dos siguientes NULLNAME:

\* Inserta un objeto en el meta en la posición i

\* (cuenta desde ARRIBA)

\* i: 1,2,...,n+1

NULLNAME MetaIns\_Arriba ( meta ob #i -> meta' )

:: ( meta ob #i )

3PICK ( meta ob #i #n )

SWAP#- ( meta ob #n-i )

#3+ ( meta ob #n-i+3 )

UNROLL ( ob1 meta' )

#1+ ( meta' )

;

\* Inserta un objeto en el meta en la posición i

\* (cuenta desde ABAJO)

\* i: n+1,...,2,1

NULLNAME MetaIns\_Abajo ( meta ob #i -> meta' )

:: ( meta ob #i )

#1+UNROLL ( ob1 meta' )

#1+ ( meta' )

;

Ejemplo 2 Metas

**Borrar un objeto de un meta.**

Puedes usar alguno uno de los dos siguientes NULLNAME:

\* Borra el objeto de la posición i del meta

\* (cuenta desde ARRIBA)

\* i: 1,2,...,n

NULLNAME MetaDel\_Arriba ( meta #i -> meta' )

:: ( meta #i )

OVER ( meta #i #n )

SWAP#- ( meta #n-i )

#2+ ( meta #n-i+2 )

ROLLDROP ( ... #n )

#1- ( meta' )

;

\* Borra el objeto de la posición i del meta

\* (cuenta desde ABAJO)

\* i: n,...,2,1

NULLNAME MetaDel\_Abajo ( meta #i -> meta' )

:: ( meta #i )

#1+ ( meta #i+1 )

ROLLDROP ( ... #n )

#1- ( meta' )

;

Ejemplo 3 Metas

**Conseguir un objeto de un meta.**

Puedes usar alguno uno de los dos siguientes NULLNAME:

\* Pone en el nivel 1 de la pila el objeto de la posición i del meta

\* (cuenta desde ARRIBA)

\* i: 1,2,...,n

NULLNAME MetaGet\_Arriba ( meta #i -> meta obi )

:: ( meta #i )

OVER ( meta #i #n )

SWAP#- ( meta #n-i )

#2+PICK ( meta obi )

;

\* Pone en el nivel 1 de la pila el objeto de la posición i del meta

\* (cuenta desde ABAJO)

\* i: n,...,2,1

NULLNAME MetaGet\_Abajo ( meta #i -> meta obi )

:: ( meta #i )

#1+PICK ( meta ob )

;

Ejemplo 4 Metas

**Reemplazar un objeto de un meta.**

Puedes usar alguno uno de los dos siguientes NULLNAME:

\* Reemplaza el objeto de la posición i del meta por el del nivel 2

\* (cuenta desde ARRIBA)

\* i: 1,2,...,n

NULLNAME MetaRepl\_Arriba ( meta ob #i -> meta' )

:: ( meta ob #i )

3PICK ( meta ob #i #n )

SWAP#- ( meta ob #n-i )

#2+ ( meta ob #n-i+3 )

UNPICK\_ ( meta' )

;

\* Reemplaza el objeto de la posición i del meta por el del nivel 2

\* (cuenta desde ABAJO)

\* i: n,...,2,1

NULLNAME MetaRepl\_Abajo ( meta ob #i -> meta' )

:: ( meta ob #i )

#1+UNPICK\_ ( meta' )

;

Capítulo 13  
Objetos Unidad

Las unidades son otro tipo de objetos compuestos. En realidad, no es difícil incluir uno en un programa, sólo es laborioso.

Las unidades empiezan con la palabra UNIT y terminan con ;. Dentro, tienen a los comandos que definen la unidad. La mejor manera de comprender como una unidad está representada es descompilandola. El objeto unidad 980\_cm/sˆ2 puede ser creado usando el siguiente código:

::

UNIT

% 980.

CHR c

"m"

umP

"s"

%2

um^

um/

umEND

;

;

Como puedes ver, la creación de unidades es hecha en notación polaca inversa usando las palabras umˆ, um\*, um/ y umP. El significado de las tres primeras palabras es fácil de adivinar. La última (umP) es usada para crear prefijos (kilo, mega, mili, etc.). Primero ingresa el prefijo como un carácter, y luego el nombre de la unidad como una cadena. Luego ingresa umP y la unidad con su prefijo es creada. Luego llama a las otras palabras que sean necesarias. Para terminar una unidad usa la palabra umEND, la cual junta el número (ingresado primero) a la unidad. El código de arriba se puede hacer más corto si usamos caracteres y cadenas ya incorporados en ROM (listados en el capítulo 6).

Puesto que las unidades son objetos compuestos, puedes usar los comandos referentes a objetos compuestos (capítulo 11) al manejar unidades. Por ejemplo, puedes usar INNERCOMP para descomponer una unidad en un objeto meta (ver capítulo 12). Para crear un objeto unidad a partir de un objeto meta usa el comando EXTN. Por ejemplo, el programa de abajo agrega la unidad m/s al número que está en la pila.

::

CK1NOLASTWD

CKREAL

"m"

"s"

um/

umEND

BINT5

EXTN

;

Observa que las palabras que empiezan con um, cuando son ejecutadas, solo se ponen ellas mismas en la pila.

Varias operaciones pueden ser hechas con unidades. La lista completa está dada abajo. Las más importantes son UM+, UM-, UM\* y UM/, cuyos significados son obvios; UMCONV, equivalente al comando **CONVERT** de User RPL; UMSI, equivalente al comando **UBASE** de User RPL, y U>nbr, el cual retorna la parte numérica de una unidad.

13.1 Referencia

13.1.1 Creando Unidades

**Direcc. Nombre Descripción**

2D74F um\* \* marca

2D759 um/ / marca

2D763 umˆ ˆ marca

2D76D umP Operador para el prefijo de la unidad.

Explicado arriba.

2D777 umEND Operador que marca el final de la unidad.

05481 EXTN ( ob1..obn #n 🡺 u )

Crea un objeto unidad.

13.1.2 Unidades ya incorporadas en ROM

**Direcc. Nombre Descripción**

2D837 (unit\_kg) 1\_kg

Kilogramo

2D863 (unit\_m) 1\_m

metro

2D883 (unit\_A) 1\_A

Amperio

2D8A3 (unit\_s) 1\_s

segundo

2D8C3 (unit\_K) 1\_K

Kelvin

2D8E3 (unit\_cd) 1\_cd

Candela

2D905 (unit\_mol) 1\_mol

mol

2D7A9 (unit\_r) 1\_r

radián

2D7C9 (unit\_sr) 1\_sr

estéreoradián

2D929 (unit\_?) 1\_ ?

2D781 (SIbasis) { 1\_kg 1\_m 1\_A 1\_s 1\_K 1\_cd 1\_mol 1\_r 1\_sr 1\_? }

2D7F5 (unit\_R) 1\_ºR

Rankine

13.1.3 Comandos Generales

**Direcc. Nombre Descripción**

2F099 U>NCQ ( u 🡺 %%num %%factor [] )

( % 🡺 %% %%1 [%0…%0] )

Retorna la parte real, el factor de conversión hacia el sistema internacional y un arreglo real de la forma:

[ kg m A s K cd mol r sr ? ]

Donde cada elemento representa el exponente de esa unidad.

Por ejemplo, 25\_N U>NCQ retornará:

%% 2.5E1 %%1 [ 1 1 0 -2 0 0 0 0 0 0 ]

Debido a que 1\_N es equivalente a 1\_kg\*m/sˆ2

**Direcc. Nombre Descripción**

2F07A UM>U ( % u 🡺 u' )

Reemplaza la parte numérica de la unidad.

Equivale al comando **→UNIT** de User RPL.

2F08C UMCONV ( u1 u2 🡺 u1' )

Cambia las unidades de u1 a unidades de u2.

Equivale al comando **CONVERT** de User RPL.

2F090 UMSI ( u 🡺 u' )

Convierte a las unidades básicas en el SI.

Equivale al comando **UBASE** de User RPL.

2F095 UMU> ( u 🡺 % u' )

Retorna número y parte normalizada de la unidad.

Para unidades, equivale al comando **OBJ→** de User RPL.

Por ejemplo: 36\_in 🡺 36. 1\_in

2F019 UNIT>$ ( u 🡺 $ )

Convierte unidad a cadena sin comillas.

Ejemplo con UNIT>$: 36\_in 🡺 "36.\_in"

Ejemplo con DO>STR: 36\_in 🡺 "'36.\_in'"

2F07B U>nbr ( u 🡺 % )

Retorna la parte numérica de la unidad.

Para esto también puedes usar el comando CARCOMP.

Equivale al comando **UVAL** de User RPL.

2F098 Unbr>U ( u % 🡺 u' )

Reemplaza la parte numérica de la unidad.

Hace SWAP luego UM>U.

2F09A TempConv ( %%Orig F\_Orig F\_Dest ob ob' 🡺 %%final )

Usada por UMCONV para convertir unidades de temperatura.

%%Orig es la parte numérica de la unidad de origen.

F\_Orig es el factor de conversión de la unidad de origen a Kelvin.

F\_Dest es el factor de conversión de la unidad de destino a Kelvin.

ob y ob' son dos objetos cualesquiera.

Los factores son:

%%1 para Kelvin,

cfC para centígrados,

%% 0.555555555555556 para Rankine

cfF para Farenheit.

Por ejemplo, para convertir 37\_ºC a Farenheit:

:: %% 3.7E1 cfC cfF ZEROZERO ;

devuelve %% 9.86E1

25EE4 KeepUnit ( % ob u 🡺 u' ob )

( % ob ob' 🡺 % ob )

Si el objeto del nivel uno es un objeto de unidad, entonces reemplaza la parte numérica de este con el número del nivel 3.

Si no lo es, entonces sólo hace DROP.

3902E ( u u' 🡺 u'' )

Muestra la unidad u de otra manera. De tal manera que las unidades de u' se muestren también en su representación.

Equivale al comando **UFACT** de User RPL.

13.1.4 Operaciones con unidades

**Direcc. Nombre Descripción**

2F081 UM+ ( u u' 🡺 u'' )

2F082 UM- ( u u' 🡺 u'' )

2F080 UM\* ( u u' 🡺 u'' )

2F083 UM/ ( u u' 🡺 u'' )

2F07D UM% ( u %porcentaje 🡺 (%porcentaje/100)\*u )

2F07F UM%T ( u u' 🡺 u'/u \* 100 )

2F07E UM%CH ( u u' 🡺 u'/u \* 100 – 100 )

2F08F UMMIN ( u u' 🡺 u? )

2F08E UMMAX ( u u' 🡺 u? )

2F096 UMXROOT ( u u' 🡺 u'' )

( u % 🡺 u' )

Halla la raíz enésima de la unidad.

2F08A UMABS ( u 🡺 |u| )

2F08B UMCHS ( u 🡺 -u )

2F092 UMSQ ( u 🡺 u2 )

2F093 UMSQRT ( u 🡺 √u )

2D949 UMSIGN ( u 🡺 % )

Retorna el signo de la parte real de la unidad.

Devuelve -1., 0. ó 1.

2D95D UMIP ( u 🡺 u' )

Parte entera.

2D971 UMFP ( u 🡺 u' )

Parte decimal.

2D985 UMFLOOR ( u 🡺 u' )

2D999 UMCEIL ( u 🡺 u' )

2D9CB UMRND ( u % 🡺 u' )

2D9EE UMTRC ( u % 🡺 u' )

2F091 UMSIN ( u 🡺 u' )

2F08D UMCOS ( u 🡺 u' )

2F094 UMTAN ( u 🡺 u' )

00F0E7 ROMPTR 0E7 00F ( u u' 🡺 u+u' )

Suma dos unidades de temperatura.

El resultado tiene las mismas unidades que u.

Equivale al comando **TINC** de User RPL.

00E0E7 ROMPTR 0E7 00F ( u u' 🡺 u+u' )

Suma dos unidades de temperatura.

El resultado tiene las mismas unidades que u.

Equivale al comando **TDELTA** de User RPL.

13.1.5 Tests

**Direcc. Nombre Descripción**

2F087 UM=? ( u u' 🡺 %flag )

2F07C UM#? ( u u' 🡺 %flag )

2F086 UM<? ( u u' 🡺 %flag )

2F089 UM>? ( u u' 🡺 %flag )

2F085 UM<=? ( u u' 🡺 %flag )

2F088 UM>=? ( u u' 🡺 %flag )

2F076 puretemp? ( [] []' 🡺 [] []' flag )

Devuelve TRUE si ambos arreglos reales representan la unidad temperatura.

Compara con EQ si los arreglos son iguales a

[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]

13.2 Ejemplos

Ejemplo 1 Unidades

**Tests adicionales sobre unidades:**

**¿Cómo puedo reconocer si una unidad es válida o inválida?**

Para saber si la unidad que está en la pila es válida o inválida, puedes usar lo siguiente. Devuelve TRUE si la unidad es válida o FALSE si la unidad es inválida.

Por ejemplo, retorna TRUE para la unidad 12\_N\*s

También retorna TRUE para la unidad 12\_N\*ZZZ si ZZZ es una variable global que contiene a una unidad válida.

NULLNAME Unidad\_Valida? ( u -> T/F )

:: ( u )

DEPTH

#1- ( u #depth ) ( )

1LAMBIND ( u )

ERRSET

:: U>NCQ ( %%num %%factor [%] )

3DROPTRUE\_ ( T )

;

ERRTRAP

:: DEPTH ( ... #DEPTH )

1GETLAM ( ... #DEPTH #depth )

#- ( ... #DEPTH-depth )

NDROPFALSE ( F )

;

( T/F )

ABND ( T/F )

;

Ejemplo 2 Unidades

**Tests adicionales sobre unidades:**

**¿Cómo puedo reconocer si una unidad es válida o inválida?**

Para saber si dos objetos de unidad son compatibles (tienen la misma dimensión) o no, puedes usar la siguiente subrutina. En la pila deben estar dos unidades válidas. Devuelve TRUE si son compatibles o FALSE si no lo son.

NULLNAME Unidades\_Compatibles? ( u u' -> T/F )

:: U>NCQ ( u %% %% []' )

UNROT2DROP ( u []' )

SWAP ( []' u )

U>NCQ ( []' %% %% [] )

UNROT2DROP ( []' [] )

EQUAL ( T/F )

;

Capítulo 14  
Objetos Simbólicos (SYMB)

Los objetos simbólicos (llamados también objetos algebraicos) también son un tipo de objeto compuesto.

Su estructura es muy similar a la de la unidades. Los objetos simbólicos son delimitados por SYMBOL y ;. Dentro de los delimitadores, la expresión es creada en notación polaca inversa.

Al descompilar la ecuación R = V/I, podemos ver como se deben escribir en el editor de Debug4x para que incluyas objetos simbólicos en tus programas.

SYMBOL

ID R

ID V

ID I

x/

x=

;

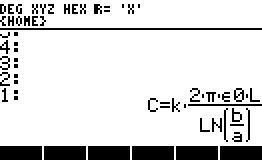
Como has visto, las variables son representadas mediante identificadores, y las funciones son funciones accesibles al usuario (User RPL), cuyos nombres son precedidos por una x minúscula en System RPL.

Para crear un objeto simbólico a partir de un meta, usa el comando SYMBN.

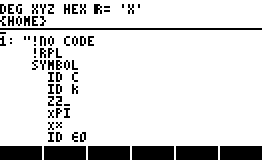
La mayoría de comandos que trata con simbólicos se encuentran en el CAS. Los comandos del CAS son tratados a partir del capítulo 43, principalmente en los capítulos 47 y 48. Sin embargo, algunos comandos que ya estaban en la HP48 han sido mantenidos por razones de compatibilidad. Estos comandos son listados aquí.

Para conseguir un objeto simbólico en el editor de Debug 4x de una manera más sencilla puedes seguir los siguientes pasos:

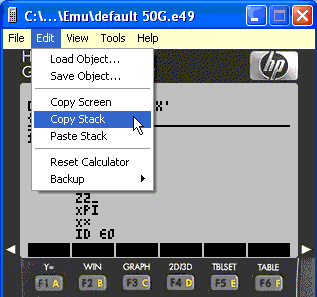
A) En el emulador, coloca el objeto simbólico en el nivel 1 de la pila.



B) Ejecuta el comando **→S2** de User RPL. A continuación verás la representación System RPL del objeto algebraico en la pila.

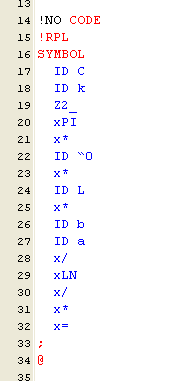
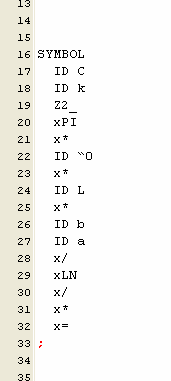


C) Selecciona Edit, luego Copy Stack.



D) En el editor de Debug 4x, presiona CTRL+V. De esta manera tendrás en el editor de Debug 4x al objeto simbólico en su representación System RPL.

Luego debes quitar las líneas innecesarias.

Este procedimiento también es útil para la mayoría de los demás objetos, como por ejemplo objetos gráficos, objetos unidad, programas, etc.

En lugar de ejecutar **🡪S2**, puedes usar el siguiente programa NULLNAME el cual retorna la representación System RPL del objeto en una sóla línea.

ASSEMBLE

CON(1) 8 ( Le dice al parseador que el comando es 'No algebraico' )

RPL

xNAME OB>1L

:: ( ob )

CK1 ( ob )

FLASHPTR 004 002 ( $ ) ( Equivale a ->S2 )

BINT92 ( $ 92 ) ( masd: asm/sysrpl mode )

TestSysFlag

?SKIP

:: BINT15 LAST$ ;

( $ )

"\0A " " " ( $ $ $ )

FLASHPTR 00F 01A ( $ % ) ( Equivale a SREPL )

DROP ( $ )

"\0A;\0A@" " ;" ( $ $ $ )

FLASHPTR 00F 01A ( $ % ) ( Equivale a SREPL )

DROP ( $ )

"\0A@" " " ( $ $ $ )

FLASHPTR 00F 01A ( $ % ) ( Equivale a SREPL )

DROP ( $ )

BINT15 BINT2

DO

( $ )

INDEX@ ( $ #i )

Blank$ ( $ "..." )

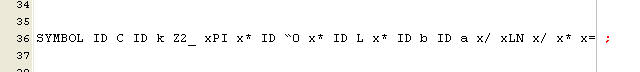
SPACE$ ( $ "..." " " )

FLASHPTR 00F 01A ( $' % )

DROP ( $' )

LOOP

;



14.1 Referencia

14.1.1 Operaciones generales

**Direcc. Nombre Descripción**

0546D SYMBN ( ob1..obn #n 🡺 symb )

2BD8C (Cr) ( ob1..obn-1 #n-1 🡺 symb )

Run Stream:

( obn 🡺 )

Hace 'R, SWAP#1+ luego SYMBN.

Crea un simbólico del meta en la pila y el siguiente objeto en el runstream. Este objeto es agregado al final del simbólico.

286E7 symcomp ( symb 🡺 symb )

( ob 🡺 symb )

Si ob es simbólico, no hace nada.

Si ob no es simbólico, hace ONE SYMBN.

2F073 SWAPcompSWAP ( symb ob' 🡺 symb ob' )

( ob ob' 🡺 symb ob' )

Hace SWAP symcomp SWAP.

28ACE (DROP?symcomp) ( %/C%/id/lam/Z/u ob' 🡺 %/C%/id/lam/Z/u )

( symb ob' 🡺 symb )

( ob ob' 🡺 symb )

Borra ob'.

Luego, si el objeto de la pila es real, complejo, de clase simbólica (id, lam, symb o entero) o unidad, no hace nada.

Otros objetos son convertidos a symb de un objeto con el comando symcomp

293A3 (?symcomp) ( %/C%/id/lam/Z/u #1 🡺 %/C%/id/lam/Z/u )

( symb #1 🡺 symb )

( ob #1 🡺 symb )

( ob1..obn #n 🡺 symb )

Si en el nivel 1 está BINT1, llama a DROP?symcomp.

Si en el nivel 1 está otro bint, llama a SYMBN.

25EA2 CRUNCH ( ob 🡺 % )

Versión interna del comando **→NUM** de User RPL.

2F110 (FINDVAR) ( symb 🡺 {} )

Retorna una lista de las variables de la ecuación.

Hace una búsqueda recursiva dentro de los programas y funciones que aparecen en la ecuación.

Por ejemplo, si en la pila se encuentra el simbólico 'X+Y+Z' y la variable 'X' está en la memoria y es un programa o simbólico que contiene a M y a N, entonces al aplicar el comando FINDVAR\_ este retornará la lista { M N Y Z }.

SYMBOL ID X ID Y x+ ID Z x+ ; 🡺 { ID M ID N ID Y ID Z }

462006 ˆEQUATION? ( ob 🡺 ob flag )

¿Es el simbólico una ecuación?

Retorna TRUE si ob es un simbólico que termina en x=.

Por ejemplo, retornará TRUE para 'X2+Y=18'

**Direcc. Nombre Descripción**

463006 ˆUSERFCN? ( ob 🡺 ob flag )

¿Es el simbólico una función?

Retorna TRUE si ob es un simbólico que termina en xFCNAPPLY.

Por ejemplo, retornará TRUE para 'f(X+5)'

29CB9 uncrunch ( 🡺 )

Run Stream:

( ob 🡺 )

Desactiva el flag resultados numéricos (desactiva el flag 3) solamente para ejecutar el siguiente comando del runstream con EVAL.

Por ejemplo:

SYMCOLCT = :: uncrunch colct ;

2BCA2 cknumdsptch1 ( sym 🡺 symf )

Usado por funciones (comandos User RPL permitidos en algebraicos) de un argumento cuando en la pila se encuentra un objeto de clase simbólica (symb, id, lam o entero).

Se usa de esta manera:

:: cknumdsptch1 <SYMfcn> <fcn> ;

Si el modo numérico está activado (flag 3 activado), ejecuta EVAL, CRUNCH y COLA es aplicado a <fcn>.

Si está en modo simbólico, <SYMfcn> es llamado (dos veces si el simbólico es una ecuación).

Por ejemplo, el comando **SIN** de User RPL hace:

:: cknumdsptch1 FLASHPTR xSYMSIN xSIN ;

cuando en la pila hay un objeto de clase simbólica.

2BB21 sscknum2 ( sym sym' 🡺 symf )

Usado por funciones (comandos User RPL permitidos en algebraicos) de dos argumento cuando en la pila se encuentran dos objetos de clase simbólica (symb, id, lam o entero).

Se usa de esta manera:

:: sscknum2 <SYMfcn> <fcn> ;

Si el modo numérico está activado (flag 3 activado), ejecuta EVAL, CRUNCH a los objetos y COLA es aplicado a <fcn>.

Si está en modo simbólico, <SYMfcn> es llamado.

Por ejemplo, el comando **+** de User RPL hace:

:: sscknum2 FLASHPTR xssSYM+ x+ ;

cuando en la pila hay 2 objetos de clase simbólica.

2BB3A sncknum2 ( sym % 🡺 symf )

Usado por funciones (comandos User RPL permitidos en algebraicos) de dos argumento cuando en la pila se encuentra un objeto de clase simbólica (symb, id, lam o entero) y un real.

Se usa de esta manera:

:: sncknum2 <SYMfcn> <fcn> ;

Lo realizado dependerá del estado del flag 3.

Por ejemplo, el comando **DARCY** de User RPL hace:

:: sncknum2 ROMPTR 0E7 02B xDARCY ;

cuando en la pila hay un objeto de clase simbólica y un real.

**Direcc. Nombre Descripción**

2BB53 nscknum2 ( % sym 🡺 symf )

Usado por funciones (comandos User RPL permitidos en algebraicos) de dos argumento cuando en la pila se encuentra un real y un objeto de clase simbólica (symb, id, lam o entero).

Se usa de esta manera:

:: nscknum2 <SYMfcn> <fcn> ;

Lo realizado dependerá del estado del flag 3.

Por ejemplo, el comando **DARCY** de User RPL hace:

:: nscknum2 ROMPTR 0E7 02A xDARCY ;

cuando en la pila hay un real y un objeto de clase simbólica.

14.1.2 Otros comandos

**Direcc. Nombre Descripción**

2EF26 SYMSHOW ( symb id/lam 🡺 symb' )

Devuelve un simbólico equivalente a symb, excepto que todas las variables del simbólico que están guardadas en memoria y hacen referencia al nombre id/lam son evaluadas hasta mostrar a ese nombre en el simbólico.

Para estos argumentos, es equivalente al comando **SHOW** de User RPL.

2F2A9 XEQSHOWLS ( symb {} 🡺 symf )

{} es una lista de nombres globales o locales.

Devuelve un simbólico equivalente a symb, excepto que todas las variables que no están en la lista son evaluadas.

Para estos argumentos, es equivalente al comando **SHOW** de User RPL.

14.1.3 Metas Simbólicos

**Direcc. Nombre Descripción**

29986 pshzerpsharg ( meta 🡺 meta #0 )

( meta 🡺 M\_last M\_rest )

Si meta representa a un objeto simbólico válido, agrega #0.

Si meta no representa un objeto simbólico válido (que contiene más de una expresión dentro), retorna la última subexpresión y el resto.

3701E pZpargSWAPUn ( meta 🡺 #0 meta )

( meta 🡺 M\_last M\_rest )

pshzerpsharg luego psh.

36FE2 plDRPpZparg ( meta&ob 🡺 meta #0 )

( meta&ob 🡺 M\_last M\_rest )

Quita el ultimo objeto del meta y luego llama a pshzerpsharg.

36F8D top&Cr ( meta1 meta2 🡺 symb )

Run Stream:

( obn 🡺 )

Crea un simbólico a partir de meta1, meta2 y el siguiente objeto en el runstream (este es agregado al final del simbólico).

Capítulo 15  
Objetos Gráficos (Grobs)

Los objetos gráficos (grobs) representan imágenes, dibujos, etc. Si quieres escribir programas que dibujen algo en la pantalla, debes conocer como usar los grobs, porque el contenido de la pantalla es un grob y tendrás que dibujar en ese grob o insertar otro grob en el grob de la pantalla.

En la sección de referencia hay comandos para crear, manipular y mostrar objetos gráficos.

Cuando tratamos con objetos gráficos debes tener dos cosas en mente:

1. Varios comandos que hacen operaciones con grobs, manipulan directamente al grob sin hacer una copia de este. Por lo tanto, todos los punteros a ese objeto en la pila serán modificados. Puedes usar el comando CKREF para asegurarte que ese objeto es único. Para más información de la memoria temporal ver la sección 25.1.4. Este tipo de operaciones son denominadas “tipo bang” y se indica esto en la explicación de cada comando en las secciones de referencia de comandos.

2. Existen dos pantallas en las calculadoras hp: pantalla de textos (ABUFF) y pantalla gráfica (GBUFF). La primera se usa para mostrar los objetos que están en la pila y la segunda para graficar funciones o datos estadísticos. (Más información sobre estas en el capítulo 39 sobre la pantalla). Para insertar un grob en alguna de estas pantallas puedes usar el comando GROB! o el comando XYGROBDISP.

3. Para poner la pantalla de texto en la pila, debes llamar al comando ABUFF. (🡪 grob )

4. Para poner la pantalla gráfica en la pila, debes llamar al comando GBUFF. (🡪 grob )

5. Para volver activa a la pantalla de texto, debes llamar al comando TOADISP. ( 🡪 )

6. Para volver activa a la pantalla gráfica, debes llamar al comando TOGDISP. ( 🡪 )

15.1 Referencia

15.1.1 Grobs ya incorporados en ROM

**Direcc. Nombre Descripción**

27AA3 (NULLPAINT) ( 🡺 grob )

0x0 Grob nulo.

27D3F CROSSGROB ( 🡺 grob )

5x5 Cursor cruz 

27D5D MARKGROB ( 🡺 grob )

5x5 Grob marca 

0860B0 ˜grobAlertIcon ( 🡺 grob )

9x9 Grob ícono de alerta 

27D7B (NullMenuLbl) ( 🡺 grob )

21x8 Etiqueta de menu normal vacía 

2E25C (InvLabelGrob) ( 🡺 grob )

21x8 Etiqueta de menú inversa vacía 

279F6 (StdBaseLabel) ( 🡺 grob )

21x8 Etiqueta de menu normal vacía inversa 

2E198 (BoxLabelGrobInv) ( 🡺 grob )

21x8 Etiqueta de menú con cuadrado inversa vacía 

2E1FA (DirLabelGrobInv) ( 🡺 grob )

21x8 Etiqueta de menú tipo directorioinversa vacía 

0870B0 ˜grobCheckKey ( 🡺 grob )

21x8 Etiqueta de menú con un “CHK” 

0880B0 ROMPTR 0B0 088 ( 🡺 grob )

131x7 Grob título 

07F0B0 ROMPTR 0B0 07F ( 🡺 grob )

6x9 “CHEK”

0800B0 ROMPTR 0B0 080 ( 🡺 grob )

6x9 “CHEK” subrayado

0810B0 ROMPTR 0B0 081 ( 🡺 grob )

6x9 “CHEK” inverso

0820B0 ROMPTR 0B0 082 ( 🡺 grob )

6x9 “CHEK” subrayado inverso

0830B0 ROMPTR 0B0 083 ( 🡺 grob )

6x9 subrayado

0840B0 ROMPTR 0B0 084 ( 🡺 grob )

6x9 subrayado inverso

0850B0 ROMPTR 0B0 085 ( 🡺 grob )

6x9 inverso

15.1.2 Dimensiones

**Direcc. Nombre Descripción**

26085 GROBDIM ( grob 🡺 #h #w )

Retorna la altura y el ancho del grob.

25EBB DUPGROBDIM ( grob 🡺 grob #h #w )

Retorna la altura y el ancho del grob.

36C68 GROBDIMw ( grob 🡺 #w )

Retorna sólo el ancho del grob.

**Direcc. Nombre Descripción**

2F324 CKGROBFITS ( g1 g2 #n #m 🡺 g1 g2' #n #m )

Reduce grob2, si este no cabe en grob1 (si quisieramos insertarlo en el grob1 en las posiciones #n,#m).

2F320 CHECKHEIGHT ( ABUFF/GBUFF #h 🡺 )

Si #h es menor a 80 (#50) en la HP 50g y HP 49g+, agrega 80-h lineas horizontales al grob en la parte inferior.

Generalmente, #h es la altura inicial del grob, y de esta manera, se fuerza a que el grob tenga una altura final de por lo menos 80.

El gráfico debe tener un ancho diferente a cero.

15.1.3 Manejo de Grobs

**Direcc. Nombre Descripción**

2607B GROB! ( grob1 grob2 #x #y 🡺 )

Reemplaza grob1 en grob2 en la posición indicada.

No importa el tamaño de los grobs, estos no cambiarán su tamaño. Si los bints son muy grandes, no hay problema.

(comando tipo bang)

26080 GROB!ZERO ( grob #x1 #y1 #x2 #y2 🡺 grob' )

Limpia una región del grob.

Si los bints son muy grandes, no hay problema.

(comando tipo bang)

368E7 GROB!ZERODRP ( grob #x1 #y1 #x2 #y2 🡺 )

Limpia una región del grob.

Si los bints son muy grandes, no hay problema.

(comando tipo bang)

2EFDB (GROB+) ( grob1 grob2 🡺 grob )

Combina dos grobs en modo OR.

Los grobs deben tener las mismas dimensiones. De lo contrario, genera el error “Argumento: valor incorr”

Equivale al comando **+** de user RPL cuando en la pila hay 2 grobs.

2F342 GROB+# ( flag grob1 grob2 #x #y 🡺 grob' )

Inserta grob2 en grob1 en la posición especificada.

Si el flag es TRUE, lo hace en modo OR.

Si el flag es FALSE, lo hace en modo XOR.

Si #x, #y son incorrectos, genera el error.

“Argumento: valor incorr”

Si es necesario, reduce a grob2, para que este entre en grob1.

(comando tipo bang)

2612F SUBGROB ( grob #x1 #y1 #x2 #y2 🡺 grob' )

Retorna una porción de un grob.

Los bints pueden estar en desorden y ser mayores al tamaño del grob.

260B2 MAKEGROB ( #h #w 🡺 grob )

Crea un grob en blanco con la altura y el ancho especificados.

Usado por el comando **BLANK** de User RPL.

2609E INVGROB ( grob 🡺 grob' )

Invierte todos los píxeles del grob.

Usado por el comando **NEG** de User RPL cuando en la pila hay un grob.

(comando tipo bang)

**Direcc. Nombre Descripción**

0BF007 ˆGROBADDext ( grob2 grob1 🡺 ob )

Adición vertical de grobs.

grob2 estará arriba y grob1 abajo.

Equivale al comando **GROBADD** de User RPL.

280C1 ORDERXY# ( #x1 #y1 #x2 #y2 🡺 #x1' #y1' #x2' #y2' )

Ordena los bints para que estos sean apropiados para definir un rectángulo en un grob.

Al final, tendremos:

#x1' ≤ #x2'

#y1' ≤ #y2'

280F8 ORDERXY% ( %x1 %y1 %x2 %y2 🡺 %x1' %y1' %x2' %y2' )

Ordena los números reales para que estos sean apropiados para definir un rectángulo en un grob.

Al final, tendremos:

%x1' ≤ %x2'

#y1' ≤ #y2'

255B5 Distance ( #Δx #Δy 🡺 #SQRT(Δxˆ2+Δyˆ2) )

Halla la distancia entre dos puntos. Redondea hacia abajo.

25F0E XYGROBDISP ( #x #y grob 🡺 )

Pone grob en HARDBUFF con la esquina superior izquierda en (#x,#y).

Si HARDBUFF es GBUFF, este GBUFF es expandido si es necesario para que contenga al grob (de dimensiones wxh). De esta manera, se fuerza a GBUFF a tener un ancho mínimo de #x+w y una altura mínima de #y+h

15.1.3 GBUFF

GBUFF es la pantalla que en User RPL se usa para graficar las funciones guardadas en la variable EQ o en ∑DAT.

Podrás conocer más comandos que manipulan a GBUFF en el capítulo 42 sobre trazado de gráficos y en el capítulo 39 sobre la pantalla.

**Direcc. Nombre Descripción**

2F0DB MAKEPICT# ( #w #h 🡺 )

Crea un grob en blanco y lo establece como el nuevo GBUFF.

El tamaño mínimo es 131x80 en la HP 50g y HP 49g+.

Grobs más pequeños serán redimensionados.

El ancho máximo es 2048.

Usado por el comando **PDIM** de User RPL cuando en la pila hay dos hxs.

25ED8 GROB>GDISP ( grob 🡺 )

Establece a grob como el nuevo GBUFF.

260DA PIXON3 ( #x #y 🡺 )

Activa el píxel indicado de GBUFF.

El punto puede estar fuera de GBUFF.

260D5 PIXOFF3 ( #x #y 🡺 )

Apaga el píxel indicado de GBUFF.

El punto puede estar fuera de GBUFF.

260E9 PIXON?3 ( #x #y 🡺 flag )

Si el píxel indicado de GBUFF está activado, retorna TRUE.

El punto puede estar fuera de GBUFF.

GBUFF no debe ser un grob nulo.

**Direcc. Nombre Descripción**

2EFA2 LINEON3 ( #x1 #y1 #x2 #y2 🡺 )

Dibuja una línea en GBUFF.

x1 debe ser menor o igual a x2.

Los puntos pueden estar fuera de GBUFF.

GBUFF no debe ser un grob nulo.

2F13F DRAWLINE#3 ( #x1 #y1 #x2 #y2 🡺 )

Dibuja una línea en GBUFF.

No se requiere que x1 sea ser menor o igual a x2.

Los puntos pueden estar fuera de GBUFF.

2EFA3 LINEOFF3 ( #x1 #y1 #x2 #y2 🡺 )

Limpia una línea en GBUFF.

x1 debe ser menor o igual a x2.

Los puntos pueden estar fuera de GBUFF.

GBUFF no debe ser un grob nulo.

2EFA4 TOGLINE3 ( #x1 #y1 #x2 #y2 🡺 )

Hace XOR a una línea en GBUFF.

x1 debe ser menor o igual a x2.

Los puntos pueden estar fuera de GBUFF.

GBUFF no debe ser un grob nulo.

2F382 TOGGLELINE#3 ( #x1 #y1 #x2 #y2 🡺 )

Hace XOR a una línea en GBUFF.

No se requiere que x1 sea ser menor o igual a x2.

Los puntos pueden estar fuera de GBUFF.

2F32C DRAWBOX# ( #x1 #y1 #x2 #y2 🡺 )

Dibuja el perímetro de un rectángulo en GBUFF.

Los bints pueden estar en desorden y ser mayores al tamaño del grob.

15.1.3 ABUFF

ABUFF es la pantalla que normalmente se usa para observar los objetos que están en la pila y donde vemos la edición de la mayoría de los objetos.

Podrás conocer más comandos que manipulan a ABUFF en el capítulo 39 sobre la pantalla.

**Direcc. Nombre Descripción**

260E4 PIXON ( #x #y 🡺 )

Activa el píxel indicado de ABUFF.

El punto puede estar fuera de ABUFF.

260DF PIXOFF ( #x #y 🡺 )

Apaga el píxel indicado de ABUFF.

El punto puede estar fuera de ABUFF.

260EE PIXON? ( #x #y 🡺 flag )

Si el pixel indicado de ABUFF está activado, retorna TRUE.

El punto puede estar fuera de ABUFF.

2EF9F LINEON ( #x1 #y1 #x2 #y2 🡺 )

Dibuja una línea en ABUFF.

x1 debe ser menor o igual a x2.

Los puntos pueden estar fuera de ABUFF.

2EFA0 LINEOFF ( #x1 #y1 #x2 #y2 🡺 )

Limpia una línea en ABUFF.

x1 debe ser menor o igual a x2.

Los puntos pueden estar fuera de ABUFF.

**Direcc. Nombre Descripción**

2EFA1 TOGLINE ( #x1 #y1 #x2 #y2 🡺 )

Hace XOR a una línea en ABUFF.

x1 debe ser menor o igual a x2.

Los puntos pueden estar fuera de ABUFF.

2EF03 DOLCD> ( 🡺 grob )

Retorna un grob de tamaño 131x80 en la HP 50g y HP 49g+.

Las filas 0-71 corresponden a las primeras 72 de ABUFF.

Las filas 72-79 corresponden a HARDBUFF2 (los menús).

Equivale al comando **LCD→** de User RPL.

2EF04 DO>LCD ( grob 🡺 )

Pone el grob en la pantalla ABUFF.

El tamaño de ABUFF no cambia al usar este comando.

Primero se limpia la pantalla ABUFF.

Luego, se inserta grob en ABUFF en la esquina superior izquierda. Si grob es muy grande, este se redimensiona para que entre en ABUFF.

Equivale al comando **→LCD** de User RPL.

15.1.4 Gráficos con Escala de Grises

**Direcc. Nombre Descripción**

25592 SubRepl ( grb1 grb2 #x1 #y1 #x2 #y2 #W #H 🡺 grb1' )

Reemplaza una parte de grob2 (con esquina superior izquierda en #x2 #y2, ancho #w y altura #h) en grob1 (posición #x1 #y1) en modo REPLACE.

Si los bints son muy grandes, no hay problemas.

(comando tipo bang)

25597 SubGor ( grb1 grb2 #x1 #y1 #x2 #y2 #W #H 🡺 grb1' )

Parecido a SubRepl, pero reemplaza el grob en modo OR.

Si los bints son muy grandes, no hay problemas.

(comando tipo bang)

2559C SubGxor ( grb1 grb2 #x1 #y1 #x2 #y2 #W #H 🡺 grb1' )

Parecido a SubRepl, pero reemplaza el grob en modo XOR.

Si los bints son muy grandes, no hay problemas.

(comando tipo bang)

25565 LineW ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Dibuja una línea blanca.

No se requiere que x1 sea ser menor o igual a x2.

Los puntos deben estar dentro del objeto gráfico, el cual no debe ser un grob nulo.

(comando tipo bang)

2556F LineG1 ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Dibuja una línea gris oscuro.

No se requiere que x1 sea ser menor o igual a x2.

Los puntos deben estar dentro del objeto gráfico, el cual no debe ser un grob nulo.

(comando tipo bang)

25574 LineG2 ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Dibuja una línea gris claro.

No se requiere que x1 sea ser menor o igual a x2.

Los puntos deben estar dentro del objeto gráfico, el cual no debe ser un grob nulo.

(comando tipo bang)

**Direcc. Nombre Descripción**

2556A LineB ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Dibuja una línea negra.

No se requiere que x1 sea ser menor o igual a x2.

Los puntos deben estar dentro del objeto gráfico, el cual no debe ser un grob nulo.

(comando tipo bang)

25579 LineXor ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Hace XOR a una línea.

No se requiere que x1 sea ser menor o igual a x2.

Los puntos deben estar dentro del objeto gráfico, el cual no debe ser un grob nulo.

(comando tipo bang)

2F218 CircleW ( grb #Cx #Cy #r 🡺 grb' )

Dibuja una circunferencia blanca.

La circunferencia puede estar fuera del objeto gráfico.

(comando tipo bang)

2F216 CircleG1 ( grb #Cx #Cy #r 🡺 grb' )

Dibuja una circunferencia gris oscura.

La circunferencia puede estar fuera del objeto gráfico.

(comando tipo bang)

2F217 CircleG2 ( grb #Cx #Cy #r 🡺 grb' )

Dibuja una circunferencia gris clara.

La circunferencia puede estar fuera del objeto gráfico.

(comando tipo bang)

2F215 CircleB ( grb #Cx #Cy #r 🡺 grb' )

Dibuja una circunferencia negra.

La circunferencia puede estar fuera del objeto gráfico.

(comando tipo bang)

2F219 CircleXor ( grb #Cx #Cy #r 🡺 grb' )

Hace XOR a una circunferencia.

La circunferencia puede estar fuera del objeto gráfico.

(comando tipo bang)

2557E Sub ( grb #x1 #y1 #x2 #y2 🡺 grb' flag )

Retorna una parte de un grob y TRUE. Los bints pueden estar en desorden y ser mayores al tamaño del grob.

Si los bints son muy grandes, de manera que no representan a puntos del grob, se retorna un grob nulo y FALSE.

25583 Repl ( grb1 grb2 #x #y 🡺 grb1' )

Copia grb2 en grb1 en modo REPLACE.

Si los bints son muy grandes, no hay problemas.

(comando tipo bang)

25588 Gor ( grb1 grb2 #x #y 🡺 grb1' )

Copia grb2 en grb1 en modo OR.

Si los bints son muy grandes, no hay problemas.

(comando tipo bang)

2558D Gxor ( grb1 grb2 #x #y 🡺 grb1' )

Copia grb2 en grb1 en modo XOR.

Si los bints son muy grandes, no hay problemas.

(comando tipo bang)

255BA PixonW ( grb #x #y 🡺 grb' )

Torna al píxel color blanco.

El punto puede estar fuera del objeto gráfico.

(comando tipo bang)

**Direcc. Nombre Descripción**

255C4 PixonG1 ( grb #x #y 🡺 grb' )

Torna al píxel color gris oscuro.

El punto puede estar fuera del objeto gráfico.

(comando tipo bang)

255C9 PixonG2 ( grb #x #y 🡺 grb' )

Torna al píxel color gris claro.

El punto puede estar fuera del objeto gráfico.

(comando tipo bang)

255BF PixonB ( grb #x #y 🡺 grb' )

Torna al píxel color negro.

El punto puede estar fuera del objeto gráfico.

(comando tipo bang)

255CE PixonXor ( grb #x #y 🡺 grb' )

Hace XOR al píxel.

El punto puede estar fuera del objeto gráfico.

(comando tipo bang)

255D3 FBoxG1 ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Dibuja un rectángulo gris claro.

Sólo superpone un rectángulo relleno.

Si los bints son muy grandes o están en desorden, no hay problemas.

(comando tipo bang)

255D8 FBoxG2 ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Dibuja un rectángulo gris oscuro.

Sólo superpone un rectángulo relleno.

Si los bints son muy grandes o están en desorden, no hay problemas.

(comando tipo bang)

255DD FBoxB ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Dibuja un rectángulo negro.

Sólo superpone un rectángulo relleno.

Si los bints son muy grandes o están en desorden, no hay problemas.

(comando tipo bang)

255E2 FBoxXor ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Dibuja un rectángulo negro.

Sólo aplica XOR al rectángulo relleno.

Si los bints son muy grandes o están en desorden, no hay problemas.

(comando tipo bang)

255E7 LboxW ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Dibuja un rectángulo blanco.

Sólo dibuja los lados del rectángulo como líneas.

Si los bints son muy grandes o están en desorden, no hay problemas.

(comando tipo bang)

255EC LBoxG1 ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Dibuja un rectángulo gris claro.

Sólo dibuja los lados del rectángulo como líneas.

Si los bints son muy grandes o están en desorden, no hay problemas.

(comando tipo bang)

**Direcc. Nombre Descripción**

255F1 LBoxG2 ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Dibuja un rectángulo gris oscuro.

Sólo dibuja los lados del rectángulo como líneas.

Si los bints son muy grandes o están en desorden, no hay problemas.

(comando tipo bang)

255F6 LBoxB ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Dibuja un rectángulo negro.

Sólo dibuja los lados del rectángulo como líneas.

Si los bints son muy grandes o están en desorden, no hay problemas.

(comando tipo bang)

255FB LBoxXor ( grb #x1 #y1 #x2 #y2 🡺 grb' )

Dibuja un rectángulo negro.

Sólo aplica XOR a los lados del rectángulo como líneas.

Si los bints son muy grandes o están en desorden, no hay problemas.

(comando tipo bang)

2F21B ToGray ( grb 🡺 grb'/grb )

Convierte un grob blanco y negro a escala de grises.

2F21A Dither ( grb 🡺 grb'/grb )

Convierte un grob con escala de grises a blanco y negro.

255A1 Grey? ( grob 🡺 flag )

Retorna TRUE si el grob es un grob con escala de grises.

255B0 ScrollVGrob ( grb #w #X #Yfinal #Y #h 🡺 grb' )

La parte del objeto gráfico cuyo extremo superior izquierdo es #X, #Y y tiene ancho #w y altura #h, es copiada hacia arriba o hacia abajo, de tal manera que su nuevo extremo superior izquierdo tendrá coordenada #X, #Yfinal.

(comando tipo bang)

15.1.5 Creando Grobs que son Etiquetas del Menú

**Direcc. Nombre Descripción**

2E166 MakeStdLabel ( $ 🡺 grob )

Crea etiqueta de menú estándar.

Por ejemplo, si la cadena es "12345", retorna 

2E189 MakeBoxLabel ( $ 🡺 grob )

Crea etiqueta de menú con un cuadrado.

Por ejemplo, si la cadena es "12345", retorna 

2E1EB MakeDirLabel ( $ 🡺 grob )

Crea etiqueta de menú tipo directorio.

Por ejemplo, si la cadena es "12345", retorna 

2E24D MakeInvLabel ( $ 🡺 grob )

Crea etiqueta de menú tipo cuadro inverso.

Por ejemplo, si la cadena es "12345", retorna 

25E7F Box/StdLabel ( $ flag 🡺 grob )

Si el flag es TRUE, hace MakeBoxLabel

Si el flag es FALSE, hace MakeStdLabel

25F01 Std/BoxLabel ( $ flag 🡺 grob )

Si el flag es TRUE, hace MakeStdLabel

Si el flag es FALSE, hace MakeBoxLabel

**Direcc. Nombre Descripción**

25E80 Box/StdLbl: ( 🡺 grob )

Hace Box/StdLabel con los 2 objetos siguientes del runstream. Por ejemplo:

::

11.

Box/StdLbl:

"ABC"

:: 25. 20. %> ;

12.

;

Resultado:



15.1.6 Convertir Cadena a Grob

**Direcc. Nombre Descripción**

25F7C $>GROB ( $ 🡺 grob )

Convierte la cadena a grob usando fuente normal.

Los saltos de línea no generarán una nueva línea.

25F86 $>GROBCR ( $ 🡺 grob )

Convierte la cadena a grob usando fuente normal.

Los saltos de línea si generarán una nueva línea.

25F81 $>grob ( $ 🡺 grob )

Convierte la cadena a grob usando minifuente.

Los saltos de línea no generarán una nueva línea.

25F8B $>grobCR ( $ 🡺 grob )

Convierte la cadena a grob usando minifuente.

Los saltos de línea si generarán una nueva línea.

05F0B3 (˜$>grobOrGROB) ( $ 🡺 grob )

Convierte cadena a grob de la siguiente manera:

Si flag 90 está desactivado (CHOOSE:cur font), hace $>GROB.

Si flag 90 está activado (CHOOSE:mini font), hace $>grob

25F24 RIGHT$3x6 ( $ #w 🡺 flag grob )

Crea un grob de tamaño wx6 a partir de la cadena (minifuente).

La cadena estará en el extremo superior izquierdo del grob.

15.1.6 Insertar Cadena en un Grob por la Izquierda

**Direcc. Nombre Descripción**

25FF9 LEFT$3x5 ( grob #x #y $ #4\*nc 🡺 grob' )

Convierte la cadena a grob (con minifuente) y lo inserta en el grob del nivel 5 en modo OR.

El extremo superior izquierdo de la cadena será #x, #y.

#x, #y = 0,1,2…

#nc es el nº máx. de caracteres que se tomarán de la cadena.

Si la cadena tiene más de #nc caracteres, es truncada.

(comando tipo bang)

26071 ERASE&LEFT$3x5 ( grob #x #y $ #4\*nc 🡺 grob' )

Convierte la cadena a grob (con minifuente) y lo inserta en el grob del nivel 5 en modo REPLACE.

El extremo superior izquierdo de la cadena será #x, #y.

#x, #y = 0,1,2…

#nc es el nº máx. de caracteres que se tomarán de la cadena.

Si #nc es mayor al tamaño de la cadena, primero se limpia un rectángulo de tamaño (4•nc)x(6) en el grob.

Si la cadena tiene más de #nc caracteres, es truncada.

(comando tipo bang)

26008 LEFT$3x5Arrow ( grob #x #y $ #6\*nc 🡺 grob' )

Similar a LEFT$3x5, pero si la cadena ha sido truncada, debería reemplazar el ultimo carácter visible con el carácter 31 (puntos).

(comando tipo bang)

2601C LEFT$3x5CR ( grob #x #y $ #4\*nc #6\*nf 🡺 grob' )

Similar a LEFT$3x5, pero si hay saltos de línea, muestra nuevas líneas de texto debajo.

#nf es el número máximo de filas que se desea conseguir.

#nc es el número máximo de caracteres en una fila.

(comando tipo bang)

26012 LEFT$3x5CRArrow ( grob #x #y $ #4\*c #6\*nf 🡺 grob' )

Similar a LEFT$5x7CR, pero en cada fila que haya sido truncada, debería reemplazar el ultimo carácter visible de esa fila con el carácter 31 (puntos).

(comando tipo bang)

25FFE LEFT$5x7 ( grob #x #y $ #6\*nc 🡺 grob' )

Convierte la cadena a grob (con fuente normal) y lo inserta en el grob del nivel 5 en modo REPLACE.

El extremo superior izquierdo de la cadena será #x, #y.

#x = 0,2,4,… #y = 0,1,2,…

#nc es el nº máx. de caracteres que se tomarán de la cadena.

Si la cadena tiene más de #nc caracteres, es truncada.

(comando tipo bang)

2606C ERASE&LEFT$5x7 ( grob #x #y $ #6\*nc 🡺 grob' )

Hace lo mismo que el comando LEFT$5x7, excepto cuando #nc es mayor al tamaño de la cadena, en este caso, primero se limpia un rectángulo de tamaño (6•nc)x(hf•nf) en el grob.

(comando tipo bang)

**Direcc. Nombre Descripción**

26003 LEFT$5x7Arrow ( grob #x #y $ #6\*nc 🡺 grob' )

Similar a LEFT$5x7, pero si la cadena ha sido truncada, reemplaza el ultimo carácter visible con el carácter 31 (puntos).

(comando tipo bang)

26017 LEFT$5x7CR ( grob #x #y $ #6\*nc #hf\*nf 🡺 grob' )

Similar a LEFT$5x7, pero si hay saltos de línea, muestra nuevas líneas de texto debajo.

#hf es la altura de la fuente normal: 6,7 u 8.

#nf es el número máximo de filas que se desea conseguir.

#nc es el número máximo de caracteres en una fila.

(comando tipo bang)

2600D LEFT$5x7CRArrow ( grob #x #y $ #6\*nc #hf\*nf 🡺 grob' )

Similar a LEFT$5x7CR, pero en cada fila que haya sido truncada, reemplaza el ultimo carácter visible de esa fila con el carácter 31 (puntos).

(comando tipo bang)

15.1.6 Insertar Cadena en un Grob por el Centro

**Direcc. Nombre Descripción**

25FEF CENTER$3x5 ( grob #x #y $ #4nc 🡺 grob' )

Convierte la cadena a grob (con minifuente) y lo inserta en el grob del nivel 5 en modo OR.

La cadena es centrada en la posición #x, #y.

#x, #y = 1,2,3…

#nc es el nº máx. de caracteres que se tomarán de la cadena.

Si la cadena tiene más de nc caracteres, es truncada.

(comando tipo bang)

2E2AA MakeLabel ( $ #4n #x grob 🡺 grob' )

Inserta $ en el grob usando CENTER$3x5 con y=5.

25FF4 CENTER$5x7 ( grob #x #y $ #6\*nc 🡺 grob' )

Convierte la cadena a grob (con fuente normal) y lo inserta en el grob del nivel 5 en modo REPLACE.

La cadena es centrada en la posición #x, #y.

#x, #y = 1,2,3…

#nc es el nº máx. de caracteres que se tomarán de la cadena.

Si la cadena tiene más de nc caracteres, es truncada.

(comando tipo bang)

15.1.7 Creando Grobs a Partir de Otros Objetos

**Direcc. Nombre Descripción**

01E004 ˆEQW3GROBsys ( ob 🡺 ext grob #0 )

( ob 🡺 ob #1 )

( ob 🡺 ob #2 )

Convierte algunos objetos a grob usando la fuente normal.

Su principal uso es para convertir objetos simbólicos, arreglos reales, arreglos complejos, matrices simbólicas y unidades.

También convierte reales, complejos, ids, lams, enteros, caracteres, rompointers con nombre, algunos objetos compuestos y algunos objetos etiquetados.

Retorna #1 cuando necesita hacerse una recolección de basura (con GARBAGE).

Retorna #2 cuando no es posible convertir el objeto a grob con este comando.

01F004 ˆEQW3GROBmini ( ob 🡺 ext grob #0 )

( ob 🡺 ob #1 )

( ob 🡺 ob #2 )

Parecido al comando anterior.

Convierte algunos objetos a grob usando la minifuente.

019004 ˆEQW3GROB ( ob 🡺 ext grob #0 )

( ob 🡺 ob #1 )

( ob 🡺 ob #2 )

Parecido a los comandos anteriores.

Convierte algunos objetos a grob usando la fuente normal (flag 81 desactivado, GRB alg cur font) o la minifuente (flag 81 activado, GRB alg mini font).

01A004 ˆEQW3GROBStk ( ob 🡺 ext grob #0 )

( ob 🡺 ob #1 )

( ob 🡺 ob #2 )

Parecido a los comandos anteriores, pero también convierte cadenas (a grobs de una línea).

Convierte algunos objetos a grob usando la fuente normal (flag 80 desactivado, GRB cur stk font) o la minifuente (flag 80 activado, GRB mini stk font).

**Direcc. Nombre Descripción**

10B001 FLASHPTR 1 10B ( ob % 🡺 grob )

Convierte ob a grob.

Si el número real es 0, primero hace una recolección de basura, luego intenta convertir ob a grob con ˆEQW3GROBmini o ˆEQW3GROBsys (según el estado del flag 81). Si no es posible convertir a grob con esos comandos, ob es convertido a cadena (si aun no lo era) con ^FSTR10 y finalmente esta cadena es convertida a grob usando la fuente normal con $>GROBCR o $>GROB según el estado del flag 70.

Si el número real es 1, ob es convertido a cadena (si aun no lo era) con ^FSTR11 y finalmente esta cadena es convertida a grob usando la minifuente con $>grobCR o $>grob según el estado del flag 70.

Si el número real es 2, ob es convertido a cadena (si aun no lo era) con ^FSTR10 y finalmente esta cadena es convertida a grob usando la fuente normal con $>GROBCR o $>GROB según el estado del flag 70.

Si el número real es 3, el resultado es el mismo que con el número real 2.

Equivale al comando **→GROB** de User RPL.

0BE007 ˆXGROBext ( ob 🡺 grob )

Convierte el objeto a grob.

Si ob es un grob, no hace nada.

Si ob es una cadena, hace: %2 FLASHPTR 1 10B

Si ob es de otro tipo, hace: %0 FLASHPTR 1 10B

0C0007 ˆDISPLAYext ( grob ob 🡺 grob' )

Primero convierte ob a grob con ˆXGROBext.

Luego une los dos grobs con ˆGROBADDext.

Finalmente, muestra el grob resultante con ViewObject, sólo si el flag 100 está activado (Step by step on).

049004 ˆIfCreateTitleGrob ( grob 🡺 grob )

( $ 🡺 grob131x7 )

( # 🡺 grob131x7 )

Convierte una cadena a grob título de dimensiones 131x7.

Si en la pila hay un bint, primero se obtiene el mensaje de error correspondiente.

Si la cadena tiene más de 32 caracteres, no se verá ningun carácter de la cadena en el grob resultante.

Por ejemplo:

:: "HONRADOS HP" FLASHPTR IfCreateTitleGrob ;

Retorna el grob:



02F002 ˆMkTitle ( $ 🡺 grob131x7 )

Similar a ˆIfCreateTitleGrob, pero sólo funciona para cadenas.

Si la cadena tiene más de 32 caracteres, no se verá ningun carácter de la cadena en el grob resultante.

15.2 Ejemplos

Ejemplo 1 Grobs

**Forzando a que un grob tenga una altura de por lo menos 72.**

El siguiente NULLNAME fuerza a que el grob de la pila tenga por lo menos una altura de 72.

\* 1) Si la altura del grob es por lo menos igual a 72, no hace nada.

\* 2) Si la altura del grob es menor a 72, aumenta filas al grob para

\* que este tenga una altura igual a 72.

NULLNAME COMPLETA72 ( grob -> grob' )

::

DUPGROBDIM ( grob #h #w )

DROP ( grob #h )

DUP BINT72 ( grob #h #h 72 )

#< ( grob #h flag )

NOTcaseDROP ( grob ) ( OBS: SALE CON grob )

( grob #h )

BINT72 ( grob #h 72 )

SWAP #- ( grob #72-h )

BINT0 ( grob #72-h #0 )

MAKEGROB ( grob grob[0]x[72-h] )

FLASHPTR GROBADDext ( grob' ) ( OBS: SALE CON grob' )

;

Ejemplo 2 Grobs

**Convirtiendo un objeto en grob pequeño.**

El siguiente NULLNAME convierte un objeto a grob pequeño.

\* CONVIERTE ob A GROB PEQUEÑO USANDO MINIFUENTE

\* Intenta convertir a grob con FLASHPTR EQW3GROBmini

\* Si no es posible hace lo siguiente:

\* Si ob es un grob, no hace nada.

\* Si ob es una cadena, hace $>grobCR

\* Si ob es de otro tipo, hace FLASHPTR FSTR11 $>grobCR

NULLNAME ob>grobmini ( ob -> grob )

:: ( ob )

BEGIN

:: DUP ( ob ob )

FLASHPTR EQW3GROBmini ( ob ext grob #0 // ob ob #1/2 )

BINT0 #=casedrop

:: UNROT2DROP TRUE ; ( OBS: SALE CON grob T ) ( SALE DE BUCLE )

( ob ob #1/2 )

BINT1 #=case

:: GARBAGE DROPFALSE ; ( OBS: SALE CON ob F ) ( REPITE BUCLE )

( ob ob )

TYPEGROB? ( ob flag )

caseTRUE

( ob )

DUPTYPECSTR? ( ob flag )

?SKIP

FLASHPTR FSTR11

( $ )

$>grobCR ( grob )

TRUE ( grob T ) ( SALE DE BUCLE )

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( grob )

;

Ejemplo 3 Grobs

**Convirtiendo un objeto en grob grande.**

El siguiente NULLNAME convierte un objeto a grob grande.

\* CONVIERTE ob A GROB GRANDE USANDO FUENTE NORMAL

\* Intenta convertir a grob con FLASHPTR EQW3GROBsys

\* Si no es posible hace lo siguiente:

\* Si ob es un grob, no hace nada.

\* Si ob es una cadena, hace $>GROBCR

\* Si ob es de otro tipo, hace FLASHPTR FSTR10 $>GROBCR

NULLNAME ob>grobsys ( ob -> grob )

:: ( ob )

BEGIN

:: DUP ( ob ob )

FLASHPTR EQW3GROBsys ( ob ext grob #0 // ob ob #1/2 )

BINT0 #=casedrop

:: UNROT2DROP TRUE ; ( OBS: SALE CON grob T ) ( SALE DE BUCLE )

( ob ob #1/2 )

BINT1 #=case

:: GARBAGE DROPFALSE ; ( OBS: SALE CON ob F ) ( REPITE BUCLE )

( ob ob )

TYPEGROB? ( ob flag )

caseTRUE

( ob )

DUPTYPECSTR? ( ob flag )

?SKIP

FLASHPTR FSTR10

( $ )

$>GROBCR ( grob )

TRUE ( grob T ) ( SALE DE BUCLE )

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( grob )

;

Capítulo 16  
Bibliotecas y Objetos de Respaldo

Las bibliotecas (libraries) son objetos muy complejos que contienen una colección de comandos. Algunos de estos comandos tienen nombre y son accesibles al usuario. Pero otros, no tienen nombre y, por lo tanto, son “ocultos”. Los objetos de respaldo (backups) son usados por la calculadora, para guardar el contenido de todo el directorio HOME y restaurarlo mas tarde, cuando se desee.

La integridad de ambos objetos puede ser verificada porque tienen un código CRC unido a ellos.

Un rompointer (tambien llamado nombre XLIB) es un puntero a un comando en una biblioteca. La única forma de acceder a un comando oculto en una biblioteca es a través de un rompointer.

Un rompointer tiene dos números. El número de la biblioteca al que pertenece y el número del comando.

Para insertar un rompointer en tu programa usa la siguiente estructura:

ROMPTR <lib> <cmd>, donde <lib> es el número de la biblioteca, y <cmd> es el número del comando y su menor valor posible es 000. Ambos números se especifican en forma hexadecimal. Los rompointers siempre se ejecutan automáticamente (como los nombres globales y locales), de tal manera que tienes que citarlos (ver sección 20.2) si quieres tenerlos en la pila.

16.1 Referencia

16.1.1 Operaciones

**Direcc. Nombre Descripción**

25EEB NEXTLIBBAK ( #addr 🡺 backup/library #nextaddr )

Retorna la siguiente biblioteca o backup.

16.1.2 Rompointers

**Direcc. Nombre Descripción**

07E50 #>ROMPTR ( #lib #cmd 🡺 ROMPTR )

Crea un rompointer.

08CCC ROMPTR># ( ROMPTR 🡺 #lib #cmd )

Retorna el número de bilioteca y el número de comando de un rompointer.

07E99 ROMPTR@ ( ROMPTR 🡺 ob T )

( ROMPTR 🡺 F )

Llama al contenido de un rompointer.

35C40 DUPROMPTR@ ( ROMPTR 🡺 ROMPTR ob T )

( ROMPTR 🡺 ROMPTR F )

Hace DUP luego ROMPTR@.

35BFF RESOROMP ( 🡺 ob )

Llama al contenido del siguiente objeto en el runstream (el cual debe ser un rompointer).

Equivale a usar :: 'R ROMPTR@ DROP ;

Por ejemplo:

:: RESOROMP ROMPTR 0B3 001 ;

retorna el contenido del rompointer.

02FEF (ROMSEC) ( ROMPTR 🡺 ? )

Llama al contenido del rompointer y hace EVAL.

Si el rompointer no es encontrado, genera el error:"Nombre XLIB indefinido"

081FB (ROMPTRDECOMP) ( ROMPTR 🡺 ID T )

( ROMPTR 🡺 F )

Si es un comando de User RPL, convierte el rompointer en un id con el mismo nombre.

07C18 (COMPILEID) ( id 🡺 comando T )

( id 🡺 id T )

( id 🡺 id F )

Si el id tiene el mismo nombre que un comando de User RPL, retorna el comando (pointer o rompointer) y TRUE.

Si el id no tiene el nombre de un comando de User RPL:

- Si existe en el directorio actual o en uno de arriba, agrega TRUE.

- De lo contrario, agrega FALSE.

34FCD ROM-WORD? ( ob 🡺 flag )

Retorna TRUE para comandos de User RPL que no sean rompointer y que sean el contenido de un rompointer.

34FC0 DUPROM-WORD? ( ob 🡺 ob flag )

DUP, luego ROM-WORD?

**Direcc. Nombre Descripción**

07E76 (PTR>ROMPTR) ( ob 🡺 ROMPTR )

Para un comando de User RPL que no sea rompointer y que sea el contenido de un rompointer, retorna el rompointer que lo llama.

35A88 ?>ROMPTR ( ob 🡺 ROMPTR )

( ob 🡺 ob )

Si ROM-WORD? y TYPECOL? dan TRUE, hace PTR>ROMPTR\_

Por ejemplo:

:: ' xSIN ?ROMPTR> ; Retorna: ROMPTR 002 051

35AAB ?ROMPTR> ( ROMPTR 🡺 ob )

( ROMPTR 🡺 ROMPTR )

( ob 🡺 ob )

Si el contenido de ROMPTR es un puntero que existe en INHARDROM? entonces retorna ese puntero. Por ejemplo:

:: ' ROMPTR 002 051 ?ROMPTR> ; Retorna: xSIN

16.1.3 Vincular y Desvincular Bibliotecas

**Direcc. Nombre Descripción**

3D5001 FLASHPTR 001 3D5 ( %lib 🡺 #lib )

Convierte un real a bint.

Si es menor de 256 o es igual a 1792, genera el error “Argumento: valor incorr”

3D3001 FLASHPTR 001 3D3 ( %lib 🡺 )

Vincula la biblioteca especificada con el directorio actual.

Equivale al comando **ATTACH** de User RPL.

2F2A7 XEQSETLIB ( #lib 🡺 )

Vincula la biblioteca especificada con el directorio actual.

Usado por el comando **ATTACH** de User RPL.

3D2001 FLASHPTR 001 3D2 ( %lib 🡺 )

Desvincula la biblioteca especificada del directorio actual.

Equivale al comando **DETACH** de User RPL.

41F001 FLASHPTR 001 41F ( #lib 🡺 )

Desvincula la biblioteca especificada del directorio actual.

Usado por el comando **DETACH** de User RPL.

07709 TOSRRP ( # 🡺 )

Vincula la biblioteca especificada con el directorio HOME.

076AE OFFSRRP ( # 🡺 )

Desvincula la biblioteca especificada al directorio HOME.

0778D (ONSRRP?) ( # 🡺 flag )

Devuelve TRUE si la biblioteca está vinculada con HOME.

2F25D SETROMPART ( rrp # 🡺 )

Vincula la biblioteca (indicada con #) al directorio rrp.

El directorio rrp no debe ser HOME.

3D7001 FLASHPTR 001 3D7 ( rrp 🡺 )

Desvincula al directorio rrp de una biblioteca, si rrp está vinculado a alguna.

El directorio rrp no debe ser HOME.

16.1.4 Más Comandos que Manejan Bibliotecas

**Direcc. Nombre Descripción**

014002 ^LIBS\_ ( 🡺 {} )

Pone en la pila una lista con todas las bibliotecas vinculadas con el directorio actual y con los directorios padre de este. Para cada biblioteca, devuelve su nombre, su número y su número de puerto como enteros.

{“nomb1” Z\_id Z\_puert “nomb2” Z\_id Z\_puert …}

También coloca las bibliotecas con nombre nulo.

Equivale al comando LIBS de User RPL.

015002 ^GETLIBS\_ ( 🡺 {} )

Pone en la pila una lista de listas con todas las bibliotecas vinculadas con el directorio actual y con los directorios padre de este. Para cada biblioteca, devuelve su nombre, y su número como bint.

{ { “nomb1” #lib1 } { “nomb2” #lib2 } ... }

No coloca las bibliotecas con nombre nulo.

07F86 (ROMPART) ( rrp 🡺 {#lib1 #lib2 …} T )

( rrp 🡺 #lib T )

( rrp 🡺 F )

( ROMPTR 🡺 #lib T )

( ROMPTR 🡺 F )

Si rrp es el directorio HOME retorna una lista con los números de las bibliotecas vinculadas con HOME en orden decreciente, incluyendo a las bibliotecas ya incorporadas en la ROM (aquellas que tienen #lib < #100 ).

Si rrp no es el directorio HOME, retorna el número de la bilioteca vinculada con el directorio y TRUE.

Si ninguna biblioteca está vinculada con rrp, retorna FALSE.

Si en la pila se encuentra un rompointer, retorna el número de la biblioteca de este rompointer.

081DE (LIB>#) ( lib 🡺 #lib )

Retorna el número correspondiente a la biblioteca del nivel uno de la pila.

080DA (NEXTROMPID) ( #lib 🡺 #lib T )

( #lib 🡺 #nextlib T )

( #lib 🡺 F )

Si la biblioteca especificada existe, agrega TRUE.

Si no existe, agrega el número de la siguiente biblioteca y TRUE.

08199 (ROMPARTNAME) ( #lib 🡺 id T )

( #lib 🡺 F )

Retorna el nombre de la biblioteca como un id y TRUE, si esta existe.

080BF (ROMPARTSIZE) ( #lib 🡺 #nibbles-10 T )

( #lib 🡺 #nextlib T )

( #lib 🡺 F )

Retorna el tamaño de la biblioteca en nibbles menos diez.

**Direcc. Nombre Descripción**

08157 (GETCONFIG) ( #libnum 🡺 ob T )

( #libnum 🡺 F )

Retorna el objeto de configuración de la biblioteca y TRUE, si la biblioteca existe.

08112 (GETHASH) ( #libnum 🡺 hxs\_table T ) (si está en puerto 0)

( #libnum 🡺 AccessPtr T ) (si está en puerto 1 ó 2)

( #libnum 🡺 F )

Gets specified library’s hash table.

07638 SETHASH ( #libnum hxs 🡺 )

0813C (GETLINK) ( #libnum 🡺 hxs\_table T ) (si está en puerto 0)

( #libnum 🡺 AccessPtr T ) (si está en puerto 1 ó 2)

( #libnum 🡺 F )

Gets specified library’s link table.

08130 (GETMSG) ( #libnum 🡺 ArryOfStrings T ) (si está en puerto 0)

( #libnum 🡺 AccessPtr T ) (si está en puerto 1 ó 2)

( #libnum 🡺 F )

Gets specified library’s message table.

0764E (SETMSG) ( ArryOfStrings #libnum 🡺 )

Sets message table of specified library.

265DA (GetLibExt) ( #lib 🡺 flag )

Retorna TRUE si la biblioteca está vinculada y tiene un message handler.

De lo contrario, sólo retorna FALSE.

25F2E (ExecGetLibsExtentions\_sup) ( lista #msg 🡺 lista' )

( meta #msg 🡺 meta' )

Llama a los message handlers de todas las bibliotecas vinculadas con el mensaje #msg. Tenga en cuenta que el mensaje de la biblioteca.

822B2 (ECRAN) ( #lib 🡺 #lib )

( romptr F #9 🡺 romptr T/F #9 )

( romptr 10 🡺 F )

Si en la pila se encuentra el número de una biblioteca llama al mensaje #lib del message handler de la biblioteca.

De otro modo, llama al mensaje 9 o 10 del message handler de la biblioteca correspondiente al rompointer del nivel 3.

Este comando se usa junto con el comando GetLibExt\_.

Por ejemplo, el siguiente programa ( #lib 🡪 ) ejecuta el mensaje de menú de biblioteca del message handler de la biblioteca cuyo número se encuentra en el nivel 1 de la pila:

:: DUP GetLibExt\_ IT ECRAN DROP ;

2F2C6 (XEQXRCL) ( :%puerto:%libnum 🡺 lib )

Pone en la pila a una biblioteca que se encuentra en el puerto indicado con la etiqueta.

La etiqueta puede ser también & para buscar en todos los puertos.

En el capítulo 25 se pueden ver otros usos de este comando.

16.1.4 Objetos de Respaldo (Backup)

**Direcc. Nombre Descripción**

081D9 BAKNAME ( bak 🡺 id T )

Retorna el nombre de un backup.

0905F BAK>OB ( bak 🡺 ob )

Retorna el objeto contenido en el backup.

Generalmente es un directorio.

2F255 ( id ob 🡺 bak )

Crea un objeto de respaldo a partir del objeto del nivel 1 de la pila y cuyo nombre será el especificado en el nivel 2 de la pila.

09075 ( id 🡺 bak )

Crea un objeto de respaldo cuyo contenido es una copia del directorio actual y cuyo nombre será el especificado en la pila.

00D002 FLASHPTR 002 00D ( :n:id 🡺 bak T )

( :n:id 🡺 :n:id F )

Si el objeto existe en los puertos 0, 1 ó 2, lo convierte a backup y retorna TRUE.

16.2 Ejemplos

Ejemplo 1 Bibliotecas

**Número de comandos visibles de una biblioteca.**

El siguiente NULLNAME retorna el número de comandos visibles de una biblioteca.

NULLNAME #Lib>#CmdVis ( #lib -> #NumeroDeComandosVisibles )

:: ( #lib )

GETHASH\_ ( AccessPtr T // F )

NOTcase

BINT0 ( sale con #0 )

( AccessPtr )

ASSEMBLE

CON(5) #2636E

RPL

( #NumeroDeComandosVisibles )

;

Ejemplo 2 Bibliotecas

**Número de comandos de una biblioteca.**

El siguiente NULLNAME retorna el número de comandos de una biblioteca.

\* Retorna el número de comandos de una biblioteca

\* La biblioteca debe de estar guardada en el puerto 0.

NULLNAME #Lib>#Cmd ( #lib -> #NumCmd )

:: ( #lib )

GETLINK\_ ( #lib hxs\_table T // #lib F )

NOTcase

ZERO

( #lib hxs\_table )

LENHXS

BINT5

#/

SWAPDROP ( #NumeroDeComandos )

;

Capítulo 17  
Datos de Biblioteca (Library Data)

Cuando creas una biblioteca con algún programa que guarde archivos para su uso posterior, puedes guardar ese archivo como un Library Data. Los Library Data no pueden ser manipulados por el usuario, de manera que no es fácil alterar su contenido con User RPL.

La estructura de un Library Data es la siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| Prólogo | DOEXT0 # 2B88 | 5 | “88B20” |
| Cuerpo | Tamaño del cuerpo | 5 |  |
| Número de biblioteca | 5 |  |
| Objetos (con sus prólogos) |  |  |
| SEMI | 5 | “B2130” |

Cualquier objeto u objetos pueden ser convertidos a Library Data. Además, un Library Data debe contener al número de la biblioteca respectiva.

17.1 Ejemplos

Ejemplo 1 Library Data

**Crear un Library Data.**

Para crear un objeto Library Data puedes usar la siguiente subrutina. El argumento debe ser un compuesto cuyo primer elemento sea el número de la biblioteca como un bint y el resto de los elementos deben ser los objetos que contendrá el Library Data.

NULLNAME >LibraryData ( :: #lib ob1...obn ; -> LibraryData )

:: ( :: #lib ob1...obn ; )

FLASHPTR 002 0A5 ( "D9D2011920#####...B2130" ) ( comando ->H de L256 )

BINT11 ( "D9D2011920#####...B2130" 11 )

LAST$ ( "#####...B2130" )

"88B20" ( "#####...B2130" "88B20" )

OVERLEN$ ( "#####...B2130" "88B20" #tamaño-5 )

#5+ ( "#####...B2130" "88B20" #tamaño )

#>HXS ( "#####...B2130" "88B20" HXStamaño )

FLASHPTR 002 0A7 ( "#####...B2130" "88B20" "Tamañ" ) ( comando A->H de L256 )

&$ ( "#####...B2130" "88B20Tamañ" )

SWAP&$ ( "88B20Tamañ#####...B2130" )

FLASHPTR 002 0A4 ( LibraryData ) ( comando H-> de L256 )

;

También puedes usar la siguiente alternativa más rápida, con una parte escrita en lenguaje ensamblador (extraído del comando ROMPTR 0E4 011, que es llamado por los comandos del entorno MSOLVR, Multiple Solver Equation).

NULLNAME >LibraryData ( :: #lib ob1...obn ; -> LibraryData )

:: ( :: #lib ob1...obn ; )

TOTEMPOB ( :: #lib ob1...obn ; )

DUP ( :: #lib ob1...obn ; :: #lib ob1...obn ; )

OSIZE ( :: #lib ob1...obn ; #tamaño\_bint+objeto+10 ) ( tamaño del LibData )

CODE

NIBHEX 8F1466081AF018FB97601431313488B2014517481AF19818FA41458F2D760142164808C

ENDCODE

( LibraryData )

;

Ejemplo 2 Library Data

**Desarmar un Library Data.**

Con el siguiente código puedes hacer la operación inversa. Dado un Library Data en la pila conseguirás el contenido del Library Data y su número de biblioteca.

NULLNAME LibraryData> ( LibraryData -> :: #lib ob1...obn ; )

:: ( LibraryData )

FLASHPTR 002 0A5 ( "88B20Tamañ#####...B2130" ) ( comando ->H de L256 )

BINT11 ( "88B20Tamañ#####...B2130" 11 )

LAST$ ( "#####...B2130" )

"D9D2011920" ( "#####...B2130" "D9D2011920" )

SWAP&$ ( "D9D2011920#####...B2130" )

FLASHPTR 002 0A4 ( :: #lib ob1...obn ; ) ( comando H-> de L256 )

;

También puedes usar la siguiente alternativa más rápida.

NULLNAME LibraryData> ( LibraryData -> :: #lib ob1...obn ; )

:: ( LibraryData )

TOTEMPOB ( LibraryData' )

CODE

NIBHEX 14713780824D9D2014117480824119201411358D94150

ENDCODE

( :: #lib ob1...obn ; )

;

Ejemplo 3 Library Data

**Hallar el número de biblioteca de un Library Data.**

Para retornar sólo el número de biblioteca del LibraryData, puedes usar cualquiera de los siguientes 3 NULLNAME. El primero toma más tiempo y el último es el más rápido.

NULLNAME LibraryData># ( LibraryData -> #lib )

:: ( LibraryData )

FLASHPTR 002 0A5 ( "88B20Tamañ#####...B2130" ) ( comando ->H de L256 )

BINT11 ( "88B20Tamañ#####...B2130" BINT11 )

BINT15 ( "88B20Tamañ#####...B2130" BINT11 BINT15 )

SUB$ ( "#####" )

"11920" ( "#####" "11920" )

SWAP&$ ( "11920#####" )

FLASHPTR 002 0A4 ( #lib ) ( comando H-> de L256 )

;

NULLNAME LibraryData># ( LibraryData -> #lib )

:: ( LibraryData )

FLASHPTR 002 0A5 ( "88B20Tamañ#####...B2130" ) ( comando ->H de L256 )

BINT11 ( "88B20Tamañ#####...B2130" BINT11 )

BINT15 ( "88B20Tamañ#####...B2130" BINT11 BINT15 )

SUB$ ( "#####" )

FLASHPTR 002 0A8 ( hxslib ) ( comando H->A de L256 )

HXS># ( #lib )

;

NULLNAME LibraryData># ( LibraryData -> #lib )

:: ( LibraryData )

TOTEMPOB ( LibraryData' )

CODE

NIBHEX 14713780824D9D2014117480824119201411358D94150

ENDCODE

( :: #lib ob1...obn ; )

CARCOMP ( #lib )

;

Ejemplo 4 Library Data

**Retornando una lista con todos los nombres globales del directorio actual que contengan a un Library Data cuyo número de biblioteca sea el que se indica.**

El siguiente NULLNAME retorna una lista con todos los nombres globales del directorio actual que contengan a un Library Data cuyo número de biblioteca sea el que se indica.

NULLNAME #>{ID}LibData ( #lib -> {} )

:: ( #lib )

%26 DOTVARS% ( #lib {} ) ( lista con todos los LibData del directorio actual )

DUPNULL{}? ( #lib {} flag )

case SWAPDROP ( OBS: SALE CON LISTA VACIA )

( #lib {} )

DUP >R ( #lib {} )

LENCOMP ( #lib #n )

NULL{} ( #lib #n {} )

SWAP ( #lib {} #n )

ZERO\_DO (DO) ( #lib {} )

RSWAP 'R RSWAP ( #lib {} IDi )

DUPSAFE@ ( #lib {} IDi LibraryData T )

DROP ( #lib {} IDi LibraryData )

LibraryData># ( #lib {} IDi #libi ) ( LibraryData># está arriba )

4PICK ( #lib {} IDi #libi #lib )

EQUAL ( #lib {} IDi flag )

ITE

>TCOMP ( #lib {} )

DROP ( #lib {} )

LOOP ( #lib {} )

SWAPDROP ( {} )

ROMPTR 0E8 016 ( {} ) ( OPCIONAL: Orden alfabético )

;

Capítulo 18  
Operaciones con la Pila

En System RPL, el uso de la pila es casi el mismo que en User RPL.

Las operaciones básicas son las mismas, excepto por pequeños cambios en los nombres: DUP, 2DUP (equivalente al comando de User RPL **DUP2**), NDUP (**DUPN**), DROP, 2DROP (**DROP2**), NDROP (**DROPN**), OVER, PICK, SWAP, ROLL, UNROLL (**ROLLD**), ROT, UNROT y DEPTH.

Todos los comandos que requieran o retornen un argumento numérico usan bints y no números reales, a menos que se indique lo contrario.

Hay muchos comandos que hacen dos o tres operaciones una después de otra. Estos comandos son listados en la sección de referencia de abajo.

La siguiente tabla muestra algunas combinaciones útiles en una forma resumida:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ⮦ | **DUP** | **DROP** | **SWAP** | **OVER** | **ROT** | **UNROT** |
| **DUP** | DUPDUP | DROPDUP | SWAPDUP | OVERDUP | ROTDUP | UNROTDUP |
| **DROP** | - | 2DROP | SWAPDROP | - | ROTDROP | UNROTDROP |
| **SWAP** | - | DROPSWAP | - | OVERSWAP | ROTSWAP | UNROTSWAP |
| **OVER** | DUPDUP | DROPOVER | SWAPOVER | 2DUP | ROTOVER | UNROTOVER |
| **ROT** | DUPROT | DROPROT | SWAPROT | - | UNROT | - |
| **UNROT** | DUPUNROT | - | ROTSWAP | OVERUNROT | - | ROT |
| **SWAPDROP** | - | DROPSWAPDROP | - | DROPDUP | DROPSWAP | UNROTSWAPDROP |
| **DROPDUP** | - | - | SWAPDROPDUP | - | - | - |
| **DROPSWAP** | - | - | SWAPDROPSWAP | - | ROTDROPSWAP | SWAPDROP |
| **2DROP** | - | 3DROP | - | - | ROT2DROP | UNROT2DROP |
| **2DUP** | - | - | SWAP2DUP | - | ROT2DUP | - |
| **3PICK** | DUP3PICK | - | SWAP3PICK | OVERDUP | - | - |
| **4PICK** | - | - | SWAP4PICK | - | - | - |
| **5PICK** | - | - | - | OVER5PICK | - | - |
| **4ROLL** | - | - | SWAP4ROLL | - | - | - |
| **4UNROLL** | DUP4UNROLL | - | - | - | - | - |
| **ROT2DROP** | - | - | ROTROT2DROP | SWAPDROP | ROTROT2DROP | - |

18.1 Referencia

En esta sección, los números 1, 2. . . n son usados para representar objetos diferentes, no necesariamente algun tipo de número.

**Direcc. Nombre Descripción**

03188 DUP ( ob è ob ob )

35CE0 DUPDUP ( ob è ob ob ob )

2D5006 ˆ3DUP ( 3 2 1 è 3 2 1 3 2 1 )

28143 NDUPN ( ob #n è ob..ob #n )

( ob #0 è #0 )

35FF3 DUPROT ( 1 2 è 2 2 1 )

3457F DUPUNROT ( 1 2 è 2 1 2 )

aka: SWAPOVER

36133 DUPROLL ( 1..n #n è 1 3..n #n 2 )

3432C DUP4UNROLL ( 1 2 3 è 3 1 2 3 )

3611F DUPPICK ( n..1 #n è n..1 #n n-1 )

35D30 DUP3PICK ( 1 2 è 1 2 2 1 )

aka: 2DUPSWAP

34431 DUP#1+PICK ( n..1 #n è n..1 #n n )

031AC 2DUP ( 1 2 è 1 2 1 2 )

35D30 2DUPSWAP ( 1 2 è 1 2 2 1 )

aka: DUP3PICK

36CA4 2DUP5ROLL ( 1 2 3 è 2 3 2 3 1 )

031D9 NDUP ( 1..n #n è 1..n 1..n )

03244 DROP ( 1 è )

357CE DROPDUP ( 1 2 è 1 1 )

37032 DROPNDROP ( 1..n #n ob è )

35733 DROPSWAP ( 1 2 3 è 2 1 )

3574D DROPSWAPDROP ( 1 2 3 è 2 )

aka: ROT2DROP, XYZ>Y

36007 DROPROT ( 1 2 3 4 è 2 3 1 )

3606B DROPOVER ( 1 2 3 è 1 2 1 )

03258 2DROP ( 1 2 è )

341D2 3DROP ( 1 2 3 è )

aka: XYZ>

341D7 4DROP ( 1..4 è )

aka: XYZW>

341DC 5DROP ( 1..5 è )

341E8 6DROP ( 1..6 è )

341F4 7DROP ( 1..7 è )

0326E NDROP ( 1..n #n è )

35FB0 #1+NDROP ( ob 1..n #n è )

aka: N+1DROP

2F0A1 RESETDEPTH ( ob1..obn obn+1..obx #n è ob1..obn )

Borra los objetos en la pila, excepto los #n primeros.

28335 (KEEP) ( ob1..obn ob1’..obm’ #m è ob1’..obm’ )

Borra los objetos en la pila, excepto los #m últimos.

0314C DEPTH ( 1..n è 1..n #n )

28187 reversym ( 1..n #n è n..1 #n )

03223 SWAP ( 1 2 è 2 1 )

3576E SWAPDUP ( 1 2 è 2 1 1 )

368B5 SWAP2DUP ( 1 2 è 2 1 2 1 )

**Direcc. Nombre Descripción**

3421A SWAPDROP ( 1 2 è 2 )

aka: XY>Y

35857 SWAPDROPDUP ( 1 2 è 2 2 )

35872 SWAPDROPSWAP ( 1 2 3 è 3 1 )

aka: UNROTDROP, XYZ>ZX

341BA SWAPROT ( 1 2 3 è 3 2 1 )

aka: UNROTSWAP, XYZ>ZYX

36C90 SWAP4ROLL ( 1 2 3 4 è 2 4 3 1 )

aka: XYZW>YWZX

3457F SWAPOVER ( 1 2 è 2 1 2 )

aka: DUPUNROT

36CB8 SWAP3PICK ( 1 2 3 è 1 3 2 1 )

35018 2SWAP ( 1 2 3 4 è 3 4 1 2 )

03295 ROT ( 1 2 3 è 2 3 1 )

3579C ROTDUP ( 1 2 3 è 2 3 1 1 )

35CA4 ROT2DUP ( 1 2 3 è 2 3 1 3 1 )

341A8 ROTDROP ( 1 2 3 è 2 3 )

aka: XYZ>YZ

3574D ROT2DROP ( 1 2 3 è 2 )

aka: DROPSWAPDROP, XYZ>Y

34195 ROTDROPSWAP ( 1 2 3 è 3 2 )

aka: XYZ>ZY

3416E ROTSWAP ( 1 2 3 è 2 1 3 )

aka: XYZ>YXZ

343BD ROTROT2DROP ( 1 2 3 è 3 )

aka: UNROT2DROP, XYZ>Z

35CCC ROTOVER ( 1 2 3 è 2 3 1 3 )

3423A 4ROLL ( 1 2 3 4 è 2 3 4 1 )

aka: FOURROLL, XYZW>YZWX

3588B 4ROLLDROP ( 1 2 3 4 è 2 3 4 )

35F06 4ROLLSWAP ( 1 2 3 4 è 2 3 1 4 )

36043 4ROLLROT ( 1 2 3 4 è 2 4 1 3 )

aka: FOURROLLROT

360E3 4ROLLOVER ( 1 2 3 4 è 2 3 4 1 4 )

34257 5ROLL ( 1 2 3 4 5 è 2 3 4 5 1 )

aka: FIVEROLL

358A7 5ROLLDROP ( 1 2 3 4 5 è 2 3 4 5 )

34281 6ROLL ( 1..6 è 2..6 1 )

aka: SIXROLL

342EA 7ROLL ( 1..7 è 2..7 1 )

aka: SEVENROLL

342BB 8ROLL ( 1..8 è 2..8 1 )

aka: EIGHTROLL

03325 ROLL ( 1..n #n è 2..n 1 )

35FC4 ROLLDROP ( 1..n #n è 2..n )

35D80 ROLLSWAP ( 1..n #n è 2..n-1 1 n )

344F2 #1+ROLL ( ob 1..n #n è 1..n ob )

34517 #2+ROLL ( a b 1..n #n è b 1..n a )

2D6006 ˆ#3+ROLL ( obn+3...obn...ob1 #n è obn+2...ob1 obn+3 )

344DD #+ROLL ( 1..n+m #n #m è 2..n+m 1 )

344CB #-ROLL ( 1..n-m #n #m è 2..n-m 1 )

**Direcc. Nombre Descripción**

3422B UNROT ( 1 2 3 è 3 1 2 )

aka: 3UNROLL, XYZ>ZXY

35D1C UNROTDUP ( 1 2 3 è 3 1 2 1 )

35872 UNROTDROP ( 1 2 3 è 3 1 )

aka: SWAPDROPSWAP, XYZ>ZX

343BD UNROT2DROP ( 1 2 3 è 3 )

aka: ROTROT2DROP, XYZ>Z

341BA UNROTSWAP ( 1 2 3 è 3 2 1 )

aka: SWAPROT, XYZ>ZYX

360CF UNROTOVER ( 1 2 3 è 3 1 2 1 )

3422B 3UNROLL ( 1 2 3 è 3 1 2 )

aka: UNROT, XYZ>ZXY

34331 4UNROLL ( 1 2 3 4 è 4 1 2 3 )

aka: FOURUNROLL, XYZW>WXYZ

35D44 4UNROLLDUP ( 1 2 3 4 è 4 1 2 3 3 )

343CF 4UNROLL3DROP ( 1 2 3 4 è 4 )

aka: XYZW>W

36057 4UNROLLROT ( 1 2 3 4 è 4 3 2 1 )

34357 5UNROLL ( 1 2 3 4 5 è 5 1 2 3 4 )

aka: FIVEUNROLL

3438D 6UNROLL ( 1..6 è 6 1..5 )

aka: SIXUNROLL

35BEB 7UNROLL ( 1..7 è 7 1..6 )

3615B 8UNROLL ( 1..8 è 8 1..7 )

28225 (9UNROLL) ( 1..9 è 9 1..8 )

3616F 10UNROLL ( 1..10 è 10 1..9 )

0339E UNROLL ( 1..n #n è n 1..n-1 )

34552 #1+UNROLL ( ob 1..n #n è n ob 1..n-1 )

34564 #2+UNROLL ( a b 1..n #n è n a b 1..n-1 )

3453D #+UNROLL ( 1..n+m #n #m è n+m 1..n+m-1 )

3452B #-UNROLL ( 1..n-m #n #m è n-m 1..n+m-1 )

032C2 OVER ( 1 2 è 1 2 1 )

35CF4 OVERDUP ( 1 2 è 1 2 1 1 )

35D6C OVERSWAP ( 1 2 è 1 1 2 )

aka: OVERUNROT

35D6C OVERUNROT ( 1 2 è 1 1 2 )

aka: OVERSWAP

36CF4 OVER5PICK ( 1 2 3 4 è 1 2 3 4 3 1 )

37046 2OVER ( 1 2 3 4 è 1 2 3 4 1 2 )

34485 3PICK ( 1 2 3 è 1 2 3 1 )

35F1A 3PICKSWAP ( 1 2 3 è 1 2 1 3 )

360F7 3PICKOVER ( 1 2 3 è 1 2 3 1 3 )

36CCC 3PICK3PICK ( 1 2 3 è 1 2 3 1 2 )

2F1C6 DROP3PICK ( 1 2 3 4 è 1 2 3 1 )

3448A 4PICK ( 1 2 3 4 è 1 2 3 4 1 )

35F2E 4PICKSWAP ( 1 2 3 4 è 1 2 3 1 4 )

36CE0 SWAP4PICK ( 1 2 3 4 è 1 2 4 3 1 )

3610B 4PICKOVER ( 1 2 3 4 è 1 2 3 4 1 4 )

3448F 5PICK ( 1 2 3 4 5 è 1 2 3 4 5 1 )

34494 6PICK ( 1..6 è 1..6 1 )

34499 7PICK ( 1..7 è 1..7 1 )

3449E 8PICK ( 1..8 è 1..8 1 )

344A3 (9PICK) ( 1..9 è 1..9 1 )

**Direcc. Nombre Descripción**

344A8 (10PICK) ( 1..10 è 1..10 1 )

032E2 PICK ( 1..n #n è 1..n 1 )

34436 #1+PICK ( 1..n #n-1 è 1..n 1 )

34451 #2+PICK ( 1..n #n-2 è 1..n 1 )

34465 #3+PICK ( 1..n #n-3 è 1..n 1 )

34474 #4+PICK ( 1..n #n-4 è 1..n 1 )

34417 #+PICK ( 1..n+m #n #m è 1..n+m 1 )

34405 #-PICK ( 1..n-m #n #m è 1..n-m 1 )

Capítulo 19  
Entornos Temporales

Las variables locales (también conocidas como variables temporales o variables lambda) funcionan de la misma manera y tienen los mismos usos que en User RPL. Tu les asignas valores a estas variables y estos valores pueden ser llamados o cambiados mientras estas variables existan. Los valores guardados son referenciados por medio de identificadores locales (también llamados identificadores lambda, o lams). Estos son muy similares a los identificadores globales que referencian a variables guardadas en la memoria (ver Capítulo 8), pero las variables existen sólo temporalmente.

Pero hay una diferencia: en System RPL puedes dar un nombre nulo (es decir, vacío) a las variables locales, por lo tanto estas son variables sin nombre. Esto ahorra memoria y es mucho más rápido. Pero antes de aprender como crear y usar variables sin nombre, aprenderemos como usar las variables normales, es decir, las variables con nombre.

19.1 Variables locales con nombre

Crear variables locales con nombre es muy similar a crear variables temporales en User RPL. Tu tienes que crear una lista de identificadores locales (llamados lams) y ejecutar el comando BIND. Para llamar el contenido de uno de estos, solo escribe su identificador local. Para guardar un nuevo valor en una variable local, pon ese valor y el lam en la pila, y ejecuta el comando STO. Para remover las variables locales de la memoria, usa el comando ABND. El código no verifica si encuentran las combinaciones BIND/ABND, de manera que puedes escribir cada uno de estos dos comandos en diferentes programas si lo deseas. Pero esto también significa que debes estar seguro de tener un ABND para cada BIND.

A continuación un pequeño programa que crea dos variables locales, llama a sus contenidos y les asigna nuevos valores a estos:

::

%12 %13

{ LAM first LAM sec } BIND ( ) ( crea nuevo entorno temporal )

( ) ( first contiene 12, y sec contiene 13 )

LAM first ( %12 ) ( llama al contenido de first: 12 )

LAM sec ( %13 ) ( llama al contenido de sec: 13 )

%+ ( %25 )

' LAM first ( %25 lam )

STO ( ) ( guarda esa suma en first )

ABND ( ) ( borra las variables de la memoria creadas por ABND )

;

19.2 Variables locales sin nombre

Como se dijo arriba, tu puedes usar variables locales sin nombre. Técnicamente, estas tienen un nombre: el nombre nulo o vacío; pero todas las variables “sin nombre” tienen el mismo nombre. Como estas variables no pueden ser identificadas por nombre, estas se diferencian en su posición. El programa de arriba puede ser escrito usando variables temporales sin nombre de la siguiente manera:

::

%12 %13

{ NULLLAM NULLLAM } BIND ( ) ( crea nuevo entorno temporal )

( ) ( LAM1 contiene a 13, LAM2 contiene a 12 )

2GETLAM ( %12 ) ( llama al contenido de LAM2: 12 )

1GETLAM ( %13 ) ( llama al contenido de LAM1: 13 )

%+ ( %25 )

2PUTLAM ( ) ( guarda esa suma en LAM2 )

ABND ( borra las variables de la memoria creadas por ABND )

;

La numeración de los lams es hecha en el mismo orden que los niveles que tenían en la pila: 1GETLAM contiene al objeto que estaba en el nivel uno, 2GETLAM contiene al objeto que estaba en el nivel dos, etc.

Hay comandos para llamar y guardar directamente hasta la variable número 27 (usar 1GETLAM a 27GETLAM\_, para llamarlas y usar 1PUTLAM hasta 27PUTLAM\_ para cambiar su valor). Para acceder a variables con números mayores a 27 (lo cual probablemente no ocurrirá con frecuencia) usar GETLAM, el cual toma como argumento a un bint que representa el número de la variable y retorna su contenido; y PUTLAM, el cual toma como argumentos a un objeto y al número de la variable y guarda ese objeto en la variable especificada.

19.3 Entornos Temporales Anidados

Es posible usar dos o más entornos temporales al mismo tiempo. Nada especial se necesita hacer durante su creación: sólo usar otro BIND antes de abandonar el previo entorno temporal. Cuando un ABND es encontrado, este siempre se refiere al más reciente BIND.

Si tu solamente usas lams con nombre, nada especial es necesario hacer. No habrá confusion con los nombres, a menos que redefinas una variable existente (pero hacer esto, sólo hará que tu programa sea enredado).

Sin embargo, cuando por lo menos uno de los entornos temporales tiene variables sin nombre, debes prestar atención a la numeración de los lams.

Notar que los comandos GETLAM se refieren tanto a variables sin nombre como a variables con nombre: 1GETLAM llama a la variable local creada más recientemente, 2GETLAM a la anterior a esa, y así sucesivamente. (cuando creas lams, la creación de estos comienza en el nivel de la pila con el número mayor, hacia aquel con el número más pequeño, de modo que la última variable creada sea aquella cuyo contenido está en el nivel uno).

Tu puedes usar los comandos GETLAM también para acceder a lams con nombre. Debido a a la naturaleza de los entornos temporales, aparecerá una variable local extra (antes de las otras) para propósitos de uso interno. Para acceder a lams sin nombre de un entorno previo, tu debes agregar el número de variables creadas en el entorno actual más uno al número que habrías usado antes del último BIND.

El siguiente programa intentará hacer más clara la explicación de arriba:

::

%102

%101

{ LAM n2 LAM n1 } BIND

1GETLAM ( Retorna 101 )

2GETLAM ( Retorna 102 )

%104

%103

{ NULLLAM NULLLAM } BIND

1GETLAM ( Retorna 103 )

2GETLAM ( Retorna 104 )

4GETLAM ( Retorna 101 )

5GETLAM ( Retorna 102 )

ABND

ABND

;

Primero, este programa asigna 102 a n2 y 101 a n1, pero estos nombres no son usados después en el programa. En su lugar, 1GETLAM es usado para acceder al valor del lam creado más recientemente, esto es, 101, al cual también pudo accederse con LAM n1. Luego, 2GETLAM retorna el valor del siguiente lam, o sea 102.

Las cosas se vuelven más complicadas cuando otro entorno es creado, esta vez con lams sin nombre. Ahora 1GETLAM retorna 103, el cual pertenece al nuevo entorno y fue la última variable creada. De igual manera, 2GETLAM también retorna una variable creada en el entorno más reciente. Si nosotros queremos acceder al valor del lam que antes tenía el número uno, necesitamos agregar el número de variables creadas en el nuevo entorno (es decir, dos) más uno (la pseudo-variable de uso interno) al número previo.

Por lo tanto, para conseguir el valor del lam 1 del entorno previo, debemos agregar tres a uno, obteniendo 4GETLAM. Y esto retorna, como se esperaba a 101. Del mismo modo, 5GETLAM retorna 102, lo mismo que 2GETLAM retornaba antes del segundo BIND.

Naturalmente, después del primer ABND (correspondiente a la creación de los lams que contenían los valores 104 y 103), 1GETLAM y 2GETLAM nuevamente retornarán 101 y 102, respectivamente.

Si has podido comprender lo explicado, no tendrás problemas para anidar entornos temporales cuando sea necesario.

19.4 Otras Formas de Crear LAMS

Primero, en vez de una lista de lams, puedes siempre poner cada lam en la pila, luego el número de lams que se crearán, y ejecutar el comando DOBIND en lugar de BIND. En realidad, BIND es equivalente a :: INNERCOMP DOBIND ;.

Sin deseas crear un gran número de variables sin nombre (por ejemplo, 24 variables), en lugar de ingresar el siguiente código (el cual toma 67.5 bytes)

...

{ NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM

NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM

NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM

NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM }

BIND

...

usa este, el cual toma sólo15 bytes, ahorrando 52.5 bytes:

... ' NULLLAM BINT24 NDUPN {}N BIND ...

Sin embargo, ¿para que crear un compuesto si este sera disgregado más tarde? Reemplaza {}N BIND por DOBIND, y ahorra 2.5 bytes más.

O también puedes usar BINT24 ' NULLLAM CACHE. Sin embargo, si usas esto, dos variables adicionales son creadas (en lugar de una), de manera que debes agregar uno a las posiciones de las variables de los ejemplos previos.

Cuando descompilas código, a veces puedes encontrar cosas como esta

... ZEROZEROZERO BINT3 DOBIND ...

La cual es otra manera de crear variables locales sin nombre. Esto funciona porque en lugar de NULLLAM, cualquier objeto de la ROM con dirección fija puede usarse, como ZERO en este ejemplo.

Lo mostrado a continuación es la manera más compacta de crear entornos temporales para N variables locales sin nombre.

|  |  |
| --- | --- |
| **N** | **Comandos para crear N variables locales sin nombre** |
| 1 | 1LAMBIND |
| 2 | ZEROZEROTWO DOBIND |
| 2 | FLASHPTR 2LAMBIND |
| 3 | FLASHPTR 3LAMBIND |
| 3 | 3NULLLAM{}\_ BIND |
| 4 | 4NULLLAM{} BIND |
| N | ' NULLLAM #N NDUPN DOBIND |

19.5 Viendo las Variables Locales con Debug4x

En Debug4x, puedes ver el contenido de todos los entornos temporales en cualquier momento en el Panel “Local Variables” que se encuentra en la ventana RPL Debugging.

Por ejemplo, si escribimos el código de la izquierda en el editor de Debug4x, fijamos un breakpoint (círculo rojo) en la posición indicada y ejecutamos el comando EjTempEnv desde el emulador, podremos ver en el Panel Local Variables, todos los entornos temporales creados y sus respectivos “protection word”.

Además de los entornos temporales que hemos creado, hay un entorno temporal creado por la calculadora y contiene los objetos que estaban presentes en la pila (y el contador) antes de ejecutar el programa EjTempEnv.

|  |  |
| --- | --- |
| Ventana Editor | Ventana RPL Debugging |
|  |  |

xNAME EjTempEnv

::

CK0

31. 32. 33.

{ LAM A LAM B LAM C } BIND

21. 22. FLASHPTR 2LAMBIND

11. 12. 13.

{ LAM A LAM X LAM Y } BIND

' LAM A ( lam )

@LAM ( %11 T )

ABND

ABND

ABND

;

19.6 Referencia

19.6.1 IDs y LAMs ya Incorporados en ROM

**Direcc. Nombre Descripción**

272FE NULLID ( 🡺 id )

Pone identificador nulo (vacío) sin evaluar en la pila.

2B3AB NULLLAM ( 🡺 ob )

( 🡺 lam )

Retorna el contenido de la variable local sin nombre creada más recientemente. Si no hay ninguna, retorna el lam.

Para retornar siempre una variable local sin nombre en la pila, debes escribir: ' NULLLAM

27155 'IDX ( 🡺 id )

Pone ID X sin evaluar en la pila.

27AE9 ('IDPAR) ( 🡺 id )

Pone ID PPAR sin evaluar en la pila.

27B25 ('IDTPAR) ( 🡺 id )

Pone ID TPAR sin evaluar en la pila.

27B07 ('IDVPAR) ( 🡺 id )

Pone ID VPAR sin evaluar en la pila.

271D3 (IDSTARTERR) ( 🡺 {} )

Pone { ID STARTERR } sin evaluar en la pila.

27308 (EvalNULLID) Evalúa el id nulo.

Si en el directorio actual hay una variable global de nombre nulo, evalúa su contenido.

De lo contrario, fija al directorio oculto como el nuevo directorio actual.

2715F (ID\_X) ID X

272F3 (CUREQ) ID EQ

27937 (ID\_SIGMADAT) ID ∑DAT

3EA01 (ID\_CST) ID CST

2798A (ID\_FV) ID FV

27963 (ID\_I%YR) ID I%YR

2795A (ID\_N) ID N

2797D (ID\_PMT) ID PMT

2799A (ID\_PYR) ID PYR

27972 (ID\_PV) ID PV

27946 (ID\_SIGMAPAR) ID ∑PAR

271D8 (ID\_STARTERR) ID STARTERR

271B9 (ID\_STARTUP) ID STARTUP

3EF97 (ID\_S) ID S

27B2F (ID\_TPAR) ID TPAR

27B11 (ID\_VPAR) ID VPAR

271A3 (IDIOPAR) ID IOPAR

19.6.2 Conversión

**Direcc. Nombre Descripción**

05B15 $>ID ( $ 🡺 ID )

362DE DUP$>ID ( $ 🡺 $ ID )

05AED (ID>LAM) ( ID 🡺 LAM )

05B01 (LAM>ID) ( LAM 🡺 ID )

19.6.3 Entornos Temporales

**Direcc. Nombre Descripción**

074D0 BIND ( obn..ob1 {lamn..lam1} 🡺 )

Crea n lams.

074E4 DOBIND ( obn..ob1 lamn..lam1 #n 🡺 )

Crea n lams.

36518 1LAMBIND ( ob 🡺 )

Crea un lam sin nombre.

Equivale a hacer: { NULLLAM } BIND

36513 DUP1LAMBIND ( ob 🡺 ob )

Hace DUP luego 1LAMBIND.

155006 ˆ2LAMBIND ( ob1 ob2 🡺 )

Crea dos lams sin nombre.

Equivale a hacer: { NULLLAM NULLLAM } BIND

156006 ˆ3LAMBIND ( ob1 ob2 ob3 🡺 )

Crea tres lams sin nombre.

Equivale a hacer: { NULLLAM NULLLAM NULLLAM } BIND

0DE0B0 ˜nNullBind ( obn..ob1 #n 🡺 )

Crea n variables sin nombre. 1LAM tiene al contador, 2LAM al primer objeto, 3LAM al segundo objeto, etc.

Equivale a usar ' NULLLAM CACHE

36A77 dvarlsBIND ( ob 🡺 )

Crea la variable local LAM 'dvar.

Equivale a usar { LAM 'dvar } BIND

07497 ABND ( 🡺 )

Abandona el entorno temporal creado más recientemente.

34D00 CACHE ( obn...ob1 #n lam 🡺 )

Crea n+1 variables todas con el mismo nombre. 1LAM tiene al contador, 2LAM al primer objeto, 3LAM al segundo objeto, etc.

34EBE DUMP ( NULLLAM 🡺 ob1..obn #n )

Inversa de CACHE, cuando el entorno temporal se creó con variables sin nombre.

Siempre hace recolección de basura.

No abandona el entorno temporal.

34D58 SAVESTACK ( obn...ob1 🡺 )

( 🡺 )

Guarda todos los objetos de la pila y el número de ellos en un entorno temporal. 1LAM tiene al contador, 2LAM al objeto del nivel uno, 3LAM al del nivel dos, etc.

34FA6 undo ( obm’...ob1’ 🡺 obn...ob1 )

( 🡺 )

Borra todos los objetos de la pila, y coloca en esta los objetos del entorno temporal creados por SAVESTACK, excepto al contador.

No abandona el entorno temporal.

**Direcc. Nombre Descripción**

07943 @LAM ( lam 🡺 ob T )

( lam 🡺 F )

Intenta llamar al contenido del lam.

Si el lam existe, entonces retorna su valor y TRUE.

De lo contrario, sólo retorna FALSE.

Nota: si el lam existe en varios entornos, llama al de un entorno más reciente.

02FD6 (DoLam) ( lam 🡺 ob )

Intenta llamar al contenido del lam. Si no existe ese lam en ningún entorno, genera el error "Nombre local indefin."

078F5 (NTH@LAM) ( lam #n 🡺 ob T )

( lam #n 🡺 F )

Intenta llamar al contenido del lam desde cualquier entorno.

#n es el número del entorno temporal.

#1 para el entorno más reciente,

#2 para el entorno creado antes del primero, etc.

Si el lam existe en el entorno especificado, entonces retorna su valor y TRUE.

De lo contrario, sólo retorna FALSE.

Por ejemplo, el siguiente programa retorna: %31 TRUE

::

31. 32. 33. { LAM A LAM B LAM C } BIND

21. 22. { LAM D LAM E } BIND

11. 12. 13. { LAM A LAM X LAM Y } BIND

' LAM A ( lam )

BINT3 ( lam #3 )

NTH@LAM\_ ( %31 T ) ( retorna contenido del )

( lam A del entorno 3 )

ABND

ABND

ABND

;

078E9 (FIRST@LAM) ( lam 🡺 ob T )

( lam 🡺 F )

Intenta llamar al contenido del lam sólo desde el entorno temporal más reciente. No busca en otros entornos.

Por ejemplo, el siguiente programa retorna: %21 TRUE

::

31. 32. 33. { LAM A LAM B LAM C } BIND

21. 22. { LAM D LAM E } BIND

' LAM A ( lam )

FIRST@LAM\_ ( F ) ( busca LAM A en entorno 1 )

DROP ( )

' LAM D ( lam )

FIRST@LAM\_ ( %21 T ) ( busca LAM D en entorno 1 )

ABND

ABND

;

07D1B STOLAM ( ob lam 🡺 )

Guarda el objeto en el lam.

Si el lam no existe, genera el error “Nombre local indefinido”

075A5 GETLAM ( #n 🡺 ob )

Retorna el contenido del lam de posición n.

34616 1GETLAM ( 🡺 ob )

34620 2GETLAM ( 🡺 ob )

**Direcc. Nombre Descripción**

3462A 3GETLAM ( 🡺 ob )

34634 4GETLAM ( 🡺 ob )

3463E 5GETLAM ( 🡺 ob )

34648 6GETLAM ( 🡺 ob )

34652 7GETLAM ( 🡺 ob )

3465C 8GETLAM ( 🡺 ob )

34666 9GETLAM ( 🡺 ob )

34670 10GETLAM ( 🡺 ob )

3467A 11GETLAM ( 🡺 ob )

34684 12GETLAM ( 🡺 ob )

3468E 13GETLAM ( 🡺 ob )

34698 14GETLAM ( 🡺 ob )

346A2 15GETLAM ( 🡺 ob )

346AC 16GETLAM ( 🡺 ob )

346B6 17GETLAM ( 🡺 ob )

346C0 18GETLAM ( 🡺 ob )

346CA 19GETLAM ( 🡺 ob )

346D4 20GETLAM ( 🡺 ob )

346DE 21GETLAM ( 🡺 ob )

346E8 22GETLAM ( 🡺 ob )

346F2 (23GETLAM) ( 🡺 ob )

346FC (24GETLAM) ( 🡺 ob )

34706 (25GETLAM) ( 🡺 ob )

34710 (26GETLAM) ( 🡺 ob )

3471A (27GETLAM) ( 🡺 ob )

075E9 PUTLAM ( ob #n 🡺 )

Guarda un nuevo contenido en el lam de orden n.

34611 1PUTLAM ( ob 🡺 )

3461B 2PUTLAM ( ob 🡺 )

34625 3PUTLAM ( ob 🡺 )

3462F 4PUTLAM ( ob 🡺 )

34639 5PUTLAM ( ob 🡺 )

34643 6PUTLAM ( ob 🡺 )

3464D 7PUTLAM ( ob 🡺 )

34657 8PUTLAM ( ob 🡺 )

34661 9PUTLAM ( ob 🡺 )

3466B 10PUTLAM ( ob 🡺 )

34675 11PUTLAM ( ob 🡺 )

3467F 12PUTLAM ( ob 🡺 )

34689 13PUTLAM ( ob 🡺 )

34693 14PUTLAM ( ob 🡺 )

3469D 15PUTLAM ( ob 🡺 )

346A7 16PUTLAM ( ob 🡺 )

346B1 17PUTLAM ( ob 🡺 )

346BB 18PUTLAM ( ob 🡺 )

346C5 19PUTLAM ( ob 🡺 )

346CF 20PUTLAM ( ob 🡺 )

346D9 21PUTLAM ( ob 🡺 )

346E3 22PUTLAM ( ob 🡺 )

346ED (23PUTLAM) ( ob 🡺 )

346F7 (24PUTLAM) ( ob 🡺 )

**Direcc. Nombre Descripción**

34701 (25PUTLAM) ( ob 🡺 )

3470B (26PUTLAM) ( ob 🡺 )

34715 (27PUTLAM) ( ob 🡺 )

3471F (DUP1PUTLAM) ( ob 🡺 ob )

Hace DUP luego 1PUTLAM.

34729 (DUP2PUTLAM) ( ob 🡺 ob )

Hace DUP luego 2PUTLAM.

34797 DUP4PUTLAM ( ob 🡺 ob )

Hace DUP luego 4PUTLAM.

364FF 1GETABND ( 🡺 1lamob )

Hace 1GETLAM luego ABND.

35DEE 1ABNDSWAP ( ob 🡺 1lamob ob )

Hace 1GETABND luego SWAP.

35F42 1GETSWAP ( ob 🡺 1lamob ob )

Hace 1GETLAM luego SWAP.

2F318 1GETLAMSWP1+ ( # 🡺 1lamob #+1 )

Hace 1GETLAM luego SWAP#1+.

3632E 2GETEVAL ( 🡺 ? )

Hace 2GETLAM luego EVAL.

3483E GETLAMPAIR ( #n 🡺 #n ob lam F )

( #n 🡺 #n T )

Consigue el contenido y el nombre de un lam del entorno temporal más reciente (10 = 1lam, 20 = 2lam, etc.).

347AB DUPTEMPENV ( 🡺 )

Duplica el entorno temporal más reciente (pero siempre pone 0 al “protection word” de este nuevo entorno temporal)

2B3A6 1NULLLAM{} ( 🡺 {} )

Pone una lista con un NULLLAM en la pila.

271F4 (2NULLLAM{}) ( 🡺 {} )

Pone una lista con dos NULLLAM en la pila.

27208 (3NULLLAM{}) ( 🡺 {} )

Pone una lista con tres NULLLAM en la pila.

2B3B7 4NULLLAM{} ( 🡺 {} )

Pone una lista con cuatro NULLLAM en la pila.

Capítulo 20  
Control del Runstream

Hasta aquí, tu has visto sólamente comandos que no afectan el normal flujo de un programa. Todos los programas presentados hasta ahora funcionan de manera secuencial, desde el primer comando hasta el último, sin ningún tipo de cambio en su orden. Sin embargo, en casi todos los programas, incluso los más simples, es necesario algún tipo de interrupción en el orden normal del programa.

A veces, necesitarás tener alguna parte del programa repetida varias veces, o alguna acción debe ser ejecutada sólo bajo ciertas condiciones.

Este capítulo describirá algunas entradas de bajo nivel que afectan el normal orden de ejecución. Los problemas planteados en el párrafo anterior pueden ser resueltos también con entradas de alto nivel (bucles del capítulo 22 y condicionales del capítulo 21). Lo más probable es que tu uses esas entradas de alto nivel, en vez de los comandos de este capítulo.

Sin embargo, este capítulo también describe algunos conceptos que ayudan a comprender como funciona un programa de System RPL, y como cambiar el flujo normal de un programa.

20.1 Algunos Conceptos

Como hemos visto en la introducción, un programa de System RPL compilado consiste de una serie de punteros a direcciones en la memoria. Básicamente un programa es ejecutado saltando a la dirección apuntada, ejecutando sea lo que sea que haya ahí, regresando al programa, saltando a la siguiente dirección, y así sucesivamente.

En realidad, es más complicado, porque hay algunos objetos como números reales, cadenas e incluso otros programas insertados dentro de los programas. Esto requiere algo “mágico” (en realidad, es sólo código escrito cuidadosamente) para manejarlo apropiadamente, pero está fuera del alcance de este documento describir como se resuelve este problema. Sólo asumiremos que cuando un objeto es encontrado, este es “ejecutado”.

Para la mayoría de objetos (como números reales y cadenas), esto significa ponerlos en la pila, para programas esto significa ejecutar sus contenidos, y para otros objetos como identificadores (id) esto significa intentar llamar a sus contenidos y ejecutarlos, o simplemente ponerlos en la pila.

Puesto que los objetos son ejecutados en orden, es necesario tener alguna clase de variable que siempre apunte al siguiente objeto a ser ejecutado. Este es llamado el INTÉRPRETE DEL PUNTERO, y es guardado en un registro del CPU. Después que cada objeto es ejecutado, este puntero es avanzado para apuntar al siguiente objeto. Cuando un DUP es encontrado en el programa, ocurre lo siguiente: en realidad, la única cosa que es guardada es la dirección #03188h. Un salto es hecho a esa dirección. En esa dirección, hay código de lenguaje máquina. Este código es ejecutado y al final el intérprete del puntero es avanzado, y un salto es hecho a la siguiente dirección, cualquiera que esta sea.

Las cosas se vuelven ligeramente más complicadas cuando uno quiere ejecutar, por ejemplo, INCOMPDROP. En la dirección de este comando hay un objeto programa, cuyo contenido es :: INNERCOMP DROP ;. El problema es que es necesario hacer una interrupción para que este (sub-)programa, ejecute todo su contenido, y luego regresar hacia atrás al programa en el cual INCOMPDROP fue llamado. Puesto que es perfectamente posible que un sub-programa tenga dentro otros sub-programas, resulta que algún tipo de pila es necesaria. Cuando un programa (o cualquier otro compuesto) es ejecutado, la dirección del objeto que se encuentra después de este compuesto es puesta en esta pila.

Luego el compuesto es ejecutado, lo que significa que el puntero del intérprete apunta a cada uno de sus objetos. Cuando esto termina, una dirección es retornada desde la pila de retornos, y la ejecución retorna ahí. Esta era la dirección del siguiente objeto en el programa previo, de manera que la ejecución se reanuda apropiadamente. Esta pila es llamada la PILA DE RETORNOS.

La descripción hecha es bastante incompleta, pero debería darte una idea de cómo funcionan las cosas. Hay muchos detalles que harían una explicación detallada de los programas en System RPL demasiado larga y complicada, por lo tanto, esta explicación detallada no será dada en este libro.

Otro concepto importante es el RUNSTREAM. Esta es la secuencia de objetos que siguen al objeto que está siendo ejecutado actualmente. Por ejemplo, durante la ejecución del comando ' en este programa

:: ' DUP :: EVAL ; % 1. ;

el runstream contiene tres objetos. El primero es el comando DUP. El segundo es el programa que tiene al comando EVAL en su interior (pero no al comando EVAL o sólo el ::), y el tercero es el número real uno. Varios comandos (incluyendo a ', como verás más abajo), toman sus argumentos de los siguientes objetos en el runstream, y no desde la pila, como la mayoría de los comandos hacen. Por lo tanto, el argumento para ' es el comando DUP.

Ahora comprenderás porque este capítulo es llamado “Control del Runstream”: los comandos aquí afectan al runstream, esto es, ellos afectan al orden en el cual serán ejecutados los objetos que forman el programa.

20.2 Comandos Runstream

Los comandos descritos aquí son las acciones básicas disponibles. En la sección de referencia de abajo encontrarás varios comandos que combinan estos comandos con otros.

**Comando Pila y Descripción**

' ( 🡺 ob )

Este es muy fácil de entender: pone el objeto que está después de el (esto es, el primer objeto en el runstream) en la pila. Este objeto será colocado, no sera ejecutado; la ejecución puede hacerse después. Por ejemplo,

:: %1 %2 ' %+ EVAL ;

es equivalente a

:: %1 %2 %+ ;

Esta acción de poner el siguiente objeto en la pila en lugar de evaluarlo es llamado citación del siguiente objeto.

'R ( 🡺 ob )

Este pone en la pila al objeto apuntado por el puntero más reciente en la pila de retornos, y quita este objeto de la pila de retornos. En otras palabras, el objeto siguiente en el compuesto que contiene al compuesto que se está ejecutando ahora, es puesto en la pila y es sacado de donde estaba antes. Sin embargo, si el objeto que debería ser puesto es SEMI, entonces un compuesto nulo es puesto en su lugar. Como un ejemplo, el comando EQ: es como el comando EQ, pero uno de los argumentos está en la pila y el otro argumento después del comando (en el runstream).

El comando EQ: está definido así:

:: 'R EQ: ;

Pone en la pila al objeto que está después de EQ: y luego llama a EQ.

ticR ( 🡺 ob TRUE )

( 🡺 FALSE )

Este es similar al comando 'R, pero este no pondrá un compuesto nulo si no hay ningún objeto para ser devuelto; en este caso retorna FALSE. Si un objeto puede ser devuelto, este es puesto en la pila seguido por TRUE.

>R ( comp 🡺 )

Este pone el puntero al cuerpo del compuesto dado como argumento en la pila de retornos. Esto significa que cuando el programa actual termine, la ejecución no irá aún al programa que llamó al compuesto actual, Antes de eso, el compuesto dado como argumento de >R será ejecutado, y sólo después de que esto termine, la ejecución se reanudará al programa que llamó al programa actual. Por ejemplo, el código mostrado abajo retorna en la pila los números reales 20, 21 y 8 en ese orden.

:: ' :: %7 %1+ ; >R %20 %21 ;

R> ( 🡺 :: )

Pone en la pila un programa cuyos contenidos son aquellos apuntados por el primer nivel en la pila de retornos, el cual es borrado. En otras palabras, este pone en la pila como un programa al resto de los objetos del programa que llamó al programa actual. Estos objetos no serán ejecutados después de que el programa actual termine.

Por ejemplo, el código de abajo devuelve en la pila :: %3 %2 ; %7

:: :: R> %7 ; %3 %2 ;

Este otro código devuelve en la pila %3 %2 %7

:: :: R> EVAL %7 ; %3 %2 ;

**Comando Pila y Descripción**

R@ ( 🡺 :: )

Este es similar a R>, pero no borra lo que había en la pila de retornos.

Por ejemplo, el código de abajo devuelve en la pila :: %3 %2 ; %7 %3 %2

:: :: R> %7 ; %3 %2 ;

IDUP ( 🡺 )

Pone el intérprete del puntero en la pila de retornos. Esto significa que después de que programa actual termine, un salto sera hecho al objeto que estaba justo después de IDUP, ejecutando de este modo el resto del programa actual una vez más.

Por ejemplo, el código de abajo devuelve en la pila %5 %6 %7 %6 %7

:: %5 IDUP %6 %7 ;

RDROP ( 🡺 )

Borra el primer nivel de la pila de retornos. Esto significa que los objetos restantes en el programa que llamó al programa actual no serán ejecutados.

Por ejemplo, el código de abajo devuelve en la pila %12 %13 %14

:: :: %12 RDROP %13 %14 ; %20 %21 ;

RDUP ( 🡺 )

Duplica el primer nivel de la pila de retornos.

Por ejemplo, el código de abajo devuelve en la pila %2 %3 %4 %7 %8 %7 %8

:: :: %2 RDUP %3 %4 ; %7 %8 ;

RSWAP ( 🡺 )

Intercambia los dos primeros niveles de la pila de retornos.

Por ejemplo, el código de abajo devuelve en la pila %2 %3 %4 %7 %8 %5 %6

::

::

:: %2 RSWAP %3 %4 ;

%5 %6

;

%7 %8

;

?SEMI ( flag 🡺 )

Si el flag es TRUE, pasa por alto el resto del programa actual.

COLA ( 🡺 )

Este ejecuta solamente el siguiente objeto en el runstream, pasando por alto el resto del programa actual. El programa de abajo devuelve en la pila %1 %2

:: %1 COLA %2 %3 %4 ;

NOTA: El objeto se ejecuta como si estuviera en el programa de arriba.

Ver abajo para algunos buenos usos de COLA.

SKIP ( 🡺 )

Pasa por alto el siguiente objeto en el runstream.

El programa de abajo devuelve en la pila %1 %3 %4

:: %1 SKIP %2 %3 %4 ;

?SKIP ( flag 🡺 )

Hace SKIP si el flag es TRUE.

20.3 Usos del comando COLA

Nuestro primer ejemplo mostrará un uso util de COLA: cuando la recursividad es usada. Supongamos que tenemos dos programas abajo para calcular el factorial de un número:

ASSEMBLE

CON(1) 8 ( Le dice al parseador que el comando es 'No algebraico' )

RPL

xNAME fact

::

CK1 ( % ) ( verifica por 1 objeto en la pila )

CKREAL ( % ) ( verifica por objeto real o entero )

DUP %0> ( % flag )

NcaseSIZEERR ( si no es positivo, genera error )

( % )

%CEIL ( % ) ( si tiene parte decimal, redondea hacia arriba )

{ LAM x } BIND ( ) ( crea entorno temporal )

%1 ( %1 ) ( Argumento inicial para factorial )

factiter ( %factorial ) ( Halla factorial )

ABND ( abandona entorno temporal )

;

NULLNAME factiter ( % -> % )

:: ( %f )

LAM x %0= ( %f flag )

?SEMI ( sale si x=0 )

( %f )

LAM x %\* ( %fox ) ( Multiplica por valor actual de x )

LAM x %1- ( %fox %x-1 )

' LAM x ( %fox %x-1 lam )

STO ( %fox )

COLA ( Ejecuta factiter en el programa de arriba... )

factiter ( %factorial ) ( o sea, como si hubiera sido llamado por fact )

;

Notar la palabra COLA antes de la invocación recursiva de factiter. Sin el comando COLA, el programa hubiera requerido muchos niveles en la pila de retornos, todos ellos apuntando a SEMI. Con COLA, nada es puesto en la pila de retornos. factiter es llamado simplemente, sin guardar la dirección a donde el intérprete del puntero debería ir al saltar atrás. Esto hace que el programa fact siempre use un número fijo de niveles en la pila de retornos.

Sin embargo, COLA no es usado sólo en este caso. Es un comando muy util en otras situaciones. Digamos que en tu proyecto frecuentemente necesites realizar un case (ver sección 21.3) comparando si un número real es igual a 3. Es necesario escribir un programa para hacer esto (como el comando ya incorporado %1=case) en lugar de repetir “%3 %= case” todas las veces.

Un primer intento sería este programa:

NULLNAME %3=case

:: %3 %= case ;

Sin embargo, esto no funcionará. Esto es porque case toma su argumento desde el runstream, esto es, desde el programa que está siendo ejecutado, y no desde el compuesto que llama a ese programa. Esto significa que el argumento para case en dicho programa es ;, el cual no es el argumento deseable. Pero hay una solución: usa COLA antes de case.

NULLNAME %3=case

:: %3 %= COLA case ;

Esto borrará el resto del runstream que se encuentra después de case, y ejecuta case en el programa de arriba. Por lo tanto, si agregamos COLA antes de case, e insertamos este nuevo subprograma en otro, como aquí:

:: ... :: %3 %= COLA case ; <accion\_T> <accion\_F> ... ;

O también:

:: ... %3=case <accion\_T> <accion\_F> ... ;

Esto funcionará como si el código hubiera sido:

:: ... %3 %= case <act\_T> <act\_F> ... ;

Lo cual es lo que nosotros queríamos. Por consiguiente, la manera correcta de definir nuestro subprograma es con COLA antes de case. Esto es una frecuente combinación, de manera que existe el comando atajo, COLAcase, el cual es equivalente a COLA seguido por case. Existen otros comandos como este que puedes verlos en la referencia de abajo.

20.4 Viendo la Pila de Retornos con Debug4x

En Debug4x, puedes ver el contenido de los primeros objetos de cada nivel de la pila de retornos (return stack) en cualquier momento en el Panel “Return Stack” que se encuentra en la ventana RPL Debugging.

Por ejemplo, si escribimos el código de la izquierda en el editor de Debug4x, fijamos un breakpoint (círculo rojo) en la posición indicada y ejecutamos el comando EjReturnStack desde el emulador, podremos ver en el Panel Return Stack, todos los niveles de la pila de retornos.

Además de los niveles en la pila de retornos generados por nuestro programa, hay más niveles de pila de retornos que permiten que todo funcione correctamente y no debemos alterar esos niveles de la pila de retornos.

|  |  |
| --- | --- |
| Ventana Editor | Ventana RPL Debugging |
|  |  |

xNAME EjReturnStack

:: CK0

11.

:: 12.

:: 13.

:: 14.

15.

16.

;

71.

72.

;

:: 81. 82. ;

82.

;

91.

93.

;

20.5 Referencia

**Direcc. Nombre Descripción**

06E8E NOP ( 🡺 )

No hace nada.

06EEB 'R ( 🡺 ob )

Trae el siguiente objeto de la pila de retornos (es decir, el siguiente objeto del compuesto que está arriba del actual) en la pila (borrándolo de donde estaba). Si la pila de retornos está vacía (contiene SEMI), un programa nulo es puesto y el puntero no es avanzado.

06F66 'REVAL ( 🡺 ? )

Hace 'R luego EVAL.

36A27 'R'R ( 🡺 ob1 ob2 )

Hace 'R dos veces.

34BEF ticR ( 🡺 ob T )

( 🡺 F )

Trae el siguiente objeto de la pila de retornos hacia la pila y pone TRUE.

Si este primer nivel de la pila de retornos está vacío, lo borra y pone FALSE en la pila.

36A4A 'RRDROP ( 🡺 ob )

Hace 'R, luego RDROP.

06F9F >R ( prog 🡺 )

( {} 🡺 )

Pone el compuesto como el primer nivel de la pila de retornos.

0701F R> ( 🡺 :: )

Trae todos los objetos del nivel 1 de la pila de retornos (quitando este nivel de la pila de retornos) y los pone en la pila en un objeto programa.

07012 R@ ( 🡺 :: )

Como R>, pero el nivel 1 de la pila de retornos no se quita.

0716B IDUP ( 🡺 )

Crea un nuevo nivel de la pila de retornos que va a contener a los objetos restantes del actual runstream.

06F8E EVAL ( ob 🡺 ? )

Evalúa objeto.

262FB COMPEVAL ( comp 🡺 ? )

Parecido a EVAL, pero en el caso de listas evalúa sus elementos como si fuera un objeto programa (en cambio, EVAL no hace nada cuando en la pila hay una lista).

34BAB 2@REVAL ( 🡺 ? )

Hace EVAL sobre el siguiente objeto del programa de arriba de arriba del actual.

34BBB 3@REVAL ( 🡺 ? )

Hace EVAL sobre el siguiente objeto del programa de arriba de arriba de arriba del actual.

26111 RDUP ( 🡺 )

Duplica el primer nivel de la pila de retornos.

06FB7 RDROP ( 🡺 )

Quita el primer nivel de la pila de retornos.

**Direcc. Nombre Descripción**

343E1 2RDROP ( 🡺 )

Quita dos niveles de la pila de retornos.

343F3 3RDROP ( 🡺 )

Quita tres niveles de la pila de retornos.

36342 DROPRDROP ( ob 🡺 )

Hace DROP luego RDROP.

3597F RDROPCOLA ( 🡺 )

Hace RDROP luego COLA.

34144 RSWAP ( 🡺 )

Cambia los niveles 1 y 2 de la pila de retornos.

368C9 RSKIP ( 🡺 )

Pasa por alto el primer objeto del nivel 1 de la pila de retornos.

Equivale a hacer 'R DROP.

2644A (RROLL) ( # 🡺 )

Mueve niveles en la pila de retornos: Al nivel n de la pila de retornos lo mueve hacia el nivel 1 de la pila de retornos.

2B8BE (OBJ>R) ( ob 🡺 )

Pone en el primer nivel de la pila de retornos a ob seguido de los demás elementos del runstream. Puede usarse para un guardado temporal. Si ob es una lista, la lista es puesta entera en la pila, no los elementos individuales.

2B8E6 (R>OBJ) ( 🡺 ob )

Retorna el primer objeto del nivel 1 de la pila de retornos y quita ese nivel 1 de la pila de retornos.

0312B SEMI ( 🡺 )

Quita el resto del runstream.

20.5.1 Citando Objetos

**Direcc. Nombre Descripción**

06E97 ' ( 🡺 nob (nextob) )

Pone el siguiente objeto del programa en la pila.

3696E DUP' ( ob 🡺 ob ob nob )

Hace DUP luego '.

36996 DROP' ( ob 🡺 nob )

Hace DROP luego '.

36982 SWAP' ( ob1 ob2 🡺 ob2 ob1 nob )

Hace SWAP luego '.

369AA OVER' ( ob1 ob2 🡺 ob1 ob2 ob1 nob )

Hace OVER luego '.

369BE STO' ( ob id/lam 🡺 nob )

Hace STO luego '.

369D2 TRUE' ( 🡺 T nob )

Pone TRUE y el siguiente objeto del programa en la pila.

369FF FALSE' ( 🡺 F nob )

Pone FALSE y el siguiente objeto del programa en la pila.

369E6 ONEFALSE' ( 🡺 #1 F nob )

Pone ONE, FALSE y el siguiente objeto del programa en la pila.

**Direcc. Nombre Descripción**

36A13 #1+' ( # 🡺 #+1 nob )

Hace #1+ luego '.

36306 'NOP ( 🡺 NOP )

Pone NOP en la pila.

3619E 'ERRJMP ( 🡺 ERRJMP )

Pone ERRJMP en la pila.

2B90B 'DROPFALSE ( 🡺 DROPFALSE )

Pone DROPFALSE en la pila.

25E6A 'DoBadKey ( 🡺 DoBadKey )

Pone DoBadKey en la pila.

25E6B 'DoBadKeyT ( 🡺 DoBadKey T )

Pone DoBadKey y TRUE en la pila.

2F32E DROPDEADTRUE ( ob 🡺 DoBadKey T )

Hace DROP, luego pone DoBadKey y TRUE en la pila.

36BBE ('x\*) ( 🡺 x\* )

Pone x\* (comando \* de User RPL) en la pila.

36BD2 'xDER ( 🡺 xDER )

Pone comando xDER en la pila.

27B43 'IDFUNCTION ( 🡺 xFUNCTION )

Pone xFUNCTION (comando FUNCTION de User RPL) en la pila.

27B6B 'IDPOLAR ( 🡺 xPOLAR )

Pone xPOLAR (comando POLAR de User RPL) en la pila.

27B7F 'IDPARAMETER ( 🡺 xPARAMETRIC )

Pone xPARAMETRIC (comando PARAMETRIC de User RPL) en la pila.

27B57 ('IDCONIC) ( 🡺 xCONIC )

Pone xCONIC (comando CONIC de User RPL) en la pila.

27B93 ('IDTRUTH) ( 🡺 xTRUTH )

Pone xTRUTH (comando TRUTH de User RPL) en la pila.

27BBB ('IDHISTOGRAM) ( 🡺 xHISTOGRAM )

Pone xHISTOGRAM (comando HISTOGRAM de User RPL) en la pila.

27BCF ('IDBAR) ( 🡺 xBAR )

Pone xBAR (comando BAR de User RPL) en la pila.

27BA7 ('IDSCATTER) ( 🡺 xSCATTER )

Pone xSCATTER (comando SCATTER de User RPL) en la pila.

27BE3 ('IDFAST3D) ( 🡺 xFAST3D )

Pone xFAST3D (comando FAST3D de User RPL) en la pila.

29ED0 'Rapndit ( meta ob1...ob4 🡺 meta&ob ob1...ob4 )

Toma ob (el siguiente objeto del programa) y lo agrega al meta que está en el nivel 5 de la pila.

36AA4 'xDEREQ ( ob 🡺 flag )

Devuelve TRUE si ob es el mismo que xDER (compara con EQ). Si no lo es, devuelve FALSE.

20.5.2 Quitando los Objetos Siguientes del Programa

**Direcc. Nombre Descripción**

06FD1 COLA Evalúa el siguiente objeto del programa y borra el resto de

este programa. Ejemplos:

:: 11. 12. COLA 13. 14. 15. ; retorna 11. 12. 13.

:: 11. COLA :: 12. %SQ\_ ; 13. 14. ; retorna 11. 144.

Nota: el objeto es evaluado en el programa de arriba. Veamos:

::

::

11.

:: 12. COLA 'R 13. 14. ;

20.

21.

;

46.

47.

;

Este programa retorna 11. 12. 46. 20. 21. 47.

36A63 ONECOLA Hace ONE, luego COLA.

3635B SWAPCOLA Hace SWAP, luego COLA.

3636F XYZ>ZCOLA Hace UNROT2DROP, luego COLA.

34AD3 COLA\_EVAL Evalúa el anterior objeto del programa y borra el resto de este

programa.

Es similar a COLA pero usa el objeto anterior al comando.

Por ejemplo:

:: 6. ' :: 7. %SQ\_ ; COLA\_EVAL 8. 9. ; retorna 6. 49.

Nota: el objeto es evaluado en el programa de arriba.

Veamos:

::

::

11.

:: 12. ' 'R COLA\_EVAL 13. 14. ;

20.

21.

;

46.

47.

;

Este programa retorna 11. 12. 46. 20. 21. 47.

35994 COLACOLA En el programa actual borra todos los objetos siguientes.

En el programa de arriba, evalúa el siguiente objeto del programa (en el programa de más arriba) y borra el resto de este programa.

En otras palabras, COLACOLA borra el resto del programa actual y hace COLA en el programa de arriba.

**Direcc. Nombre Descripción**

El siguiente programa retorna 11. 12. 225. 18. 19. 20.

::

::

11.

:: 12. COLACOLA 13. 14. ;

:: 15. %SQ\_ ;

16.

17.

;

18.

19.

20.

;

El siguiente programa retorna 11. 12. 21. 18. 19. 22.

::

::

::

11.

:: 12. COLACOLA 13. 14. ;

'R

16.

;

18.

19.

;

21.

22.

;

0714D SKIP Pasa por alto el siguiente objeto del programa.

Por ejemplo

:: 11. SKIP 12. 13. 14. ; retorna 11. 13. 14.

0715C (2SKIP) Pasa por alto los siguientes dos objetos del programa.

El siguiente programa retorna 11. 12. 15. 16. 17.

:: 11. 12. 2SKIP\_ 13. 14. 15. 16. 17. ;

35715 skipcola Hace SKIP, luego COLA.

El siguiente programa retorna 11. 13. 16. 17.

:: :: 11. skipcola 12. 13. 14. 15. ;

16.

17.

;

3570C 2skipcola Hace 2SKIP, luego COLA.

35703 3skipcola Borra los siguientes 3 objetos, luego hace COLA.

356D5 5skipcola Borra los siguientes 5 objetos, luego hace COLA.

363FB COLASKIP Borra el resto del programa actual y borra un objeto del

programa de arriba.

El siguiente programa retorna 11. 16. 17. 18.

:: :: 11. COLASKIP 12. 13. 14. ;

15.

16.

17.

18.

;

20.6 Ejemplos

Ejemplo 1 RunStream

**Uso del comando 'R.**

En este ejemplo se ilustra el uso del comando 'R, el cual trae un objeto desde el primer nivel de la pila de retornos.

\* ( -> 21. 81. 22. 82. )

::

:: 21. ( 21. ) ( PilaRetornos: 81. 82. )

'R ( 21. 81. ) ( PilaRetornos: 82. ) ( llama a ob de la pila de retornos )

22. ( 21. 81. 22. ) ( PilaRetornos: 82. )

;

81. ( 21. 81. 22. ) ( Ya no se ejecuta de nuevo, pues fue llamado por 'R )

82. ( 21. 81. 22. 82. )

;

Ejemplo 2 RunStream

**Uso del comando 'R.**

Otro ejemplo con el comando 'R.

Aquí podemos ver que cuando en el primer nivel de la pila de retornos hay un objeto programa, el comando 'R trae a ese objeto programa hacia la pila sin evaluarlo.

\* ( -> 14. tag )

::

:: 12. ( %12 ) ( PilaRetornos: prog )

13. ( %12 %13 ) ( PilaRetornos: prog )

'R ( %12 %13 :: %+ DO>STR ; ) ( PilaRetornos: 14. )

EVAL ( "25." ) ( PilaRetornos: 14. )

"SUMA" ( "25." "SUMA" ) ( PilaRetornos: 14. )

>TAG ( tag ) ( PilaRetornos: 14. )

;

:: %+ DO>STR ; ( Ya no se ejecuta de nuevo, pues fue llamado por 'R )

14. ( tag 14. )

SWAP ( 14. tag )

;

Ejemplo 3 RunStream

**Uso del comando >R para guardar objetos.**

Aquí usamos la pila de retornos como un lugar donde podemos guardar objetos. El comando >R pone los objetos de la pila en el primer nivel de la pila de retornos y el comando 'R los trae desde la pila de retornos ahí hacia la pila RPN.

\* ( -> 81. 21. 82. 22. 83. 23. )

::

{ 21. 22. 23. } ( {} )

>R ( ) ( Pone los objetos en el nivel 1 de pila de retornos )

( ) ( ReturnStack: 21. 22. 23. )

81. ( 81. ) ( 81. ) ( ReturnStack: 21. 22. 23. )

'R ( 81. 21. ) ( 81. 21. ) ( ReturnStack: 22. 23. )

82. ( 81. 21. 82. ) ( 81. 21. 82. ) ( ReturnStack: 22. 23. )

'R ( 81. 21. 82. 22. ) ( 81. 21. 82. 22. ) ( ReturnStack: 23. )

83. ( 81. 21. 82. 22. 83. ) ( 81. 21. 82. 22. 83. ) ( ReturnStack: 23. )

'R ( 81. 21. 82. 22. 83. 23. ) ( 81. 21. 82. 22. 83. ) ( ReturnStack: :: ; )

;

Ejemplo 4 RunStream

**Uso del comando RSWAP para cambiar dos niveles de la pila de retornos.**

Aquí usamos un bucle DO/LOOP (ver sección 22.2) para sumar los elementos de dos listas.

Por ejemplo: {100. 200. 300.} {5. 6. 7.} retornará { 105. 206. 307. }

Se usa el comando >R para poner los elementos de la lista de la pila RPN en la pila de retornos, para que sean llamados más adelante.

Como veremos en otro capítulo, el comando ZERO\_DO crea un nuevo nivel en la pila de retornos (además hace otras cosas).

Por lo tanto, los números reales que queremos llamar ya no están en el nivel 1 de la pila de retornos. Ahora están en el nivel 2 de la pila de retornos.

Si deseamos llamar a estos números reales del nivel 2 de la pila de retornos, antes debemos usar el comando RSWAP. Este comando cambia los niveles 1 y 2 de la pila de retornos. Ahora los números reales están en el nivel 1 de la pila de retornos y ya podemos usar el comando 'R para traer a un número real. Luego volvemos a usar RSWAP para cambiar nuevamente los niveles 1 y 2 de la pila de retornos, lo que permite que el entorno DO/LOOP funcione adecuadamente.

\* Suma los elementos de dos listas, elemento por elemento

\* {%} {%}' -> {%}'' )

:: ( {%} {%}' )

>R ( {%} ) ( ReturnStack: reales de lista {%}' )

INNERDUP ( %1...%n #n ) ( ReturnStack: reales de lista {%}' )

ZERO\_DO ( RS1: ROLL RSWAP 'R RSWAP %+ ISTOP@ LOOP {}N )

( ... ) ( RS2: reales de lista {%}' )

ROLL ( ... %i )

RSWAP ( ... %i ) ( RS1: reales de lista {%}' )

( RS2: ROLL RSWAP 'R RSWAP %+ ISTOP@ LOOP {}N )

'R ( ... %i %i' ) ( trae ob del nivel 1 de la pila de retornos )

RSWAP ( ... %i %i' ) ( RS1: ROLL RSWAP 'R RSWAP %+ ISTOP@ LOOP {}N )

( RS2: reales de lista {%}' )

%+ ( ... %i'' )

ISTOP@ ( ... %i'' #n )

LOOP

( %1''...%n'' #n ) ( ReturnStack: SEMI ) ( {%}' se acabo )

{}N ( {%}'' )

;

Ejemplo 5 RunStream

**Uso del comando RROLL para cambiar dos niveles de la pila de retornos.**

El comando RROLL\_ ( #n 🡺 ) permite cambiar dos niveles de la pila de retornos. Trae el nivel n de la pila de retornos hacia el nivel 1 de la pila de retornos.

\* ( -> 10. 220. 113. 114. 53. 54. )

::

::

::

:: 10.

BINT3 RROLL\_

220.

;

53.

;

54.

;

113.

114.

;

Ejemplo 6 RunStream

**Uso del comando 2@REVAL**

\* ( -> 10. 11. 12. "aaaX" 13. 21. "aaaX" 31. )

::

10.

:: 11.

:: 12.

2@REVAL ( llama a :: "aaa" "X" &$ ; y lo evalúa )

13.

;

21.

;

:: "aaa" "X" &$ ;

31.

;

Ejemplo 7 RunStream

**Uso del comando 3@REVAL.**

\* ( -> 12. "aaaX" 13. 21. 31. "aaaX" 41. )

::

::

::

:: 12.

3@REVAL ( llama a :: "aaa" "X" &$ ; y lo evalúa )

13.

;

21.

;

31.

;

:: "aaa" "X" &$ ;

41.

;

Capítulo 21  
Condicionales

En System RPL, los condicionales son un poco diferentes a los de User RPL.

La primera diferencia, es que en User RPL, “falso” es representado por el número real cero; y otro valor representa “verdadero”. En System RPL, “falso” es representado por la palabra FALSE, y “verdadero” es representado por la palabra TRUE. Al ejecutar TRUE o FALSE, estas palabras se colocan en la pila. Todos los comandos que hacen un test, retornan una de estas dos palabras. Comandos como IT o case toman TRUE o FALSE como argumento.

Se puede convertir TRUE o FALSE a los números reales 0 o 1 con COERCEFLAG. Para hacer la transformación opuesta puedes usar el comando %0<>.

Hay muchos comandos que ponen TRUE, FALSE, o alguna combinación de ellos en la pila. Véalos en la lista de abajo.

Los operadores booleanos también están presentes. Son: NOT, AND, OR y XOR. Hay algunas combinaciones. Véalos en la lista de abajo.

21.1 Tests

Los comandos tests son comandos que toman uno o más argumentos y retornan TRUE o FALSE, después de hacer alguna clase de comparación o prueba entre los argumentos. Los tests que toman como argumento a un determinado tipo de objeto son listados en el capítulo correspondiente a ese tipo de objeto. Los tests para saber si un objeto es de un deteminado tipo son listados en el capítulo 29. Otras clases de tests son listadas en la referencia de este capítulo. Los más importantes de estos tests son EQ y EQUAL. Ambos toman dos objetos y retornan un flag.

El comando EQ verifica si los dos objetos son el mismo objeto, es decir, verifica si ambos objetos ocupan la misma dirección en la memoria.

El comando EQUAL verifica si los objetos son iguales en terminos del contenido de los objetos.

La diferencia es que :: BINT2 # 2 EQUAL ; retorna TRUE, pero si EQUAL es reemplazada por EQ, entonces el programa retorna FALSE, porque un objeto es el bint2 (ya incorporado en ROM), cuya dirección es #3311B, y el otro es un bint cuya dirección no se puede predecir, pero de hecho no está en la ROM.

Otro ejemplo: si pones una cadena en el nivel 1 y presionas ENTER (o ejecutas el comando DUP), EQ y EQUAL retornarán TRUE. Sin embargo, si tu ingresas una cadena, y luego escribes nuevamente la misma cadena en la pila, sólo EQUAL retornará TRUE. Esto sucede porque los contenidos de las cadenas son los mismos, pero son diferentes objetos en la memoria, pues ocupan diferentes direcciones en la memoria.

Cuando sea posible debes usar EQ en tus programas debido a que es más rápido que EQUAL.

21.2 If. . . Then. . . Else

La mayoría de las veces los condicionales TRUE y FALSE serán argumentos de los comandos IT y ITE.

**Comando y su respectiva acción**

IT ( flag 🡺 )

Si el flag es TRUE, el siguiente objeto es ejecutado, de lo contrario ese objeto es pasado por alto.

ITE ( flag 🡺 )

Si el flag es TRUE, el siguiente objeto es ejecutado, y el segundo es pasado por alto.

Si el flag es FALSE, el siguiente objeto es pasado por alto y el segundo es ejecutado.

Con el siguiente código: si en la pila se encuentra el número real cero, entonces se cambia a uno, pero si en la pila se encuentra otro número, entonces no hace nada.

... DUP %0= IT %1+ ...

El siguiente código pone en la pila la cadena "Iguales" si los dos objetos son iguales, y "Diferentes" si no los son.

... EQUAL ITE "Iguales" "Diferentes" ...

Naturalmente, cuando necesites ejecutar varios comandos, necesitarás incluirlos dentro de un programa usando los delimitadores :: y ;.

21.3 Case

Los comandos CASE son útiles para tomar decisiones. El comando básico es case, pero hay combinaciones de este con tests y otros comandos.

El comando case toma un flag en el nivel uno.

Si el flag es TRUE, el siguiente objeto es ejecutado, pero sólo este, y el resto del programa es pasado por alto. Por lo tanto, TRUE case es equivalente a COLA.

Si el flag es FALSE, el siguiente objeto del programa es pasado por alto y la ejecución continua después de ese objeto. Por lo tanto, FALSE case es equivalente a SKIP.

El ejemplo mostrado abajo muestra como hacer una estructura case similar a las encontradas en otros lenguajes (incluso User RPL). Este pone una cadena que representa el bint que se encuentra en el nivel uno.

::

DUP #0= case "Cero"

DUP BINT1 #= case "Uno"

DUP BINT2 #= case "Dos"

...

;

Hay muchos comandos que combinan case con otros comandos. Uno de ellos es el comando OVER#=case. No es dificil suponer lo que hace este comando. Primero hace OVER. Luego, #= compara dos bints. Finalmente, el case funciona como se explico arriba. Usando este comando el código mostrado arriba podría reescribirse asi:

::

BINT0 OVER#=case "Cero"

BINT1 OVER#=case "Uno"

BINT2 OVER#=case "Dos"

...

;

En la sección de referencia de abajo, encontrarás una lista de los comandos que ejecutan un case además de alguna otra acción. Los nombres de estos comandos se componen de una parte inicial, luego el case mismo y una parte final. Algunos comandos tienen sólo la parte inicial o la parte final, algunos tienen ambas partes. La parte inicial representa los comandos que son ejecutados antes del case, y deberián representar fielmente lo que hace el comando para comprender su acción. Por ejemplo, el comando NOTcase es equivalente a NOT seguido por case. Para la parte final, las cosas se vuelven más complicadas, pues hay dos clases de parte final.

El primer tipo tiene la parte final escrita en letras mayúsculas. Para estos comandos, esa parte final es ejecutada si el flag es TRUE. Tú solamente debes colocar la acción para el caso que el flag sea FALSE. Por ejemplo, este código:

... caseDROP <FalseAction> ...

es equivalente a

... case DROP <FalseAction> ...

El segundo tipo tiene la parte final en letras minúsculas. En este caso, los comandos en la parte final son ejecutados junto con el objeto siguiente del programa. Por ejemplo, este código:

... casedrop <TrueAction> <FalseAction> ...

es equivalente a

... case :: DROP <TrueAction> ; <FalseAction> ...

Desafortunadamente, algunos comandos han sido mal nombrados y la convención mencionada no se ha tenido en cuenta. Esas entradas están marcadas claramente en la referencia de abajo.

También los “diagramas de pila” de la mayoría de los comandos de abajo no son verdaderos diagramas de pila.

Lo que está al lado izquierdo de la flecha es el contenido de la pila antes de llamar al comando, como es usual. ob1 y ob2 son objetos diferentes.. f1 y f2 son flags diferentes; T representa TRUE y F, FALSE. #m y #n representan dos enteros binarios. #set es el número de un flag que se encuentra activado, #clr es el número de un flag que se encuentra desactivado.

A la derecha de la flecha se encuentran los objetos que serán ejecutados según los argumentos dados al comando. Estos tienen la forma:

:: <test\_word> <ob1> ... <obn> ;

En los diagramas, <rest> representa todos los objetos que aparecen después del objeto que aparece antes de <rest>. En este lado derecho también hay objetos que aparecen sin los paréntesis < >. Estos son objetos que aparecerán en la pila después de que el comando sea ejecutado, y no son objetos del programa que se encuentren después del comando.

21.4 Referencia

21.4.1 Flags Booleanos

**Direcc. Nombre Descripción**

2602B COERCEFLAG ( T 🡺 %1 )

( F 🡺 %0 )

Convierte un flag de System RPL a flag de User RPL.

301BA %0<> ( % 🡺 flag )

Puede ser usado para convertir un flag de User RPL a flag de System RPL.

03A81 TRUE ( 🡺 T )

27E87 TrueTrue ( 🡺 T T )

36540 TrueFalse ( 🡺 T F )

aka: TRUEFALSE

03AC0 FALSE ( 🡺 F )

36554 FalseTrue ( 🡺 F T )

aka: FALSETRUE

283E8 FalseFalse ( 🡺 F F )

27E9B failed ( 🡺 F T )

35280 DROPTRUE ( ob 🡺 T )

2D7006 ˆ2DROPTRUE ( ob ob' 🡺 T )

35289 DROPFALSE ( ob 🡺 F )

35B32 2DROPFALSE ( ob1 ob2 🡺 F )

28211 NDROPFALSE ( ob1..obn #n 🡺 F )

2812F SWAPTRUE ( ob1 ob2 🡺 ob2 ob1 T )

374BE SWAPDROPTRUE ( ob1 ob2 🡺 ob2 T )

35EF2 XYZ>ZTRUE ( ob1 ob2 ob3 🡺 ob3 T )

2962A RDROPFALSE ( 🡺 F )

Pone FALSE en la pila y borra el resto del programa actual.

Por ejemplo este programa:

:: %5 %6 RDROPFALSE %7 %8 ;

retorna %5 %6 FALSE

Este otro programa:

:: %15 :: %16 RDROPFALSE %17 %18 ; %19 ;

retornará %15 %16 FALSE %19

El siguiente programa retornará %5 FALSE

:: %5 TRUE ITE RDROPFALSE %6 %21 %22 ;

El siguiente programa retornará %5 %6 %21 %22

:: %5 FALSE ITE RDROPFALSE %6 %21 %22 ;

03AF2 NOT ( flag 🡺 flag' )

Retorna FALSE si la entrada es TRUE, y viceversa.

03B46 AND ( flag1 flag2 🡺 flag )

Retorna TRUE si ambos flags son TRUE.

03B75 OR ( flag1 flag2 🡺 flag )

Retorna TRUE si al menos un flag es TRUE.

03ADA XOR ( flag1 flag2 🡺 flag )

Retorna TRUE si los flags son diferentes.

365F9 ORNOT ( flag1 flag2 🡺 flag )

Retorna FALSE si al menos un flag es TRUE.

**Direcc. Nombre Descripción**

35C7C NOTAND ( flag1 flag2 🡺 flag )

Retorna TRUE si el flag1 es TRUE y el flag2 es FALSE.

35CB8 ROTAND ( flag1 ob flag2 🡺 ob flag )

Retorna TRUE si ambos flags son TRUE.

21.4.2 Tests Generales

**Direcc. Nombre Descripción**

03B2E EQ ( ob1 ob2 🡺 flag )

Retorna TRUE si ambos objetos son el mismo objeto, es decir, si ocupan el mismo espacio físico en la memoria. Sólo las direcciones de los objetos son comparadas. Ejemplos:

:: # 1 # 1 EQ ; retorna FALSE.

:: # 1 DUP EQ ; retorna TRUE.

:: # 1 DUP TOTEMPOB EQ ; retorna FALSE.

:: "ABC" "ABC" EQ ; retorna FALSE.

:: "ABC" DUP EQ ; retorna TRUE.

:: "ABC" DUP TOTEMPOB EQ ; retorna FALSE.

:: BINT1 BINT1 EQ ; retorna TRUE.

:: BINT1 #1 EQ ; retorna TRUE.

:: BINT1 # 1 EQ ; retorna FALSE.

36621 2DUPEQ ( ob1 ob2 🡺 ob1 ob2 flag )

Hace 2DUP luego EQ.

3664E EQOR ( flag ob1 ob2 🡺 flag' )

Hace EQ luego OR.

3607F EQOVER ( ob3 ob1 ob2 🡺 ob3 flag ob3 )

Hace EQ luego OVER.

3663A EQ: ( ob1 🡺 :: flag <ob2> <rest> ; )

Hace EQ con un objeto antes y el otro después del comando.

Ejemplos:

:: # 1 EQ: # 1 ; retorna FALSE.

:: BINT1 EQ: BINT1 ; retorna TRUE.

:: BINT1 EQ: # 1 ; retorna FALSE.

:: % 1 EQ: % 1 ; retorna FALSE.

:: %1 EQ: %1 ; retorna TRUE.

:: %1 EQ: % 1 ; retorna FALSE.

:: { %1 %2 } CARCOMP EQ: %1 ; retorna TRUE.

:: { %1 %2 } CARCOMP EQ: % 1 ; retorna FALSE.

36635 DUPEQ: ( ob1 🡺 :: ob1 flag <ob2> <rest> ; )

Hace DUP luego EQ:

03B97 EQUAL ( ob1 ob2 🡺 flag )

Retorna TRUE si los objetos son iguales (pero no necesariamente tienen la misma dirección en memoria, es decir, no necesariamente son el mismo objeto).

Por lo tanto, retorna TRUE si sus prólogos y contenidos son los mismos. Ejemplos:

:: # 1 # 1 EQUAL ; retorna TRUE.

:: # 1 DUP EQUAL ; retorna TRUE.

:: # 1 DUP TOTEMPOB EQUAL ; retorna TRUE.

**Direcc. Nombre Descripción**

:: %1 %1 EQUAL ; retorna TRUE.

:: %1 % 1 EQUAL ; retorna TRUE.

:: %1 Z1\_ EQUAL ; retorna FALSE.

:: BINT1 BINT1 EQUAL ; retorna TRUE.

:: BINT1 #1 EQUAL; retorna TRUE.

:: BINT1 # 1 EQUAL; retorna TRUE.

3660D EQUALNOT ( ob1 ob2 🡺 flag )

Retorna TRUE si los objetos son diferentes.

36662 EQUALOR ( flag ob1 ob2 🡺 flag' )

Hace EQUAL luego OR.

0FF006 ˆContains? ( ob1 ob2 🡺 ob1 ob2 flag )

Prueba si ob1 contiene a ob2.

Si ob1 es un simbólico, entonces busca en ob1, incluso en simbólicos dentro de ob1.

Si ob1 es una lista (o una matriz simbólica), entonces busca si ob2 es elemento de ob1, pero no busca en listas (o filas) contenidas en ob1.

Si ob1 es otro tipo de objeto, entonces prueba si ob1 y ob2 son iguales.

Ejemplos:

:: SYMBOL ID X Z3\_ x+ Z9\_ x= ; ' x= FLASHPTR Contains? ;

retorna SYMBOL ID X Z3\_ x+ Z9\_ x= ; ' x= TRUE.

:: { ID X Z3\_ x+ Z9\_ x= } x= FLASHPTR Contains? ;

retorna { ID X Z3\_ x+ Z9\_ x= } x= TRUE.

:: SYMBOL ID X SYMBOL ID f ; BINT1 xFCNAPPLY ; ' ID f FLASHPTR Contains? ;

retorna

SYMBOL ID X SYMBOL ID f ; BINT1 xFCNAPPLY ;

' ID f TRUE.

:: { ID X SYMBOL ID f ; BINT1 xFCNAPPLY }

' ID f FLASHPTR Contains? ;

retorna

{ ID X SYMBOL ID f ; BINT1 xFCNAPPLY } ID f FALSE.

:: { %6 %7 %8 %9 } % 7 FLASHPTR Contains? ;

retorna { %6 %7 %8 %9 } % 7 TRUE.

:: { %6 { %7 %8 } %9 } % 7 FLASHPTR Contains? ;

retorna { %6 { %7 %8 } %9 } % 7 FALSE.

:: MATRIX ID X ; ' ID X FLASHPTR Contains? ;

retorna MATRIX ID X ; ID X TRUE.

:: MATRIX MATRIX ID X ; ; ' ID X FLASHPTR Contains? ;

retorna MATRIX MATRIX ID X ; ; ID X FALSE.

:: MATRIX MATRIX ID X ; ; MATRIX ID X ; FLASHPTR Contains? ;

retorna

MATRIX MATRIX ID X ; ; MATRIX ID X ; TRUE.

:: %7 %7 FLASHPTR Contains? ;

retorna %7 %7 TRUE.

:: %7 % 7 FLASHPTR Contains? ;

retorna %7 % 7. TRUE.

:: ' :: %6 %7 ; ' :: %6 %7 ; FLASHPTR Contains? ;

retorna :: %6 %7 ; :: %6 %7 ; TRUE.

:: ' :: %6 %7 ; ' :: %6 % 7 ; FLASHPTR Contains? ;

retorna :: %6 %7 ; :: %6 % 7 ; FALSE.

21.4.3 Tests con True/False

**Direcc. Nombre Descripción**

34AA1 ?SEMI ( T 🡺 :: ; )

( F 🡺 :: <ob1> <rest> ; )

Si el flag es TRUE, pasa por alto el resto del programa.

Si el flag es FALSE, no hace nada.

34A92 NOT?SEMI ( T 🡺 :: <ob1> <rest> ; )

( F 🡺 :: ; )

Si el flag es FALSE, pasa por alto el resto del programa.

Si el flag es TRUE, no hace nada.

3692D ?SEMIDROP ( ob T 🡺 :: ob ; )

( ob F 🡺 :: <ob1> <rest> ; )

Si el flag es TRUE, pasa por alto el resto del programa.

Si el flag es FALSE, hace DROP.

34BD8 NOT?DROP ( ob T 🡺 :: ob <ob1> <rest> ; )

( ob F 🡺 :: <ob1> <rest> ; )

Si el flag es TRUE, no hace nada.

Si el flag es FALSE, hace DROP.

35F56 ?SWAP ( ob1 ob2 T 🡺 :: ob2 ob1 <ob1> <rest> ; )

( ob1 ob2 F 🡺 :: ob1 ob2 <ob1> <rest> ; )

Si el flag es TRUE, hace SWAP.

Si el flag es FALSE, no hace nada.

35DDA ?SKIPSWAP ( ob1 ob2 T 🡺 :: ob1 ob2 <ob1> <rest> ; )

( ob1 ob2 F 🡺 :: ob2 ob1 <ob1> <rest> ; )

Si el flag es TRUE, no hace nada.

Si el flag es FALSE, hace SWAP.

35F97 ?SWAPDROP ( ob1 ob2 T 🡺 :: ob2 <ob1> <rest> ; )

( ob1 ob2 F 🡺 :: ob1 <ob1> <rest> ; )

Si el flag es TRUE, hace SWAP DROP.

Si el flag es FALSE, hace DROP.

35F7E NOT?SWAPDROP ( ob1 ob2 T 🡺 :: ob2 <ob1> <rest> ; )

( ob1 ob2 F 🡺 :: ob1 <ob1> <rest> ; )

Si el flag es TRUE, hace DROP.

Si el flag es FALSE, hace SWAP DROP.

070FD RPIT ( T ob 🡺 :: ob\_ejecutado <ob1> <rest> ; )

( F ob 🡺 :: <ob1> <rest> ; )

Si el flag es TRUE, ejecuta el objeto ob.

Si el flag es FALSE, borra el objeto ob de la pila.

070C3 RPITE ( T ob1 ob2 🡺 :: ob1\_ejec <ob1> <rest> ; )

( F ob1 ob2 🡺 :: ob2\_ejec <ob1> <rest> ; )

Si el flag es TRUE, ejecuta el objeto ob1 y borra el objeto ob2.

Si el flag es FALSE, ejecuta objeto ob2 y borra el objeto ob1.

34AF4 COLARPITE ( T ob1 ob2 🡺 :: ob1\_ejec ; )

( F ob1 ob2 🡺 :: ob2\_ejec ; )

Pasa por alto el resto del programa. Además:

Si el flag es TRUE, ejecuta el objeto ob1 y borra ob2.

Si el flag es FALSE, ejecuta el objeto ob2 y borra ob1.

**Direcc. Nombre Descripción**

34B4F 2'RCOLARPITE ( T 🡺 :: ob1\_ejec ; )

( F 🡺 :: ob2\_ejec ; )

Pila de retornos:

( ob1 ob2 🡺 )

Pasa por alto el resto del programa. Además:

Si el flag es TRUE, ejecuta el objeto ob1 y borra ob2.

Si el flag es FALSE, ejecuta el objeto ob2 y borra ob1.

Por ejemplo, este programa retorna 21. 100. 25. 26.

::

:: %21 TRUE 2'RCOLARPITE %22 %23 %24 ;

:: %10 %10 %\* ;

:: %15 %15 %\* ;

%25

%26

;

Si se cambia el flag a FALSE, retorna 21. 225. 25. 26.

34A22 IT ( T 🡺 :: <ob1> <rest> ; )

( F 🡺 :: <ob2> <rest> ; )

Si el flag es FALSE, pasa por alto el siguiente objeto del programa.

0712A ?SKIP ( T 🡺 :: <ob2> <rest> ; )

( F 🡺 :: <ob1> <rest> ; )

Si el flag es TRUE, pasa por alto el siguiente objeto del programa.

aka: NOT\_IT

34B3E ITE ( T 🡺 :: <ob1> <ob3> <rest> ; )

( F 🡺 :: <ob2> <rest> ; )

36865 COLAITE ( T 🡺 :: <ob1> ; )

( F 🡺 :: <ob2> ; )

34ABE ITE\_DROP ( ob T 🡺 :: <ob2> <rest> ; )

( ob F 🡺 :: ob <ob1> <rest> ; )

Si el flag es TRUE, borra ob y pasa por alto el siguiente objeto del programa.

Si el flag es FALSE, no hace nada.

36EED ANDITE ( f1 f2 🡺 :: <ob1> <ob3> <rest> ; )

( f1 f2 🡺 :: <ob2> <rest> ; )

349F9 case ( T 🡺 :: <ob1> ; )

( F 🡺 :: <ob2> <rest> ; )

Si el flag es TRUE, ejecuta el siguiente objeto del programa y pasa por alto el resto.

Si el flag es FALSE, pasa por alto el siguiente objeto del programa.

34A13 NOTcase ( T 🡺 :: <ob2> <rest> ; )

( F 🡺 :: <ob1> ; )

36D4E ANDcase ( f1 f2 🡺 :: <ob1> ; )

( f1 f2 🡺 :: <ob2> <rest> ; )

36E6B ANDNOTcase ( f1 f2 🡺 :: <ob1> ; )

( f1 f2 🡺 :: <ob2> <rest> ; )

359E3 ORcase ( f1 f2 🡺 :: <ob1> ; )

( f1 f2 🡺 :: <ob2> <rest> ; )

**Direcc. Nombre Descripción**

3495D casedrop ( ob T 🡺 :: <ob1> ; )

( ob F 🡺 :: ob <ob2> <rest> ; )

Si el flag es TRUE, hace DROP, ejecuta el siguiente objeto del programa y pasa por alto el resto.

Si el flag es FALSE, pasa por alto el siguiente objeto del programa.

3494E NOTcasedrop ( ob T 🡺 :: ob <ob2> <rest> ; )

( ob F 🡺 :: <ob1> ; )

34985 case2drop ( ob1 ob2 T 🡺 :: <ob1> ; )

( ob1 ob2 F 🡺 :: ob1 ob2 <ob2> <rest> ; )

34976 NOTcase2drop ( ob1 ob2 T 🡺 :: ob1 ob2 <ob2> <rest> ; )

( ob1 ob2 F 🡺 :: <ob1> ; )

349B1 caseDROP ( ob T 🡺 :: ; )

( ob F 🡺 :: ob <ob1> <rest> ; )

Si el flag es TRUE, hace DROP y pasa por alto el resto del programa.

Si el flag es FALSE, no hace nada.

El resultado es el mismo que poner las palabras case DROP

349C6 NOTcaseDROP ( ob T 🡺 :: ob <ob1> <rest> ; )

( ob F 🡺 :: ; )

368FB casedrptru ( ob T 🡺 T )

( ob F 🡺 :: ob <ob1> <rest> ; )

Nota: debería llamarse caseDRPTRU.

365B3 casedrpfls ( ob T 🡺 F )

( ob F 🡺 :: ob <ob1> <rest> ; )

Nota: debería llamarse caseDRPFLS.

36B3A NOTcsdrpfls ( ob T 🡺 :: ob <ob1> <rest> ; )

( ob F 🡺 F )

Nota: debería llamarse NOTcaseDRPFLS.

349D6 case2DROP ( ob1 ob2 T 🡺 :: ; )

( ob1 ob2 F 🡺 :: ob1 ob2 <ob1> <rest> ; )

349EA NOTcase2DROP ( ob1 ob2 T 🡺 :: ob1 ob2 <ob1> <rest> ; )

( ob1 ob2 F 🡺 :: ; )

365CC case2drpfls ( ob1 ob2 T 🡺 F )

( ob1 ob2 F 🡺 :: ob1 ob2 <ob1> <rest> ; )

Nota: debería llamarse case2DRPFLS.

3652C caseTRUE ( T 🡺 T )

( F 🡺 :: <ob1> <rest> ; )

36914 NOTcaseTRUE ( T 🡺 :: <ob1> <rest> ; )

( F 🡺 T )

365E5 caseFALSE ( T 🡺 F )

( F 🡺 :: <ob1> <rest> ; )

2B2C5 NOTcaseFALSE ( T 🡺 :: <ob1> <rest> ; )

( F 🡺 F )

**Direcc. Nombre Descripción**

359AD COLAcase ( T 🡺 :: <ob1> ; )

( F 🡺 :: <ob2> <rest> ; )

Pasa por alto el resto del programa y ejecuta case en el programa de arriba.

Por ejemplo, este programa retorna 15. 16. 24.

::

%15

:: %16 TRUE COLAcase %2 %3 %4 ;

%24

%25

%26

;

Si se cambia el flag a FALSE, retorna 15. 16. 25. 26.

359C8 COLANOTcase ( T 🡺 :: <ob2> <rest> ; )

( F 🡺 :: <ob1> ; )

Pasa por alto el resto del programa y ejecuta NOTcase en el programa de arriba.

21.4.4 Tests con enterios binarios

**Direcc. Nombre Descripción**

363B5 #=?SKIP ( #m #n 🡺 :: <ob2> <rest> ; )

( #m #n 🡺 :: <ob1> <rest> ; )

363E2 #>?SKIP ( #m #n 🡺 :: <ob1> <rest> ; )

( #m #n 🡺 :: <ob2> <rest> ; )

35C54 #=ITE ( #m #n 🡺 :: <ob1> <ob3> <rest> ; )

( #m #n 🡺 :: <ob2> <rest> ; )

36F29 #<ITE ( #m #n 🡺 :: <ob1> <ob3> <rest> ; )

( #m #n 🡺 :: <ob2> <rest> ; )

36F3D #>ITE ( #m #n 🡺 :: <ob2> <rest> ; )

( #m #n 🡺 :: <ob1> <ob3> <rest> ; )

348D2 #=case ( #m #n 🡺 :: <ob1> ; )

( #m #n 🡺 :: <ob2> <rest> ; )

348E2 OVER#=case ( #m #n 🡺 :: #m <ob1> ; )

( #m #n 🡺 :: #m <ob2> <rest> ; )

34939 #=casedrop ( #m #n 🡺 :: <ob1> ; )

( #m #n 🡺 :: #m <ob2> <rest> ; )

Nota: debería llamarse OVER#=casedrop.

36590 #=casedrpfls ( #m #n 🡺 F )

( #m #n 🡺 :: #m <ob1> <rest> ; )

Nota: debería llamarse OVER#=caseDRPFLS.

36D9E #<>case ( #m #n 🡺 :: <ob2> <rest> ; )

( #m #n 🡺 :: <ob1> ; )

36D76 #<case ( #m #n 🡺 :: <ob1> ; )

( #m #n 🡺 :: <ob2> <rest> ; )

36DCB #>case ( #m #n 🡺 :: <ob2> <rest> ; )

( #m #n 🡺 :: <ob1> ; )

34A7E #0=?SEMI ( #0 🡺 :: ; )

( # 🡺 :: <ob1> <rest> ; )

36383 #0=?SKIP ( #0 🡺 :: <ob2> <rest> ; )

( # 🡺 :: <ob1> <rest> ; )

**Direcc. Nombre Descripción**

36F15 #0=ITE ( #0 🡺 :: <ob1> <ob3> <rest> ; )

( # 🡺 :: <ob2> <rest> )

36ED4 DUP#0=IT ( #0 🡺 :: #0 <ob1> <rest> ; )

( # 🡺 :: # <ob2> <rest> ; )

36F51 DUP#0=ITE ( #0 🡺 :: #0 <ob1> <ob3> <rest> ; )

( # 🡺 :: # <ob2> <rest> ; )

348FC #0=case ( #0 🡺 :: <ob1> ; )

( # 🡺 :: <ob2> <rest> ; )

348F7 DUP#0=case ( #0 🡺 :: #0 <ob1> ; )

( # 🡺 :: # <ob2> <rest> ; )

3490E DUP#0=csedrp ( #0 🡺 :: <ob1> ; )

( # 🡺 :: # <ob2> <rest> ; )

36D21 DUP#0=csDROP ( #0 🡺 :: ; )

( # 🡺 :: # <ob1> <rest> ; )

36D8A #1=case ( #1 🡺 :: <ob1> ; )

( # 🡺 :: <ob2> <rest> ; )

3639C #1=?SKIP ( #1 🡺 :: <ob2> <rest> ; )

( # 🡺 :: <ob1> <rest> ; )

36DB2 #>2case ( #0/#1/#2 🡺 :: <ob2> <rest> ; )

( # 🡺 :: <ob1> ; )

25E72 ?CaseKeyDef ( # #' 🡺 :: ' ob1 T ; )

( # #' 🡺 :: # <ob2> <rest> ; )

Compara dos bints.

Si son iguales, coloca el siguiente objeto del programa seguido por TRUE y pasa por alto el resto del programa.

Si son diferentes, coloca el primer bint y pasa por alto el siguiente objeto del programa.

Por ejemplo, este programa retorna :: %11 %2 %\* ; TRUE

::

BINT7 BINT7

?CaseKeyDef

:: %11. %2 %\* ;

%23

%24

;

Cambiando los bints a BINT7 BINT8 retorna BINT7 %23 %24

**Direcc. Nombre Descripción**

25E73 ?CaseRomptr@ ( # #' 🡺 ob T )

( # #' 🡺 F )

( # #' 🡺 :: # <ob2> <rest> ; )

Compara dos bints.

Si son iguales, busca en el siguiente objeto del programa (que debe ser un rompointer) y pasa por alto el resto. Si el rompointer existe coloca su contenido seguido por TRUE. De lo contrario coloca FALSE.

Si son diferentes, coloca el primer bint y pasa por alto el siguiente objeto del programa.

Por ejemplo, este programa retorna el contenido del rompointer seguido por TRUE

::

BINT7 BINT7

?CaseRomptr@

ROMPTR 0AB 05E

%23

%24

;

Si se coloca un rompointer que no existe, retorna FALSE

Al cambiar los bints a BINT7 BINT8 retorna BINT7 %23 %24

21.4.5 Tests con números reales y complejos.

**Direcc. Nombre Descripción**

2B149 %0=case ( %0 🡺 :: %0 <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

Equivale a usar DUP %0 EQUAL case

Equivale a usar DUP %0= case

Equivale a usar DUP%0= case

36DDF j%0=case ( %0 🡺 :: <ob1> ; )

( ob 🡺 :: <ob2> <rest> ; )

Equivale a usar %0 EQUAL case

Equivale a usar %0= case

2B15D C%0=case ( C%0 🡺 :: C%0 <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

Equivale a usar DUP C%0 EQUAL case

Equivale a usar DUP C%0= case

2B11C num0=case ( 0 🡺 :: 0 <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

La condición para que el test sea TRUE es que en la pila se encuentre un cero real o un cero complejo.

Equivale a usar

DUP %0 EQUAL OVER C%0 EQUAL OR case

2B1A3 %1=case ( %1 🡺 :: %1 <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

Equivale a usar DUP %1 EQUAL case

Equivale a usar DUP %1 %= case

2B1C1 C%1=case ( C%1 🡺 :: C%1 <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

Equivale a usar DUP C%1 EQUAL case

**Direcc. Nombre Descripción**

2B176 num1=case ( 1 🡺 :: 1 <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

La condición para que el test sea TRUE es que en la pila se encuentre un uno real o un uno complejo.

Equivale a usar

DUP %1 EQUAL OVER C%1 EQUAL OR case

2B20C %2=case ( %2 🡺 :: %2 <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

Equivale a usar DUP %2 EQUAL case

Equivale a usar DUP %2 %= case

2B22A C%2=case ( C%2 🡺 :: C%2 <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

Equivale a usar DUP C% 2 0 EQUAL case

2B1DF num2=case ( 2 🡺 :: 2 <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

La condición para que el test sea TRUE es que en la pila se encuentre un dos real o un dos complejo.

Equivale a usar

DUP %2 EQUAL OVER C% 2 0 EQUAL OR case

2B289 %-1=case ( %-1 🡺 :: %-1 <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

Equivale a usar DUP %-1 EQUAL case

Equivale a usar DUP %-1 %= case

2B2A7 C%-1=case ( C%-1 🡺 :: C%-1 <ob1> ; )

( ob 🡺 ob <ob2> <rest> ; )

Equivale a usar DUP C% -1 0 EQUAL case

2B25C num-1=case ( -1 🡺 :: -1 <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

La condición para que el test sea TRUE es que en la pila se encuentre un -1 real o un -1 complejo.

Equivale a

DUP %-1 EQUAL OVER C% -1 0 EQUAL OR case

21.4.6 Tests con Objetos Meta

**Direcc. Nombre Descripción**

2AFFB MEQ1stcase ( meta&ob1 ob2 🡺 :: meta&ob1 <ob1> ; )

( meta&ob1 ob2 🡺 :: meta&ob1 <ob2> <rest> ; )

La condición para que el test sea TRUE es que ob1 y ob2 sean el mismo objeto (comparación con EQ).

Por ejemplo, este programa retorna %11 %12 %13 BINT3 %20

::

%11 %12 %13 BINT3

%13

MEQ1stcase

%20

%21

%22

;

Este otro programa retorna %11 %12 %13 BINT3 %21 %22

::

%11 %12 %13 BINT3

%100

MEQ1stcase

%20

%21

%22

;

2AF37 AEQ1stcase ( meta&ob1 🡺 :: meta&ob1 <ob2> ; )

( meta&ob1 🡺 :: meta&ob1 <ob3> <rest> ; )

Hace lo mismo que MEQ1stcase pero con ob2 después del comando.

Por ejemplo, este programa retorna %11 %12 %13 BINT3 %20

::

%11 %12 %13 BINT3

AEQ1stcase

%13

%20

%21

%22

;

2B01B MEQopscase ( meta1&ob1 meta2&ob2 ob3

🡺 :: meta1&ob1 meta2&ob2 <ob1> ; )

( meta1&ob1 meta2&ob2 ob3

🡺 :: meta1&ob1 meta2&ob2 <ob2> <rest>; )

La condición para que el test sea TRUE es que ob1, ob2 y ob3 sean el mismo objeto (comparación con EQ).

Por ejemplo, este programa

retorna %11 %12 %13 BINT3 %17 %13 BINT2 %20

::

%11 %12 %13 BINT3

%17 %13 BINT2

%13

MEQopscase

%20

%21

%22

;

**Direcc. Nombre Descripción**

2B06A AEQopscase ( meta1&ob1 meta2&ob2

🡺 :: meta1&ob1 meta2&ob2 <ob2> ; )

( meta1&ob1 meta2&ob2

🡺 :: meta1&ob1 meta2&ob2 <ob3> <rest> ; )

Hace lo mismo que MEQopscase pero con ob3 después del comando.

2B083 Mid1stcase ( meta&ob 🡺 :: meta&ob <ob1> ; )

( meta&ob 🡺 :: meta&ob <ob2> <rest>; )

La condición para que el test sea TRUE es que ob sea un id o un lam.

Por ejemplo, este programa retorna %11 ' LAM F BINT2 %13

::

%11 ' LAM F BINT2

Mid1stcase

%13

%23

%24

;

2AE32 M-1stcasechs ( Meta&(%<0) 🡺 :: Meta&ABS(%) <ob1> ; )

( Meta&xNEG 🡺 :: Meta <ob1> ; )

( Meta&ob 🡺 :: Meta&ob <ob2> <rest> ; )

La condición para que el test sea TRUE es que el último objeto del meta sea un número real negativo (halla el valor absoluto de ese número) o sea el comando xNEG (corta el meta).

Por ejemplo, este programa retorna %11 % 4. BINT2 %13

::

%11 %-4 BINT2

M-1stcasechs

%13

%23

%24

;

21.4.7 Tests con Objetos Cualesquiera

**Direcc. Nombre Descripción**

36EBB EQIT ( ob1 ob1 🡺 :: <ob1> <rest> ; )

( ob1 ob2 🡺 :: <ob2> <rest> ; )

Equivale a usar EQ IT

36F01 EQITE ( ob1 ob1 🡺 :: <ob1> <ob3> <rest> ; )

( ob1 ob2 🡺 :: <ob2> <rest> ; )

Equivale a usar EQ ITE

36D3A jEQcase ( ob1 ob1 🡺 :: <ob1> ; )

( ob1 ob2 🡺 :: <ob2> <rest> ; )

Equivale a usar EQ case

34999 EQcase ( ob1 ob1 🡺 :: ob1 <ob1> ; )

( ob1 ob2 🡺 :: ob1 <ob2> <rest> ; )

Equivale a usar OVER EQ case

Nota: debería llamarse OVEREQcase.

359F7 REQcase ( ob 🡺 :: ob <ob2> ; )

( ob 🡺 :: ob <ob3> <rest> ; )

EQcase con el otro objeto después del comando.

**Direcc. Nombre Descripción**

34920 EQcasedrop ( ob1 ob1 🡺 :: <ob1> ; )

( ob1 ob2 🡺 :: ob1 <ob2> <rest> ; )

Equivale a usar OVER EQ casedrop

Nota: debería llamarse OVEREQcasedrop

35A10 REQcasedrop ( ob 🡺 :: <ob2> ; )

( ob 🡺 :: <ob3> <rest> ; )

EQcasedrop con el otro objeto después del comando.

36D62 EQUALcase ( ob1 ob1 🡺 :: <ob1> ; )

( ob1 ob2 🡺 :: <ob2> <rest> ; )

Equivale a usar EQUAL case

36E7F EQUALNOTcase ( ob1 ob1 🡺 :: <ob2> <rest> ; )

( ob1 ob2 🡺 :: <ob1> ; )

Equivale a usar EQUAL NOT case

36D08 EQUALcasedrp ( ob ob1 ob2 🡺 :: <ob1> ; )

( ob ob1 ob2 🡺 :: ob <ob2> <rest> ; )

Equivale a usar EQUAL casedrop

2AD81 EQUALcasedrop ( ob1 ob2 🡺 :: <ob1> ; )

( ob1 ob2 🡺 :: ob1 <ob2> <rest> ; )

Equivale a usar OVER EQUAL casedrop

29E99 tok=casedrop ( $ $' 🡺 :: <ob1> ; )

( $ $' 🡺 :: $ <ob2> <rest> ; )

Para dos cadenas equivale a usar OVER EQUAL casedrop

Nota: debería llamarse OVERtok=casedrop.

2ADBD nonopcase ( prog 🡺 :: prog <ob2> <rest> ; )

( ob 🡺 :: ob <ob1> ; )

La condición para que el test sea TRUE es que el objeto no sea un programa.

Equivale a usar DUP TYPECOL? NOT case

Equivale a usar DUPTYPECOL? NOT case

2B0CC idntcase ( id 🡺 :: id <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

La condición para que el test sea TRUE es que el objeto sea un id.

Equivale a usar DUP TYPEIDNT? case

Equivale a usar DUPTYPEIDNT? case

36E93 dIDNTNcase ( id 🡺 :: id <ob2> <rest> ; )

( ob 🡺 :: ob <ob1> ; )

La condición para que el test sea TRUE es que el objeto no sea un id.

Equivale a usar DUP TYPEIDNT? NOT case

Equivale a usar DUPTYPEIDNT? NOT case

2B0EF idntlamcase ( id/lam 🡺 :: id/lam <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

La condición para que el test sea TRUE es que el objeto sea un id o un lam.

Equivale a usar DUP TYPEIDNT? OVER TYPELAM? OR case

Equivale a usar DUP FLASHPTR TYPEIDNTLAM? Case

**Direcc. Nombre Descripción**

36DF3 REALcase ( % 🡺 :: <ob1> ; )

( ob 🡺 :: <ob2> <rest> ; )

La condición para que el test sea TRUE es que el objeto sea un número real.

Equivale a usar TYPEREAL? case

36EA7 dREALNcase ( % 🡺 :: % <ob2> <rest> ; )

( ob 🡺 :: ob <ob1> ; )

La condición para que el test sea TRUE es que el objeto no sea un número real.

Equivale a usar DUP TYPEREAL? NOT case

36E07 dARRYcase ( [] 🡺 :: [] <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

La condición para que el test sea TRUE es que el objeto sea un arreglo.

Equivale a usar DUP TYPEARRY? case

36E43 dLISTcase ( {} 🡺 :: {} ob1 ; )

( ob 🡺 :: ob <ob2> <rest> ; )

La condición para que el test sea TRUE es que el objeto sea una lista.

Equivale a usar DUP TYPELIST? case

260C6 NOTLISTcase ( {} 🡺 :: {} <ob2> <rest> ; )

( ob 🡺 :: ob <ob1> ; )

La condición para que el test sea TRUE es que el objeto no sea una lista.

Equivale a usar DUP TYPELIST? NOT case

260D0 NOTSECOcase ( seco 🡺 :: seco <ob2> <rest> ; )

( ob 🡺 :: ob <ob1> ; )

La condición para que el test sea TRUE es que el objeto no sea un programa.

Equivale a usar DUP TYPECOL? NOT case

Equivale a usar DUPTYPECOL? NOT case

Este comando hace lo mismo que nonopcase

260CB NOTROMPcase ( romp 🡺 :: romp <ob2> <rest> ; )

( ob 🡺 :: ob <ob1> ; )

La condición para que el test sea TRUE es que el objeto no sea un rompointer.

Equivale a usar DUP TYPEROMP? NOT case

Equivale a usar DUPTYPEROMP? NOT case

2ADE0 numb1stcase ( %/C%/[]/LNKARRY 🡺 :: %/C%/[]/LNKARRY <ob1> ; )

( ob 🡺 :: ob <ob2> <rest> ; )

La condición para que el test sea TRUE es que el objeto sea número real, número complejo, arreglo o arreglo vinculado.

Equivale a usar

DUPTYREAL? OVER TYPECMP? OR OVER TYPEARRY? OR OVER XEQTYPE? SWAPDROP %23 %= OR case

21.4.8 Miscelánea

**Direcc. Nombre Descripción**

36F65 UserITE ( #set 🡺 :: <ob1> <ob3> <rest> ; )

( #clr 🡺 :: <ob2> <rest> ; )

Equivale a usar TestUserFlag ITE

36F79 SysITE ( #set 🡺 :: <ob1> <ob3> <rest> ; )

( #clr 🡺 :: <ob2> <rest> ; )

Equivale a usar TestSysFlag ITE

36C4F caseDoBadKey ( T 🡺 :: DoBadKey ; )

( F 🡺 :: <ob1> <rest> ; )

Si el flag es TRUE, pasa por alto el resto del programa y emite un sonido.

Equivale a usar case DoBadKey

aka: caseDEADKEY

36C36 caseDrpBadKy ( ob T 🡺 :: DoBadKey ; )

( ob F 🡺 :: ob <ob1> <rest> ; )

Si el flag es TRUE, borra ob, pasa por alto el resto del programa y emite un sonido.

Equivale a usar case DropBadKey

Equivale a usar casedrop DoBadKey

361B2 caseERRJMP ( T 🡺 :: ERRJMP ; )

( F 🡺 :: <ob1> <rest> ; )

Si el flag es TRUE, hace ERRJMP

Equivale a usar case ERRJMP

36B53 caseSIZEERR ( T 🡺 :: SETSIZEERR ; )

( F 🡺 :: <ob1> <rest> ; )

Si el flag es TRUE, hace SETSIZEERR

Equivale a usar case SETSIZEERR

36B67 NcaseSIZEERR ( T 🡺 :: <ob1> <rest> ; )

( F 🡺 :: SETSIZEERR; )

Si el flag es FALSE, hace SETSIZEERR

Equivale a usar NOT case SETSIZEERR

36BAA NcaseTYPEERR ( T 🡺 :: <ob1> <rest> ; )

( F 🡺 :: SETTYPEERR ; )

Si el flag es FALSE, hace SETTYPEERR

Equivale a usar NOT case SETTYPEERR

25EEE NoEdit?case ( 🡺 :: <ob1> ; )

( 🡺 :: <ob2> <rest> ; )

La condición para que el test sea TRUE es que no exista una línea de edición activa.

Parecido a usar EditLExistS? NOT case

36E57 EditExstCase ( 🡺 :: <ob1> ; )

( 🡺 :: <ob2> <rest> ; )

La condición para que el test sea TRUE es que exista una línea de edición activa.

Equivale a usar EditLExistS? case

2BE36 (AlgebraicModecase) ( 🡺 :: <ob1> ; )

( 🡺 :: <ob2> <rest> )

La condición para que el test sea TRUE es que la calculadora esté en modo algebraico.

Equivale a usar BINT95 TestSysFlag case

Capítulo 22  
Bucles (Loops)

Como en User RPL, en System RPL también hay dos tipos de bucles: bucles indefinidos y bucles definidos.

Un bucle indefinido es aquel para el cual no se sabe con anterioridad, cuantas veces será ejecutado. Será repetido hasta que se cumpla una condición específica.

Un bucle definido, por otra parte, es ejecutado un número de veces ya especificadas antes de su inicio. Sin embargo, en System RPL puedes cambiar el número de veces que se repite el bucle mientras se está ejecutando.

En las descripciones de abajo, los elementos entre < > pueden consistir de varios objetos, a menos que se indique lo contrario.

22.1 Bucles Indefinidos

En System RPL, los bucles indefinidos pueden hacerse de tres maneras.

A) El primer tipo es el bucle WHILE, el cual es creado de esta forma:

BEGIN

<test>

WHILE

<objetos\_bucle>

REPEAT

Esta clase de bucle ejecuta <test>, y si el test es TRUE, entonces <objetos\_bucle> es ejecutado, y el proceso empieza nuevamente. Si el test retorna FALSE, entonces la ejecución pasa a estar después de REPEAT. Si la primera vez que se evalúa al test, este retorna FALSE, el bucle nunca será ejecutado. Este bucle requiere que <objetos\_bucle> sea un objeto único. La mayoría de las veces este es un objeto programa. Sin embargo, Debug4x siempre adiciona al código que escribes los delimitadores :: y ;.

B) El segundo tipo de bucle indefinido es el bucle UNTIL. El cual es creado de esta manera:

BEGIN

<obj\_bucle&test>

UNTIL

Este bucle siempre es ejecutado por lo menos una vez. El comando UNTIL espera un flag. Si este es FALSE, entonces <obj\_bucle&test> es ejecutado nuevamente. Si el flag es TRUE, la ejecución pasa a estar después de UNTIL.

C) También hay un tercer tipo de bucle indefinido, el bucle AGAIN:

BEGIN

<obj\_bucle>

AGAIN

Este bucle no tiene un test. Para terminarlo, debe ocurrir un error, o la pila de retornos debe ser directamente manipulada. Esto es útil si el código del bucle contiene varios lugares diferentes en los cuales decisiones sobre repetir el bucle o salir de el tienen que ser hechas.

22.1.1 Como Funcionan los Bucles Indefinidos

Los bucles indefinidos están formados por combinaciones de los comandos BEGIN, WHILE, REPEAT, UNTIL y AGAIN. Estos comandos no tienen nada especial, pues son como cualquier otro comando, pero cuando se combinan permiten hacer bucles. Estos funcionan manipulando el runstream y la pila de retornos, por lo cual asegurate de comprender esos conceptos (ver sección 20.1 si tienes dudas) para entender esta sección.

**Comando Pila y Acción Individual**

BEGIN ( 🡺 )

Copia el resto del runstream al primer nivel de la pila de retornos.

Esto significa que después de que el programa actual termine, un salto sera hecho al objeto que estaba justo después de BEGIN, ejecutando de este modo el resto del programa actual una vez más.

Hace exactamente lo mismo que el comando IDUP (sección 20.2).

UNTIL ( flag 🡺 )

Si el flag es TRUE, borra el primer nivel de la pila de retornos.

Si el flag es FALSE, ignora el resto del programa actual y duplica el primer nivel de la pila de retornos.

WHILE ( flag 🡺 )

Si el flag es TRUE, no hace nada.

Si el flag es FALSE, borra el primer nivel de la pila de retornos y pasa por alto los siguientes 2 objetos del runstream.

REPEAT ( 🡺 )

Ignora el resto del programa actual y duplica el primer nivel de la pila de retornos.

AGAIN ( 🡺 )

Ignora el resto del programa actual y duplica el primer nivel de la pila de retornos.

Gracias a esta explicación, ahora podemos comprender como funcionan los bucles indefinidos, y también el porque el bucle BEGIN...WHILE...REPEAT requiere un objeto único entre WHILE y REPEAT.

22.2 Bucles Definidos

Los bucles definidos son creados con los comandos DO y LOOP (u otros comandos equivalentes). El comando DO toma dos bints de la pila, que representan el valor de parada y el valor inicial. El valor inicial es guardado como el índice actual, el cual puede ser llamado con el comando INDEX@. El valor de parada puede ser llamado con el comando ISTOP@. Puedes fijar un nuevo valor para cada uno de ellos con los comandos INDEXSTO e ISTOPSTO, respectivamente.

La contraparte del comando DO es el comando LOOP o también +LOOP. El comando LOOP incrementa el valor del índice en uno, y verifica si el nuevo valor es mayor o igual que el valor de parada, finalizando el bucle si esto se cumple. De lo contrario, el bucle es ejecutado nuevamente. El comando +LOOP funciona de maner similar, pero incrementa el índice en una cantidad igual al bint que está en el nivel 1 de la pila.

La forma estándar de un bucle DO/LOOP es:

#parada #inicio DO

<objetos\_bucle>

LOOP

En este caso, <objetos\_bucle> es ejecutado para cada valor del índice desde #inicio hasta #parada-1. Nota que el valor de parada es mayor que el valor que sería usado en User RPL. Debes prestar atención a eso. También, el valor de parada viene antes del valor inicial.

La forma estándar de un bucle DO/+LOOP es:

#parada #inicio DO

<objetos\_bucle>

#incremento +LOOP

Puedes usar varios comandos que producen bucles DO, tal como ONE\_DO. Estos son listados en la sección de referencia de abajo.

Aquí un ejemplo de un bucle simple, el cual retorna los bints #1h, #2h, #3h y #4h a la pila.

::

BINT5 BINT1

DO

INDEX@

LOOP

;

Este programa podría ser cambiado a:

::

BINT5 ONE\_DO

INDEX@

LOOP

;

22.2.1 Como Funciona un Bucle Definido

Si tienes alguna familiaridad con conceptos como la pila de retornos y el runstream (descritos en la sección 20.1), esta sección te explicarará como funciona un bucle DO.

Cuando el comando DO es ejecutado, este pone el intérprete del puntero (el cual apunta al primer objeto que se encuentra después de DO) a la pila de retornos. Este también crea un entorno DoLoop, guardando el valo inicial y el valor de parada.

La ejcución continua normalmente, evaluando todos los comandos entre DO y LOOP.

Cuando LOOP es ejecutado, este incrementa el valor actual del índice del entorno DoLoop más reciente. Si este es mayor o igual que el valor de parada de ese entorno, el entorno es destruido y el nivel 1 de la pila de retornos es quitado. Esto remueve el puntero al primer objeto después de DO, y la ejecución continúa normalmente después de LOOP. Si el valor es más pequeño que el valor de parada, entonces el intérprete del puntero es fijado al valor más alto en la pila de retornos, causando que la ejecución se reinicie al primer objeto después de DO.

22.3 Referencia

22.3.1 Bucles Indefinidos

**Direcc. Nombre Descripción**

0716B IDUP ( 🡺 )

Pone el intérprete del puntero en la pila de retornos.

071A2 BEGIN ( 🡺 )

Pone el intérprete del puntero en la pila de retornos.

071AB AGAIN ( 🡺 )

Fija el intérprete del puntero en el primer nivel de la pila de retornos, sin quitarlo.

071E5 REPEAT ( 🡺 )

Fija el intérprete del puntero en el primer nivel de la pila de retornos, sin quitarlo.

071C8 UNTIL ( flag 🡺 )

Si es FALSE, hace AGAIN. De lo contrario, RDROP.

3640F NOT\_UNTIL ( flag 🡺 )

NOT luego UNTIL.

35B96 #0=UNTIL ( # 🡺 # )

Debería llamarse DUP#0=UNTIL.

071EE WHILE ( flag 🡺 )

Si es TRUE no hace nada.

Si es FALSE hace RDROP luego 2SKIP\_.

36428 NOT\_WHILE ( flag 🡺 )

NOT luego WHILE.

36441 DUP#0<>WHILE ( # 🡺 )

DUP#0<> luego WHILE.

22.3.2 Bucles Definidos

**Direcc. Nombre Descripción**

073F7 DO ( #parada #inicio 🡺 )

073C3 ZERO\_DO ( #parada 🡺 )

364C8 DUP#0\_DO ( #parada 🡺 #fin )

073CE ONE\_DO ( #parada 🡺 )

073DB #1+\_ONE\_DO ( #parada 🡺 )

364E1 toLEN\_DO ( {} 🡺 {} )

Desde BINT1 hasta #elementos.

Equivale a hacer DUPLENCOMP #1+\_ONE\_DO

07334 LOOP ( 🡺 )

073A5 +LOOP ( # 🡺 )

Incrementa el índice por el número especificado.

364AF DROPLOOP ( ob 🡺 )

36496 SWAPLOOP ( ob1 ob2 🡺 ob2 ob1 )

07221 INDEX@ ( 🡺 # )

Llama al valor del contador del loop más reciente.

3645A DUPINDEX@ ( ob 🡺 ob # )

3646E SWAPINDEX@ ( ob1 ob2 🡺 ob2 ob1 # )

36482 OVERINDEX@ ( ob1 ob2 🡺 ob1 ob2 ob1 # )

367D9 INDEX@#- ( # 🡺 #' )

**Direcc. Nombre Descripción**

07270 INDEXSTO ( # 🡺 )

Guarda el bint en el contador del loop más reciente.

07249 ISTOP@ ( 🡺 # )

Llama al valor de parada del loop más reciente.

07295 ISTOPSTO ( # 🡺 )

Guarda el bint como valor de parada del loop más reciente.

283FC ISTOP-INDEX ( 🡺 # )

07258 JINDEX@ ( 🡺 # )

Llama al valor del contador del segundo loop más reciente.

072AD JINDEXSTO ( # 🡺 )

Guarda el bint en el contador del segundo loop más reciente.

07264 JSTOP@ ( 🡺 # )

Llama al valor de parada del segundo loop más reciente.

072C2 JSTOPSTO ( # 🡺 )

Guarda el bint como valor de parada del segundo loop más reciente.

3709B ExitAtLOOP ( 🡺 )

Este comando es para salir del loop más reciente.

Al ejecutarlo no se sale del loop inmediatamente.

El comando sólo guarda #0 como valor de parada del loop más reciente, de modo que todos los objetos hasta el siguiente LOOP serán evaluados.

Equivale a usar ZERO ISTOPSTO

aka: ZEROISTOPSTO

22.4 Ejemplos

Ejemplo 1 Bucle Definido

**Usando dos bucles DO/LOOP anidados.**

En este ejemplo se usan dos bucles DO, uno dentro de otro.

\* Este programa muestra en la pantalla cada uno de los índices

\* de los dos bucles, todas las veces que se ejecuta el bucle

\* J: 100, 200, 300, 400 ( CAPITULOS )

\* I: 1, 2, 3 ( SECCIONES )

ASSEMBLE

CON(1) 8 ( Le dice al parseador que el comando es 'No algebraico' )

RPL

xNAME Ej2Bucles ( -> )

::

CK0 ( ) ( NINGUN ARGUMENTO ES REQUERIDO )

CLEARLCD ( ) ( LIMPIA LA PANTALLA )

401 100 DO ( ) ( BUCLE J: 100, 200, 300, 400 )

4 1 DO ( ) ( BUCLE I: 1, 2, 3 )

"CAPITULO " JINDEX@ #>$ &$ DISPROW1 ( ) ( MUESTRA VALOR DE J )

"SECCION " INDEX@ #>$ &$ DISPROW2 ( ) ( MUESTRA VALOR DE I )

"" FlashWarning ( ) ( MUESTRA MENSAJE RECTANG )

LOOP ( ) ( INCREMENTA I )

100 +LOOP ( ) ( INCREMENTA J )

;



Ejemplo 2 Bucle Definido

**Usando un bucle DO/LOOP tantas veces como elementos tiene una lista.**

En este ejemplo se usa el comando toLEN\_DO para repetir un bucle tantas veces como elemetos tiene el compuesto ubicado en el nivel 1.

\* Este programa comprueba si todos los elementos de una lista

\* son números reales.

\* Si la lista está vacía retorna FALSE.

\* Si la lista tiene sólo reales, retorna TRUE.

\* Si la lista tiene algún objeto no real, retorna FALSE

NULLNAME ListaReal? ( {} -> flag )

:: ( {} )

DUPNULL{}? ( {} flag )

case

DROPFALSE ( sale con FALSE )

( {} )

TRUE 1LAMBIND ( {} ) ( crea entorno temporal con LAM1=TRUE )

( {} )

toLEN\_DO

DUPINDEX@ ( {} {} #i ) ( duplica y retorna el índice )

NTHCOMPDROP ( {} obi ) ( retorna el objeto i-ésimo de la lista )

TYPEREAL? ( {} flag ) ( retorna TRUE si es real )

1GETLAM ( {} flag flag' ) ( retorna el valor del LAM1 )

AND ( {} flag'' )

1PUTLAM ( {} ) ( guarda en LAM1 )

LOOP

( {} )

DROP ( )

1GETABND ( flag ) ( retorna LAM1 y quita entorno temporal )

;

Ejemplo 3 Bucle Definido

**Haciendo que un bucle DO/LOOP no se vuelva a repetir.**

En este ejemplo se usa el comando ExitAtLOOP para hacer que el bucle DO/LOOP no se vuelva a repetir cuando encuentra un objeto no real en una lista.

\* Este programa comprueba si todos los elementos de una lista

\* son números reales.

\* Si la lista está vacía retorna FALSE.

\* Si la lista tiene sólo reales, retorna TRUE.

\* Si la lista tiene algún objeto no real, retorna FALSE

NULLNAME ListaReal? ( {} -> flag )

:: ( {} )

DUPNULL{}? ( {} flag )

case

DROPFALSE ( sale con FALSE )

( {} )

TRUE 1LAMBIND ( {} ) ( crea entorno temporal con LAM1=TRUE )

( {} )

toLEN\_DO

DUPINDEX@ ( {} {} #i ) ( duplica y retorna el índice )

NTHCOMPDROP ( {} obi ) ( retorna el objeto i-ésimo de la lista )

TYPEREAL? ( {} flag ) ( retorna TRUE si es real )

NOT\_IT

:: ExitAtLOOP ( {} ) ( EL BUCLE NO SE VOLVERÁ A REPETIR )

FALSE ( {} F )

1PUTLAM ( {} ) ( guarda FALSE en LAM1 )

;

LOOP

( {} )

DROP ( )

1GETABND ( flag ) ( retorna LAM1 y quita entorno temporal )

;

Ejemplo 4 Bucle Definido

**Reiniciendo un bucle DO/LOOP.**

En este ejemplo se reinicia el bucle DO/LOOP, asignando un nuevo valor al índice (una unidad menos que #inicial) con el comando INDEXSTO.

En la última repetición del bucle (cuando #indice=#parada-1) se fija a #indice como cero (una unidad menos que #inicial=#1) usando ZERO INDEXSTO.

Si el #inicial hubiera sido #0, para reiniciar el bucle, debemos hacer MINUSONE INDEXSTO.

\* Este programa muestra los elementos de la lista {41,42,43,44}

\* uno por uno durante 10 segundos.

\* Si pasa más tiempo, se muestra el mensaje "TIEMPO EXCEDIDO" y el

\* bucle no se vuelve a repetir.

\* Si se termina de mostrar todos los elementos de la lista, el

\* bucle se reinicia para mostrar nuevamente los elementos.

ASSEMBLE

CON(1) 8 ( Le dice al parseador que el comando es 'No algebraico' )

RPL

xNAME ReinicioDO ( -> )

::

CK0 ( ) ( NINGUN ARGUMENTO ES REQUERIDO )

CLKTICKS ( hxs1 ) ( hora del sistema en ticks )

1LAMBIND ( ) ( lo guarda en LAM1 en nuevo entorno temp. )

{ 41. 42. 43. 44. } ( {} )

toLEN\_DO

CLKTICKS ( {} hxs2 ) ( hora del sistema en ticks )

1GETLAM ( {} hxs2 hxs1 ) ( llama a la hora inicial )

bit- ( {} hxs' ) ( resta 2 hxs: interv tiempo en ticks )

HXS>% ( {} % ) ( interv tiempo en ticks )

81920. ( {} % %81920 ) ( 81920 = 8192o10 = 10segundos )

%> ( {} flag ) ( TRUE, si pasaron más de 10 s )

ITE

:: ExitAtLOOP ( {} ) ( EL BUCLE NO SE VOLVERÁ A REPETIR )

"TIEMPO EXCEDIDO" ( {} $ )

FlashWarning ( {} )

;

:: DUPINDEX@ ( {} {} #i ) ( duplica y retorna el índice )

NTHCOMPDROP ( {} obi ) ( retorna el objeto i-ésimo de la lista )

DO>STR ( {} $i ) ( lo convierte a cadena )

FlashWarning ( {} ) ( muetra el elmento iésimo en pantalla )

ISTOP-INDEX ( {} #parada-#indice )

#1= ( {} flag )

NOT?SEMI ( si es FALSE, ignora el resto del runstream )

( {} )

ZERO ( {} #0 )

INDEXSTO ( {} ) ( #0 será el nuevo índice: reiniciará bucle )

;

LOOP ( aumenta #index en 1 )

( {} )

DROP ( )

ABND ( )

;

Ejemplo 5 Bucle Definido

**Finalizando INMEDIATAMENTE un bucle DO/LOOP.**

En este ejemplo, el bucle DO/LOOP se termina inmediatamente al ejecutarse el subprograma ExitAtLOOP\_AHORA.

Para conseguir esto, el subprograma ExitAtLOOP\_AHORA fija al índice un valor igual a #parada y lleva la ejecución al objeto que se encuentra después de LOOP.

\* Hace lo mismo que el ejm 4, pero cambia ExitAtLOOP por ExitAtLOOP\_AHORA

ASSEMBLE

CON(1) 8 ( Le dice al parseador que el comando es 'No algebraico' )

RPL

xNAME ReinicioDO ( -> )

:: CK0 ( ) ( NINGUN ARGUMENTO ES REQUERIDO )

CLKTICKS ( hxs1 ) ( hora del sistema en ticks )

1LAMBIND ( ) ( lo guarda en LAM1 en nuevo entorno temporal )

{ 41. 42. 43. 44. } ( {} )

toLEN\_DO

CLKTICKS ( {} hxs2 ) ( hora del sistema en ticks )

1GETLAM ( {} hxs2 hxs1 ) ( llama a la hora inicial )

bit- ( {} hxs' ) ( resta 2 hxs: interv tiempo en ticks )

HXS>% ( {} % ) ( interv tiempo en ticks )

81920. ( {} % %81920 ) ( 81920 = 8192o10 = 10segundos )

%> ( {} flag ) ( TRUE, si pasaron más de 10 s )

ITE

:: ExitAtLOOP\_AHORA ( {} ) ( EL BUCLE TERMINA INMEDIATAMENTE )

"TIEMPO EXCEDIDO" ( {} $ ) ( ESTO YA NUNCA SE EJECUTARÁ )

FlashWarning ( {} ) ( ESTO YA NUNCA SE EJECUTARÁ )

;

:: DUPINDEX@ ( {} {} #i ) ( duplica y retorna el índice )

NTHCOMPDROP ( {} obi ) ( retorna el objeto i-ésimo de la lista )

DO>STR ( {} $i ) ( lo convierte a cadena )

FlashWarning ( {} ) ( muetra el elmento iésimo en pantalla )

ISTOP-INDEX ( {} #parada-#indice )

#1= ( {} flag )

NOT?SEMI ( si es FALSE, ignora el resto del runstream )

( {} )

ZERO ( {} #0 )

INDEXSTO ( {} ) ( #0 será el nuevo índice: reiniciará bucle )

;

LOOP ( aumenta #index en 1 )

( {} )

DROP ( )

ABND ( )

;

\* Finaliza un bucle DO/LOOP inmediatamente, cuando

\* este termina con LOOP, +LOOP, SWAPLOOP o DROPLOOP

NULLNAME ExitAtLOOP\_AHORA ( -> )

:: R@ ( prog )

:: DUP ( prog prog )

' LOOP ( prog prog LOOP )

EQUALPOSCOMP ( prog #POS/#0 )

DUP#0<> ( prog #POS/#0 flag )

case SWAPDROP

( prog #POS/#0 )

DROPDUP ( prog prog )

' +LOOP ( prog prog +LOOP )

EQUALPOSCOMP ( prog #POS/#0 )

DUP#0<> ( prog #POS/#0 flag )

case SWAPDROP

( prog #POS/#0 )

DROPDUP ( prog prog )

' SWAPLOOP ( prog prog SWAPLOOP )

EQUALPOSCOMP ( prog #POS/#0 )

DUP#0<> ( prog #POS/#0 flag )

case SWAPDROP

( prog #POS/#0 )

DROP ( prog )

' DROPLOOP ( prog DROPLOOP )

EQUALPOSCOMP ( #POS/#0 )

;

DUP#0= ( #POS/#0 flag )

COLAITE

:: DROP RDROP COLA ExitAtLOOP\_AHORA ;

:: R@ RDROP SWAP #1+ OVER LENCOMP SUBCOMP ISTOP@ INDEXSTO [LOOP] >R ;

;

Ejemplo 6 Bucle Indefinido

**Finalizando INMEDIATAMENTE un bucle AGAIN.**

En este ejemplo, el bucle AGAIN se termina inmediatamente al ejecutarse el subprograma SaleBucleAGAIN\_AHORA.

Para conseguir esto, el subprograma SaleBucleAGAIN\_AHORA manipula la pila de retornos y el runstream, como indica la explicación de la derecha luego de cada acción.

\* No se requieren argumentos.

\* Edita un arreglo real con el ESCRITOR DE MATRICES.

\* Si uno se retira de la edición, retorna FALSE.

\* Si uno acepta con ENTER, retorna el anuevo arreglo real y TRUE

\* El arreglo real debe tener 2 columnas y por lo menos 2 filas,

\* si no es así se muestra un mensaje y el bucle se repite otra

\* vez, regresandote al escritor de matrices con el arreglo real

\* que escribiste.

\* Este programa llama al NULLNAME SaleBucleAGAIN\_AHORA, que

\* permite salir de un bucle AGAIN, inmediatamente y en

\* cualquier momento

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME EditaMatriz ( -> [[%]] T // F )

::

CK0 ( ) ( No se requieren argumentos )

ARRY [ 13. 15. ] ( [%] )

BEGIN

RunSafeFlags

:: BINT91 ClrSysFlag

FLASHPTR DoOldMatrixReal

; ( RealArry T // F )

NOT\_IT

:: FALSE ( F )

SaleBucleAGAIN\_AHORA ( F ) ( BUCLE TERMINA AHORA )

;

( RealArry )

DUP

FLASHPTR MDIMS ( RealArry #f #c T // RealArry #elem F )

ITE

:: ( RealArry #f #c )

#2= ( RealArry #f flag )

SWAP ( RealArry flag #f )

BINT1 #> ( RealArry flag flag' )

AND ( RealArry flag'' )

ITE

:: ( RealArry )

TRUE ( RealArry T )

SaleBucleAGAIN\_AHORA ( RealArry T ) ( BUCLE TERMINA AHORA )

;

:: ( RealArry )

"INGRESA ARREGLO REAL\0ANºcolumn = 2\0ANºfilas > 1"

FlashWarning ( RealArry )

;

;

:: ( RealArry #elem )

DROP ( RealArry )

"No ingreses vector\0AINGRESA ARREGLO REAL\0ANºcolumn = 2\0ANºfilas > 1"

FlashWarning ( RealArry )

;

AGAIN

CLEARLCD ( [[%]] T // F ) ( limpia la pantalla )

"SALISTE DEL BUCLE"

FlashWarning

( [[%]] T // F )

;

\* Permite salir de un bucle AGAIN, inmediatamente

\* y en cualquier momento

NULLNAME SaleBucleAGAIN\_AHORA ( -> )

::

R@ ( prog )

' AGAIN ( prog AGAIN )

EQUALPOSCOMP ( #POS/#0 )

DUP#0= ( #POS/#0 flag )

COLAITE

:: ( #0 )

DROP ( )

RDROP ( ) ( QUITA 1º nivel de pila de retornos )

( pues no contiene a AGAIN )

COLA ( ) ( EJECUTA SGTE OBJ EN EL PROG DE ARRIBA )

SaleBucleAGAIN\_AHORA ( )

;

:: ( #POS )

R@ ( #POS prog )

RDROP ( #POS prog ) ( QUITA 1º nivel de pila de )

( retornos [contiene a AGAIN] )

RDROP ( #POS prog ) ( QUITA sgte nivel de pila de )

( retornos [obj después de BEGIN] )

SWAP ( prog #POS )

#1+ ( prog #POS+1 )

OVER ( prog #POS+1 prog )

LENCOMP ( prog #POS+1 #n )

SUBCOMP ( prog' ) ( objs situados después de AGAIN )

>R ( ) ( pone en 1º nivel de pila de retornos )

;

;

Capítulo 23  
Manejo de Errores

Cuando un error ocurre en un programa de System RPL, normalmente el programa es abortado y un rectángulo es mostrado en la pantalla con el mensaje de error. Sin embargo, a veces es deseable que el programa atrape al error para que la ejecución del programa continue, o quizás para mostrar la ocurrencia del error de una forma diferente.

Otras veces el programa necesita generar un error. Por ejemplo, si el usuario ingreso argumentos inválidos para el programa, este debería ser abortado y se puede generar el mensaje de error “Argumento incorrecto”, en lugar de continuar el programa y poner la máquina en peligro de un crash.

23.1 Atrapando Errores

Puedes interceptar la ejecución de un error generado en tu programa, usando la siguiente estructura:

::

...

ERRSET

:: <objetos sospechosos> ;

ERRTRAP

:: <código a ejecutarse si error ocurre> ;

...

;

Si <objetos sospechosos> y/o <código a ejecutarse si error ocurre> son un único objeto, entonces no es necesario incluirlos dentro de un objeto programa.

Esto funciona de la siguiente manera:

1) Primero se ejecuta <objetos sospechosos>

a) Si un error ocurre ahí, entonces el programa pasa a ejecutar <código a ejecutarse si error ocurre>.

b) Si ningún error ocurre ahí, entonces <código a ejecutarse si error ocurre> no se ejecutará.

2) Después se evalúa el resto del programa.

La acción de <código a ejecutarse si error ocurre> es completamente flexible. El número del error actual puede ser llamado con el comando ERROR@, y tu programa puede hacer diferentes acciones basadas en el tipo de error que ocurrió. Los mensajes de error y sus números respectivos los puedes encontrar en el apéndice E.

23.1.1 La Palabra de Protección (Protection Word)

Cada entorno temporal (ver capítulo 19), entorno DO/LOOP (capítulo 22) y nivel de pila virtual (capítulo 24) tiene un “protection word”.

El propósito de esto es permitir que el subsitema de manejo de errores pueda distinguir cuales entornos fueron creados antes del proceso de buscar atrapar al error, y cuales después. De esta manera, todos los entornos que fueron creados después de iniciado el intento de atrapar el error, serán borrados si sucede un error. Por ejemplo, considerar el siguiente código:

::

...

1LAMBIND

...

TEN ZERO\_DO

ERRSET

::

...

1LAMBIND

...

BINT5 ONE\_DO

<objeto sospechoso está aquí>

LOOP

ABND

...

;

ERRTRAP

:: <manejador de errores> ;

LOOP

...

ABND

;

Si un error es generado en el objeto sospechoso, entonces el error es atrapado. El bucle DO/LOOP más interno y el entorno temporal más interno serán borrados, gracias al “protection word”.

Cuando alguno de estos entornos es creado, su “protection word” es fijado como cero. El comando ERRSET incrementa el protection word de los entornos (uno de cada uno de los tres tipos) creados anteriormente. De esta manera, esos entornos tienen ahora su “protection word” mayor a cero (el protection Word fue inicializado como cero cuando fue creado).

Los comandos ERRTRAP y ERRJMP borran todos los entornos que tengan su “protection word” igual a cero (desde el más reciente hasta el más antiguo) hasta que encuentren un entorno (de cualquiera de los tres tipos) con un “protection word” distinto a cero. Estos entornos eran aquellos que ya existían antes de que se inicie el proceso de buscar capturar un error, porque ellos tienen sus valores incrementado por el comando ERRSET. De esta manera, todos los entornos creados después de iniciada la busqueda de errores (entornos con con su “protection word” igual a cero) serán borrados. Otro efecto de los comandos ERRTRAP y ERRJMP es que ellos decrementan el “protection word” de los entornos creados anteriormente (entornos con “protection word” mayor a cero), de tal forma que el proceso funcionará correctamente si hay varios niveles de anidamiento.

Otras conclusiones:

• Si dentro del código que se encuentra entre los comandos ERRSET y ERRTRAP, abres un entorno de cualquiera de los tres tipos, entonces también debes poner un comando para cerrar dicho entorno (por ejemplo LOOP, DropVStack y ABND) dentro de ese código.

• Dentro del código que se encuentra entre los comandos ERRSET y ERRTRAP no puedes cerrar algún entorno (de cualquiera de los tres tipos) creado antes de ese ERRSET.

23.2 Generando Errores

El subsistema de manejo de errores es invocado por el comando ERRJMP. Si un error es atrapado, entonces el manejador de errores es ejecutado. Pero si un error ocurres sin ser atrapado, entonces se ejecuta el manejador de errores por defecto.

En la mayoría de los casos, cuando se genera un error, permites que el manejador de errores por defecto trate con este. Esta acción por defecto hace un beep (si está desactivado el flag 56), y muestra una descripción del error dentro de un rectángulo.

El mensaje de error mostrado depende de dos cosas: el número del error, el cual define el texto del mensaje de error (como “Argumento incorrecto” o “Muy pocos argumentos”), y el nombre del último comando guardado.

Este nombre del ultimo comando es automáticamente guardado por los comandos de la forma CK<n> descritos en el capítulo 30. Como se menciona ahí, si estás estás escribiendo un programa que no es parte de una biblioteca, ningún nombre de comando se guarda, entonces no deberás usar un comando de la forma CK<n> porque un nombre feo sería mostrado.

Para definir al número de error, usa el comando ERRORSTO. Este comando espera un bint como argumento: el número del error. Los errores son listados en el apéndice E.

Hay algunos comandos que hacen este proceso de forma automática. Por ejemplo, el comando SETTYPEERROR genera el error “Argumento incorrecto”, que también se pudo generar haciendo # 202 ERRORSTO ERRJMP. Estos comandos son listados en la sección de referencia de abajo. También hay comandos como estos pero que generan errores del CAS y se describen en el capítulo 53.

Sin embargo, a veces es deseable generar un mensaje de error con cualquier texto que no esté en la lista de mensajes de error ya incorporada en la calculadora. Para hacer esto, primero necesitas guardar la cadena de texto con el comando EXITMSGSTO. Luego, guarda #70000 como el número del error (nota que este bint está ya incorporado en ROM con la palabra #EXITERR) con ERRORSTO. Finalmente, solo llama a ERRJMP.

Los procesos descritos arriba pueden ser simplificados usando los comandos DO#EXIT y DO$EXIT. El primero toma un bint como argumento, guarda ese número y llama a ERRJMP. El ultimo es usado con cadenas, este comando toma una cadena como argumento y hace las acciones descritas en el párrafo anterior. Sin embargo, ambas entradas también llaman al comando AtUserStack, el cual le dice al sistema de manejo de errores que mantenga la pila como está en ese momento (sin borrar ningún objeto). Por lo tanto, no debes usar estos comandos (DO#EXIT y DO$EXIT) si hay objetos en la pila (puestos por tu programa) que deberían ser borrados cuando un error ocurre. El borrado automático de objetos de la pila no ingresados por el usuario cuando un error ocurre será descrito con más detalle en la sección 30.1.

23.3 Referencia

23.3.1 Comandos Generales

**Direcc. Nombre Descripción**

26067 ERRBEEP ( 🡺 )

Solo hace un beep. Un sonido similar al que se escucha cuando se genera un error.

04D0E ERRORSTO ( # 🡺 )

Guarda un nuevo número de error.

04D33 ERRORCLR ( 🡺 )

Guarda cero como el núevo número del error.

04ED1 ERRJMP ( 🡺 )

Invoca al subsistema de manejo de errores.

04EA4 ABORT ( 🡺 )

Hace ERRORCLR y luego ERRJMP.

Se muestra el mensaje de error “Interrupted” y no se muestra el nombre del último comando guardado.

04CE6 ERROR@ ( 🡺 # )

Retorna el número del error actual.

26431 PTR 26431 ( 🡺 # )

Retorna el número del último error generado.

36883 ERROROUT ( # 🡺 )

Guarda un nuevo número de error y genera el error.

Hace :: ERRORSTO ERRJMP ;

25EAE DO#EXIT ( # 🡺 )

Guarda un nuevo número de error, llama a AtUserStack y genera el error.

Hace :: ERRORSTO AtUserStack ERRJMP ;

25EB0 DO%EXIT ( % 🡺 )

Similar a DO#EXIT, pero un número real es su argumento.

Equivale al comando **DOERR** de User RPL cuando en la pila hay un número real.

23.3.2 Atrapando Errores

**Direcc. Nombre Descripción**

04E5E ERRSET ( 🡺 )

Establece un nuevo proceso de buscar capturar errores. 04EB8 ERRTRAP ( 🡺 )

Cuando se atrapa un error, el objeto que está a continuación es evaluado.

Aún si ningún error ocurre, se removerán todos los entornos (temporales, DO/LOOP y de pila virtual) creados desde el último ERRSET.

23.3.3 Mensajes de Error Personalizados

**Direcc. Nombre Descripción**

04E07 GETEXITMSG ( # 🡺 $ )

Retorna EXITMSG (mensaje de error definido por el usuario).

El bint puede ser cualquier número y no afecta el resultado.

04E37 EXITMSGSTO ( $ 🡺 )

Guarda cadena como EXITMSG.

25EAF DO$EXIT ( $ 🡺 )

Guarda cadena como EXITMSG, #70000 como número de error, hace AtUserStack y genera el error.

Hace lo siguiente:

:: EXITMSGSTO #EXITERR ERRORSTO AtUserStack ERRJMP ;

Equivale al comando **DOERR** de User RPL cuando en la pila hay una cadena.

23.3.4 Tabla de Mensajes de Error

**Direcc. Nombre Descripción**

04D87 JstGETTHEMSG ( #error 🡺 $msj )

Consigue un mensaje desde la tabla de mensajes de error.

Para conseguir un mensaje desde una biblioteca, usa la fórmula: #error = #100\*#lib + #LibError

#lib es el número de la biblioteca.

#LibError es el número del error dentro de la biblioteca y puede estar entre #1 y #FF.

Cada biblioteca puede contener hasta 255 mensajes de error.

Por ejemplo, para conseguir los mensajes de error de la biblioteca 1240 (#4D8h) usa los bints:

# 4D801, #4D802, … , #4D8FF

04D64 GETTHEMESG ( #error 🡺 $msj )

Si el bint es #70000, hace GETEXITMSG.

De lo contrario, hace JstGetTHEMESG.

39332 (?GetMsg) ( #error 🡺 $msj )

( ob 🡺 ob )

Si el argumento es un bint, hace JstGETTHEMSG para retornar un mensaje de error.

Otros argumentos son retornados sin cambios.

04DD7 (SPLITmsg) ( #error 🡺 #LibError #lib )

Equivale a hacer: :: # 100 #/ ;

08130 (GETMSG) ( #lib 🡺 [$] T )

( #lib 🡺 ACPTR T )

( #lib 🡺 F )

Si la biblioteca está en el puerto cero, retorna un arreglo de cadenas con los mensajes de error de esa biblioteca y TRUE.

Si la biblioteca está en otro puerto, retorna un access pointer y TRUE.

De otro modo, retorna FALSE.

23.3.5 Comandos que Generan Errores

**Direcc. Nombre Descripción**

04FB6 SETMEMERR Error 001h

Genera el error "Memoria insuficiente"

(Insufficient Memory).

04FC2 (SETDIRRECUR) Error 002h

Genera el error "Directorio recursivo"

(Directory Recursion).

04FCE (SETLAMERR) Error 003h

Genera el error "Nombre local indefin."

(Undefined Local Name).

05016 SETROMPERR Error 004h

Genera el error "Nombre XLIB indefinido"

(Undefined XLIB Name).

04FAA (SETLBERR) Error 006h

Genera el error "Corte de corriente"

(Power Lost).

04FF2 SETPORTNOTAV Error 00Ah

Genera el error "Puerta no disponible"

(Port Not Available).

26134 SYNTAXERR Error 106h

Genera el error "Sintaxis incorrntos"

(Invalid Syntax).

26521 (SETUNDOERR) Error 124h

Genera el error "LAST STACK desactivado"

(LAST STACK Disabled).

260C1 NOHALTERR Error 126h

Genera el error "HALT no permitido"

(HALT Not Allowed).

26116 SETCIRCERR Error 129h

Genera el error "Referencia circular"

(Circular Referente).

262E2 SETSTACKERR Error 201h

Genera el error "Muy pocos argumentos"

(Too Few Arguments).

262DD SETTYPEERR Error 202h

Genera el error "Argumento incorrecto"

(Bad Argument Type).

262D8 SETSIZEERR Error 203h

Genera el error "Argumento:valor incorr"

(Bad Argument Value).

262E7 SETNONEXTERR Error 204h

Genera el error "Nombre no definido"

(Undefined Name).

2F458 SETIVLERR Error 304h

Genera el error "Resultado indefinido"

(Undefined Result).

2F37B SetIOPARErr Error C12h

Genera el error "IOPAR inválido"

(Invalid IOPAR).

3721C Sig?ErrJmp ( 🡺 )

Llama a ERRJMP si el número del último error ocurrido (llama con ERROR@) es alguno de esta lista: {13E 123 DFF}.

**Direcc. Nombre Descripción**

25F10 ederr ( 🡺 )

Manejador de errores para aplicaciones que usan savefmt1 y rstfmt1 para guardar y restaurar el actual formato de mostrar números.

Hace :: rstfmt1 ERRJMP ;

Capítulo 24  
La Pila Virtual (Virtual Stack)

La calculadora HP tiene una “Pila Virtual”. De hecho existe una pila de pilas (o metapila). La pila de la parte más alta (y en condiciones normales, la única) es la pila RPN, en la cual el usuario ingresa objetos y desde la cual los comandos toman y retornan sus argumentos. Esta pila será llamada “pila RPN”. El conjunto (o más específicamente, la pila) de pilas será llamada “Pila Virtual”, con las primeras letras en mayúscula.

Puedes poner toda la pila RPN (o parte de esta) en un nuevo nivel de la “Pila Virtual”. Un nivel de la Pila Virtual será llamado “pila virtual”, con las letras iniciales en minúscula. Después de poner la pila RPN en ese nuevo nivel de la Pila Virtual, puedes manipular este de cualquier manera, y puedes en cualquier momento restaurar el contenido previamente puesto en esa pila virtual. O también puedes crear un nuevo nivel en la Pila Virtual, teniendo de esta manera dos pilas virtuales, en adición a la pila RPN, la cual puede ser usada independientemente.

Cada una de estas pilas virtuales contiene una cantidad fija de objetos y también un contador del número de objetos de esa pila virtual. El número de objetos es determinado cuando la pila virtual es creada, y no es posible agregar más objetos después. Los comandos que retornan la pila virtual como un meta retornan este contador, los otros comandos no devuelven el contador. Cuando creas una nueva pila virtual puedes poner en esa, algunos objetos de la pila RPN (con los comandos cuyo argumento es un meta) o todos los objetos de la pila RPN. Puedes destruir el nivel más bajo de la pila virtual, ya sea desapareciendo su contenido, o retornando su contenido en la pila RPN como un meta o no (es decir, con el contador al final de los objetos o sin el), sin importar si ese nivel de la pila virtual fue creado a partir de un meta o a partir de todos lo objetos de la pila RPN.

La Pila virtual es usada en casi todas las aplicaciones de la calculadora HP. Es extremadamente útil (y realmente rápida) cuando quieres guardar inmediatamente una pila completa sin usar mucha memoria.

Por ejemplo, en los formularios de entrada se aplica la Pila Virtual en el uso de la línea de comandos: te permite escribir comandos en dicha línea y conseguir sus resultados. Si escribes allí DROP, conseguirás el error: “Muy pocos argumentos” aún si la pila no está vacía. Esto funciona así: antes de que la calculadora ejecute el comando, esta guarda la totalidad de la pila RPN en la Pila Virtual, luego ejecuta el comando. Una vez que el comando ha terminado de ser ejecutado, se restaura la pila virtual copiada.

La Pila Virtual está ubicada dentro de una cadena, la cual es el primer objeto en TEMPOB. Tiene una estructura similar a la pila de Variables Locales, y es protegida exactamente igual que las variables locales. Si atrapas un error, las pilas virtuales creadas entre ERRSET y ERRTRAP serán borradas automáticamente, igual como sucede con los bloques de variables locales (Ver sección 23.1.1 para más información). Puedes ver ejemplos de aplicación de la Pila Virtual al final de este capítulo y en la sección 35.9.

24.1 Viendo la Pila Virtual con Debug4x

En Debug4x, puedes ver el contenido de todos los niveles de la Pila Virtual en el Panel Virtual Stack que se encuentra en la ventana RPL Debugging.

Por ejemplo, si escribimos el código de la izquierda en el editor de Debug4x, fijamos un breakpoint (círculo rojo) en la posición indicada y ejecutamos el comando EjemploVStack desde el emulador, podremos ver en el Panel Virtual Stack, todos los niveles de la Pila Virtual y sus respectivos “protection word”.

En la sección de referencia de comandos de abajo, nos referimos a la pila virtual más reciente como “la del nivel más alto”.

|  |  |
| --- | --- |
| Ventana Editor | Ventana RPL Debugging |
|  |  |

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME EjemploVStack

::

CK0

11. 12. 13. 3 PushMetaVStack&Drop

"21" "22" "23" "24" 4 PushMetaVStack&Drop

ERRSET

:: { %31\_ %32\_ }

"HON"

2

PushMetaVStack&Drop

%5 %0 %/

DropVStack

;

ERRTRAP

:: "División entre cero" FlashWarning ;

DropVStack

DropVStack

;

24.2 Referencia

24.1.1 Comandos Generales

**Direcc. Nombre Descripción**

26265 PushMetaVStack ( obn..ob1 #n 🡺 obn..ob1 #n )

Pila Virtual:

( 🡺 [obn..ob1] )

Pone los objetos del meta como una nueva pila virtual.

La pila RPN permanece sin cambios.

25F1D PushMetaVStack&Drop ( obn..ob1 #n 🡺 )

Pila Virtual:

( 🡺 [obn..ob1] )

Hace PushMetaVStack y luego borra el meta de la pila RPN.

25F19 PopMetaVStack ( 🡺 obn..ob1 #n )

Pila Virtual:

( [obn..ob1] 🡺 )

Traslada el contenido de la pila virtual del nivel más alto hacia la pila RPN, seguido por el contador.

25F1A PopMetaVStackDROP ( 🡺 obn..ob1 )

Pila Virtual:

( [obn..ob1] 🡺 )

Hace PopMetaVStack y luego borra el contador.

2624C GetMetaVStack ( 🡺 obn..ob1 #n )

Pila Virtual:

( [obn..ob1] 🡺 [obn..ob1] )

Copia los objetos de la pila virtual del nivel más alto hacia la pila RPN, seguidos por el contador.

La Pila Virtual permanece sin cambios.

25F17 GetMetaVStackDROP ( 🡺 obn..ob1 )

Pila Virtual:

( [obn..ob1] 🡺 [obn..ob1] )

Hace GetMetaVStack y luego borra el contador.

25F20 PushVStack&Keep ( obn..ob1 obm'..ob1' #m 🡺 obm'..ob1' #m )

Pila Virtual:

( 🡺 [obn..ob1] )

Traslada todos los objetos de pila RPN que estén por encima del meta hacia una nueva pila virtual, removiendo estos elementos de la pila RPN, pero dejando intacto al meta.

25F21 PushVStack&KeepDROP ( obn..ob1 obm'..ob1' #m 🡺 obm'..ob1' )

Pila Virtual:

( 🡺 [obn..ob1] )

Hace PushVStack&Keep y luego DROP.

25F1C PopVStackAbove ( obm'..ob1' 🡺 obn..ob1 obm'..ob1' )

Pila Virtual:

( [obn..ob1] 🡺 )

Traslada el contenido de la pila virtual más reciente (como el comando PopMetaVStackDROP) hacia la pila RPN, pero sobre el actual contenido de la pila RPN.

Esto deshace la acción del comando PushVStack&Keep (o la del comando PushVStack&KeepDROP).

**Direcc. Nombre Descripción**

26215 DropVStack ( 🡺 )

Pila Virtual:

( [obn..ob1] 🡺 )

Borra la pila virtual del nivel más alto de la Pila Virtual.

265D5 (SetMetaVStack) ( obn'..ob1' #n 🡺 )

Pila Virtual:

( [obn..ob1] 🡺 [obn'..ob1'] )

Reemplaza los objetos del meta en la pila virtual.

En la pila virtual deben haber “n” objetos.

24.1.2 Comandos que Operan con Toda la Pila RPN

**Direcc. Nombre Descripción**

25F1E PushVStack ( obn...ob1 🡺 obn...ob1 )

Pila Virtual:

( 🡺 [obn...ob1] )

Copia toda la pila RPN a la Pila Virtual.

La pila RPN permanence sin cambios.

25F1F PushVStack&Clear ( obn..ob1 🡺 )

Pila Virtual:

( 🡺 [obn..ob1] )

Hace PushVStack y luego borra toda la pila RPN.

25F18 GetVStack ( obm..ob1 🡺 obn'..ob1' )

Pila Virtual:

( [obn'..ob1'] 🡺 [obn'..ob1'] )

Copia la pila virtual del nivel más alto a la pila RPN.

La Pila Virtual no cambia, pero la anterior pila RPN desaparece.

25F1B PopVStack ( obm...ob1 🡺 obn'...ob1' )

Pila Virtual:

( [obn'...ob1'] 🡺 )

Traslada la pila virtual del nivel más alto a la pila RPN.

La anterior pila RPN desaparece.

24.1.3 Comandos que Operan con un Elemento de la Pila Virtual

**Direcc. Nombre Descripción**

26229 GetElemTopVStack ( #i 🡺 obi )

Pila Virtual:

( [obn..ob1] 🡺 [obn..ob1] )

Retorna el objeto iésimo de la pila virtual más alta, contando desde arriba.

“Contando desde arriba” significa que el objeto para el cual i=#0 es aquel del nivel n, i=#1 se refiere al objeto del nivel n-1 y asi sucesivamente.

Nota: este comando no verifica si #i es válido.

2626F PutElemTopVStack ( nuevo\_ob #i 🡺 )

Pila Virtual:

( [obn..ob(n-i)..ob1] 🡺 [obn..nuevo\_ob..ob1] )

Reemplaza el objeto iésimo de la pila virtual más alta con el objeto nuevo\_ob, contando desde arriba.

Nota: este comando no verifica si #i es válido.

**Direcc. Nombre Descripción**

26224 GetElemBotVStack ( #i 🡺 obi )

Pila Virtual:

( [obn..ob1] 🡺 [obn..ob1] )

Retorna el objeto iésimo de la pila virtual más alta, contando desde abajo.

“Contando desde abajo” significa que el objeto para el cual i=#0 es aquel del nivel 1, i=#1 se refiere al objeto del nivel 2 y asi sucesivamente.

Nota: este comando no verifica si #i es válido.

2626A PutElemBotVStack ( nuevo\_ob #i 🡺 )

Pila Virtual:

( [obn..obi..ob1] 🡺 [obn..nuevo\_ob..ob1] )

Reemplaza el objeto iésimo de la pila virtual más alta con el objeto nuevo\_ob, contando desde abajo.

Nota: este comando no verifica si #i es válido.

24.1.4 Protection Word

**Direcc. Nombre Descripción**

26233 GetVStackProtectWord ( 🡺 # )

Retorna el “protection word” de la pila virtual más alta.

2622E SetVStackProtectWord ( # 🡺 )

Usarlo adecuadamente:

Fija un nuevo “protection word” para la pila virtual más alta.

24.3 Ejemplos

Ejemplo 1 Pila Virtual

**Intercambiar dos niveles de pila virtual.**

A veces es deseable intercambiar dos niveles de pila virtual. Por ejemplo, para conseguir un elemento de una pila virtual que no está en el nivel más alto o para reemplazar algún elemento de esa pila virtual que no es la más reciente.

\* Intercambia los dos pilas virtuales de

\* los niveles más altos.

\* Conservan su "protection word".

NULLNAME SWAP\_VSTACK ( -> )

:: ( )

GetVStackProtectWord ( #pw1 )

PopMetaVStack ( #pw1 meta1 )

#1+ ( meta1' )

GetVStackProtectWord ( meta1' #pw2 )

PopMetaVStack ( meta1' #pw2 meta2 )

#1+ ( meta1' meta2' )

psh ( meta2' meta1' )

#1- ( meta2' #pw1 meta1 )

PushMetaVStack&Drop ( meta2' #pw1 )

SetVStackProtectWord ( meta2' )

#1- ( #pw2 meta2 )

PushMetaVStack&Drop ( #pw2 )

SetVStackProtectWord ( )

;

Capítulo 25  
Operaciones con la Memoria

Las operaciones básicas para guardar y llamar son los comandos:

CREATE, STO y @:

**Comando Pila y Descripción**

CREATE ( ob id 🡺 )

Crea una variable con el nombre id y contenido ob. Siempre es creada en el directorio actual.

Un error ocurre si ob es o contiene al directorio actual (“Directorio recursivo”).

Este comando no verifica si ya existe una variable con el mismo nombre en el directorio actual. Si es así, es creada otra variable con el mismo nombre.

STO ( ob id 🡺 )

( ob lam 🡺 )

En el caso del LAM, STO lo busca en el entorno temporal más recientemente creado, y sube si es necesario. Si lo encuentra, entonces guarda ob en el lam. Un error ocurre si el lam no está definido.

En el caso del id, STO lo busca en el directorio actual, y sube a los directorios padre si es necesario. Si lo encuentra, entonces guarda ob en el id. Si no lo encuentra, entonces crea la variable en el directorio actual y le asigna ob.

@ ( id 🡺 ob TRUE )

( id 🡺 FALSE )

( lam 🡺 ob TRUE )

( lam 🡺 FALSE )

Intenta retornar el contenido de la variable (nombre global) o del identificador temporal (nombre local).

Retorna el contenido guardado en el nombre y TRUE si existe, o sólo FALSE si la variable no existe o no está definido un lam con ese nombre.

En el caso de los id, empieza buscando en el directorio actual, y si es necesario sube hacia los directorios padre.

En el caso de los LAMS, empieza buscando en el entorno temporal creado más recientemente, y sube si es necesario.

Un problema con STO y @ es que si tu le das, digamos xSIN como el argumento, el cuerpo entero de la función es guardada en la variable. Por esta razón, es mejor usar SAFESTO y SAFE@, los cuales funcionan como STO y @, pero además estos convierten los contenidos ROM en nombres XLIB (SAFESTO) y los devuelven (SAFE@).

Notar que la palabra “SAFE” en estas y otras entradas solo significa que estas entradas hacen las conversiones descritas arriba. No hacen alguna otra verificación o seguridad adicional.

Hay muchos otros comandos relacionados a memoria, los cuales encontrarás en la lista de abajo.

25.1 Referencia

25.1.1 Llamando

**Direcc. Nombre Descripción**

0797B @ ( id/lam 🡺 ob T )

( id/lam 🡺 F )

Comando básico para llamar.

35C2C DUP@ ( id/lam 🡺 id/lam ob T )

( id/lam 🡺 id/lam F )

Hace DUP luego @.

2F064 Sys@ ( ID 🡺 ob T )

( ID 🡺 F )

Cambia temporalmente al directorio HOME y ejecuta @ ahí.

35A5B SAFE@ ( id/lam 🡺 ob T )

( id/lam 🡺 F )

Para lams hace @.

Para ids también hace ?ROMPTR> al objeto encontrado.

35A56 DUPSAFE@ ( id/lam 🡺 id/lam ob T )

( id/lam 🡺 id/lam F )

Hace DUP luego SAFE@.

25EF7 SAFE@\_HERE ( id 🡺 ob F )

( id 🡺 T )

( lam 🡺 ob F )

( lam 🡺 T )

Como SAFE@, pero funciona sólo en el directorio actual.

Equivale a :: DoHere: SAFE@ ;

00C002 FLASHPTR 002 00C ( id/{}/tag 🡺 ob T )

( id/{}/tag 🡺 id/{}/tag F )

Parecido al comando @ pero con este podrás llamar un objeto ubicado en cualquier parte.

Ejemplos:

Para llamar en el directorio actual o arriba de este:

ID NOMBRE FLASHPTR 002 00C

Para llamar sólo en el directorio actual:

{ ID NOMBRE } FLASHPTR 002 00C

Para llamar sólo en subdirectorios del directorio actual:

{ ID DIR1 ID DIR2 ID NOMBRE } FLASHPTR 002 00C

Para llamar sólo en el directorio HOME:

{ xHOME ID NOMBRE } FLASHPTR 002 00C

Para llamar desde cualquier lugar del directorio HOME:

{ xHOME ID DIR1 ID DIR2 ID NOMBRE } FLASHPTR 002 00C

Para llamar un objeto situado en un puerto (0, 1 ó 2):

TAG 2 ID NOMBRE FLASHPTR 002 00C

TAG 2 { ID NOMBRE } FLASHPTR 002 00C

TAG 2 { ID DIR1 ID DIR2 ID NOMBRE } FLASHPTR 002 00C

Para llamar un objeto situado en la tarjeta SD:

TAG 3 ID NOMBRE FLASHPTR 002 00C

TAG 3 { ID NOMBRE } FLASHPTR 002 00C

TAG 3 "NOMBRE" FLASHPTR 002 00C

TAG 3 { "NOMBRE" } FLASHPTR 002 00C

TAG 3 "CARP1/CARP2/NOMBRE" FLASHPTR 002 00C

TAG 3 { "CARP1/CARP2/NOMBRE" } FLASHPTR 002 00C

**Direcc. Nombre Descripción**

2F2A3 XEQRCL ( id/lam 🡺 ob )

Como SAFE@, pero si el id o lam no es encontrado, entonces genera el error “Nombre no definido”.

Equivale al comando **RCL** de User RPL cuando en la pila hay un id o un lam.

2F2A3 (DUPXEQRCL) ( id 🡺 id ob )

( lam 🡺 lam ob )

2F24E LISTRCL ( id/{}/tag 🡺 ob )

Como FLASHPTR 002 00C, pero si la variable no es encontrada, entonces genera el error “Nombre no definido”.

Equivale al comando **RCL** de User RPL cuando en la pila hay una lista.

2F2C6 (XEQXRCL) ( tag 🡺 ob )

Llama a FLASHPTR 002 00C.

Si no es encontrada la variable, quita una etiqueta y llama nuevamente a FLASHPTR 002 00C.

Si no es encontrada la variable, quita otra etiqueta y llama nuevamente a FLASHPTR 002 00C.

Si después de quitadas todas las etiquetas, la variable no es encontrada, genera el error “Nombre no definido”.

Equivale al comando **RCL** de User RPL cuando en la pila hay un objeto etiquetado.

25.1.2 Guardando

**Direcc. Nombre Descripción**

07D27 STO ( ob id/lam 🡺 )

En el caso del id, STO lo busca en el directorio actual, y sube a los directorios padre si es necesario. Si lo encuentra, entonces guarda ob en el id. Si no lo encuentra, entonces crea la variable en el directorio actual y le asigna ob.

Si reemplaza algún objeto, ese objeto es copiado a TEMPOB y los punteros son actualizados.

En el caso del LAM, STO lo busca en el entorno temporal más recientemente creado, y sube si es necesario. Si lo encuentra, entonces guarda ob en el lam. Un error ocurre si el lam no está definido.

2F380 SysSTO ( ob ID 🡺 )

Cambia temporalmente al directorio HOME y ejecuta STO ahí.

35A29 SAFESTO ( ob id/lam 🡺 )

Para lams sólo hace STO.

Para ids también hace ?>ROMPTR a ob antes de guardarlo con STO.

25E79 XEQSTOID ( ob id/lam 🡺 )

Para lams, sólo hace STO.

Para ids también hace SAFESTO, pero funciona sólo en el directorio actual y no sobreescribe un directorio.

aka: ?STO\_HERE

**Direcc. Nombre Descripción**

3E823 xSTO> ( ob id 🡺 ob )

( ob symb 🡺 ob )

Como xSTO, pero si el argumento del nivel 1 es un simbólico, usa el primer elemento de este como la variable para escribir.

Es el comando **►** de User RPL.

011002 FLASHPTR 002 011 ( library/bak %n 🡺 )

( library/bak tag 🡺 )

( ob tag 🡺 ob T )

Guarda un objeto en un puerto o en la tarjeta SD.

Si se va a guardar una biblioteca o un backup en el puerto 0,1 ó 2, en el nivel 1 de la pila puede estar sólo el número del puerto como real o entero (si estuviera un etiquetado, su contenido es ignorado).

Para guardar una biblioteca o un backup en la tarjeta SD si se debe especificar un nombre dentro del etiquetado.

Ejemplos:

Para guardar un objeto en un puerto (0, 1 ó 2):

Objeto TAG 2 ID NOMBRE FLASHPTR 002 011

Para guardar un objeto en la tarjeta SD:

Objeto TAG 3 ID NOMBRE FLASHPTR 002 011

Objeto TAG 3 { ID NOMBRE } FLASHPTR 002 011

Objeto TAG 3 "NOMBRE" FLASHPTR 002 011

Objeto TAG 3 { "NOMBRE" } FLASHPTR 002 011

Objeto TAG 3 "CARP1/CARP2/NOMBRE" FLASHPTR 002 011

Objeto TAG 3 { "CARP1/CARP2/NOMBRE" } FLASHPTR 002 011

0BD007 ˆPROMPTSTO1 ( id/lam 🡺 )

Suspende las operaciones de pila y pide al usuario la entrada de uno o más datos. Lo ingresado por el usuario es guardado en la variable especificada.

El entorno de entrada de datos es similar al que se obtiene con el comando **INPUT** de User RPL.

Al presionar CANCL una vez, la línea de edición es limpiada. Si ahora se presiona CANCL de nuevo, la entrada de datos es interrumpida generando un mensaje de error.

Al confirmar la línea de edición con ENTER, se guarda en la variable el objeto ingresado con ?STO\_HERE.

Si se ingresa más de un objeto, en la variable será guardado un objeto programa que contenga a los objetos ingresados (sin los delimitadores « y »).

Si se ingresa un único objeto algebraico, este es evaluado y el resultado es guardado en la variable.

Si se ingresa algo incorrecto, se genera el error: "Sintaxis incorrecta".

Equivale al comando **PROMPTSTO** de User RPL.

085D3 REPLACE ( newob oldob 🡺 newob )

El objeto del nivel 1 debe ser el contenido de alguna variable del directorio actual o de sus subdirectorios.

El objeto del nivel 2 pasará a ser el contenido de esa variable.

069004 ˆRENAME ( id ob 🡺 )

Renombra un nombre global.

id es el nuevo nombre.

ob es el contenido de un nombre global que ya existe.

**Direcc. Nombre Descripción**

08696 CREATE ( ob id 🡺 )

Crea una variable con el nombre id y contenido ob.

Siempre es creada en el directorio actual.

Un error ocurre si ob es o contiene al directorio actual (“Directorio recursivo”).

Este comando no verifica si ya existe una variable con el mismo nombre en el directorio actual. Si es así, es creada otra variable con el mismo nombre.

25.1.3 Borrando

**Direcc. Nombre Descripción**

08C27 PURGE ( id 🡺 )

Borra una variable.

Busca borrar en el directorio actual y si no encuentra, busca en los directorios padre hasta borrar una variable con ese nombre.

25E78 ?PURGE\_HERE ( id 🡺 )

Borra una variable sólo en el directorio actual.

Si id corresponde a un directorio, sólo será borrado si es un directorio vacío o con todas sus variables ocultas.

Equivale al comando **PURGE** de User RPL cuando en la pila hay un id.

012002 FLASHPTR 002 012 ( library/bak %n 🡺 )

( library/bak tag 🡺 )

( ob tag 🡺 ob T )

Guarda un objeto en un puerto o en la tarjeta SD.

Si se va a guardar una biblioteca o un backup en el puerto 0,1 ó 2, en el nivel 1 de la pila puede estar sólo el número del puerto como real o entero (si estuviera un etiquetado, su contenido es ignorado).

Para guardar una biblioteca o un backup en la tarjeta SD si se debe especificar un nombre dentro del etiquetado.

Ejemplos:

Para borrar un objeto de un puerto (0, 1 ó 2):

TAG 2 ID NOMBRE FLASHPTR 002 012

Para borrar un objeto en la tarjeta SD:

TAG 3 ID NOMBRE FLASHPTR 002 012

TAG 3 { ID NOMBRE } FLASHPTR 002 012

TAG 3 "NOMBRE" FLASHPTR 002 012

TAG 3 { "NOMBRE" } FLASHPTR 002 012

TAG 3 "CARP1/CARP2/NOMBRE" FLASHPTR 002 012

TAG 3 { "CARP1/CARP2/NOMBRE" } FLASHPTR 002 012

1D3006 ˆSAFEPURGE ( id 🡺 )

Si la variable existe y:

- Si el flag 120 está activado (silent mode on), la variable es borrada inmediatamente con PURGE.

- Si el flag 120 está desactivado (silent mode off), se hace la pregunta “Purge current variable?” al usuario. Si el usuario escoge YES, la variable es borrada con PURGE.

Si se intenta borrar un lam que ya existe, se genera el error “Purge current variable ”.

Si se intenta borrar un lam que no existe, no hace nada.

**Direcc. Nombre Descripción**

43F001 FLASHPTR 001 43F ( {id/tag/xPICT} 🡺 )

Ejecuta ?PURGE\_HERE a cada id de la lista.

Ejecuta FLASHPTR 002 012 a cada etiquetado de la lista.

Ejecuta XEQPURGEPICT a la palabra xPICT.

25.1.4 Directorios

**Direcc. Nombre Descripción**

25EA1 CREATEDIR ( id 🡺 )

Comando que crea un directorio vacío dentro del actual.

Si id no contiene objeto alguno, crea un directorio vacío.

Si id contiene un directorio vacío o un objeto que no es directorio , entonces lo borra con ?PURGE\_HERE y luego crea el directorio vacío.

Si id ya contiene un directorio no vacío, entonces manda el mensaje de error “Directorio no vacío”.

Equivale al comando **CRDIR** de User RPL

25EA1 (CREATERRP) ( id 🡺 )

Comando que crea que un directorio vacío dentro del actual.

Este comando no verifica si ya existe una variable con el mismo nombre en el directorio actual. Si es así, se crea el directorio vacío con el mismo nombre de todas maneras.

08326 LASTRAM-WORD ( rrp 🡺 ob T )

( rrp 🡺 F )

Llama al primer objeto del directorio.

Si el directorio está vacío retorna FALSE.

25EE7 LastNonNull ( rrp 🡺 ob T )

( rrp 🡺 F )

Llama al primer objeto del directorio, si no tiene nombre nulo.

Si el primer objeto del directorio tiene nombre nulo (nullid) o el directorio está vacío retorna sólo FALSE.

08376 PREVRAM-WORD ( ob 🡺 ob' T )

( ob 🡺 F )

ob debe encuentrarse en algún lugar del directorio HOME o uno de sus subdirectorios.

Retorna el siguiente objeto del directorio en el que se encuentra ob y TRUE.

Si ob es el último objeto en su directorio, retorna sólo FALSE.

25EF2 PrevNonNull ( ob 🡺 ob' T )

( ob 🡺 F )

ob debe encuentrarse en algún lugar del directorio HOME o uno de sus subdirectorios.

Retorna el siguiente objeto (si no tiene nombre nulo) del directorio en el que se encuentra y TRUE.

Si el siguiente objeto del directorio tiene nombre nulo, entonces retorna sólo FALSE.

Si ob es el último objeto en su directorio, retorna sólo FALSE.

**Direcc. Nombre Descripción**

082E3 RAM-WORDNAME ( ob 🡺 id )

Llama al nombre del objeto de la pila, si este está guardado en la memoria en HOME o en el puerto 0.

Si tiene nombre nulo o no está en lla memoria, retorna NULLID

Si ob es el directorio HOME, también pone NULLID en la pila.

25F14 XEQPGDIR ( id 🡺 )

Borra un directorio del directorio actual.

Si en el directorio actual, id contiene a un objeto de otro tipo, entonces manda el mensaje de error: “Argumento: valor incorr”.

Equivale al comando **PGDIR** de User RPL.

25EC4 DoHere: ( 🡺 )

El siguiente objeto del runstream es evaluado solamente para el directorio actual.

2F296 XEQORDER ( { id1 id2.. } 🡺 )

( id 🡺 )

Ordena las variables en el directorio actual moviendo las variables dadas al comienzo del directorio.

No admite un id nulo como argumento.

Equivale al comando **ORDER** de User RPL.

25EB9 DOVARS ( 🡺 {id1 id2..} )

Retorna una lista con todas las variables del directorio actual.

Si en el directorio actual hay una variable nula, retorna una lista con los objetos que estén antes de dicha variable.

Equivale al comando **VARS** de User RPL.

25EB8 DOTVARS% ( % 🡺 {} )

Retorna una lista de variables en el directorio actual que contienen un objeto del tipo dado por el número real (y que además están antes de la variable nula).

Equivale al comando **TVARS** de User RPL cuando en la pila hay un número real.

2C3FA (DOTVARS) ( {# #' ...} 🡺 {} )

( TRUE 🡺 {} )

Retorna una lista de las variables en el directorio actual que contengan un objeto de cualquiera de los tipos dados por los bints de la lista (y que además están antes de la variable nula).

Si el argumento es TRUE, retorna todas las variables del directorio actual.

Este comando es el núcleo de los comandos **TVARS** y **VARS** de User RPL.

25EF1 PATHDIR ( 🡺 { xHOME ID DIR1 ID DIR2 … } )

Retorna la ruta actual.

Equivale al comando **PATH** de User RPL.

2F265 UPDIR ( 🡺 )

Va al directorio padre.

Equivale al comando **UPDIR** de User RPL.

**Direcc. Nombre Descripción**

08309 (MYRAMROMPAIR) ( rrp 🡺 rrp' T )

( rrp 🡺 F )

Si rrp es HOME, pone FALSE.

De lo contrario, pone el directorio padre de rrp y TRUE.

08D92 HOMEDIR ( 🡺 )

Establece a HOME como el directorio actual.

Equivale al comando **HOME** de User RPL.

aka: SYSCONTEXT

08DC4 (SYSSTOPSIGN) ( 🡺 )

Establece a HOME como el directorio STOPSIGN.

08DD4 SYSRRP? ( rrp 🡺 flag )

Retorna TRUE si rrp es el directorio HOME.

08D5A CONTEXT@ ( 🡺 rrp )

Llama al directorio actual.

08D08 CONTEXT! ( rrp 🡺 )

Fija el directorio como el nuevo directorio actual.

08D82 (STOPSIGN@) ( 🡺 rrp )

Llama al directorio STOPSIGN.

08D4A (STOPSIGN!) ( rrp 🡺 )

Fija el directorio como el nuevo directorio STOPSIGN.

3712C SaveVarRes ( 🡺 )

Crea dos variables locales sin nombre que contienen al directorio actual y al directorio STOPSIGN.

Equivale a

:: CONTEXT@ STOPSIGN@\_ 2NULLLAM{}\_ BIND ;

37186 RestVarRes ( 🡺 )

Primero, establece HOME como el directorio actual y el directorio STOPSIGN (por si un error ocurre luego).

Luego, restaura el directorio actual y el directorio STOPSIGN desde LAM2 y LAM1 respectivamente.

Equivale a

:: HOMEDIR SYSSTOPSIGN\_ 1GETLAM 2GETLAM ABND

CONTEXT! STOPSIGN\_ ;

¿Qué es el directorio STOPSIGN?

El directorio STOPSIGN puede ser el directorio HOME o alguno de sus subdirectorios. Sirve para saber hasta donde buscaremos una variable (cuando subimos en los directorios) al usar algunos comandos de este capítulo.

Por defecto, el directorio STOPSIGN es el directorio HOME. De esta manera, si usamos por ejemplo, el comando @ para llamar al contenido de un nombre global (id), entonces la búsqueda empezará en el directorio actual, y de no encontrarse el id, la búsqueda continúa hacia arriba hasta llegar al directorio HOME si es necesario.

Ahora, si tenemos por ejemplo el arbol de directorios de la figura. Si el directorio actual es DIR4 y establecemos a DIR2 como el directorio STOPSIGN, entonces si colocamos en la pila un id y ejecutamos el comando @, entonces la búsqueda empieza en el directorio actual (DIR4) y de no encontrar ahí la variable entonces busca hacia arriba en DIR3 o en DIR2 si es necesario, pero de no encontrar la variable ya no busca más arriba de DIR2, pues este es el directorio STOPSIGN.

Si el directorio STOPSIGN no es HOME y nosotros estamos en el directorio STOPSIGN o en alguno de sus subdirectorios, entonces al escribir el nombre de un comando User RPL con las letras del teclado, el comando no será reconocido (Sólo podrán ser reconocidos los comandos de bibliotecas ligadas al directorio actual o a directorios padre de este hasta el directorio STOPSIGN). Por esto, si por alguna razón cambiamos el directorio STOPSIGN en un programa, antes de terminar el programa debemos devolverlo a su valor por defecto (HOME) usando el comando SYSSTOPSIGN\_

¿Si en la pila hay una lista, cómo puedo saber si esa lista es la ruta de algún directorio existente en HOME?

Puedes usar la siguiente subrutina.

En el nivel 1 de la pila debe haber una lista.

Si esa lista representa a una ruta de algún directorio existente en la forma

{ xHOME ID DIR1 … ID DIRn }, entonces retorna el directorio seguido de TRUE.

De lo contrario, sólo retorna FALSE.

NULLNAME {}\_EsRuta? ( {} -> rrp T // F )

::

DUPNULL{}? ( {} flag )

case DROPFALSE ( Sale con: FALSE )

( {} )

DUP CARCOMP ( {} ob1 )

' xHOME EQUALNOT ( {} flag )

case DROPFALSE ( Sale con: FALSE )

( {} )

DUPLENCOMP #1= ( {} flag )

casedrop :: CONTEXT@ HOMEDIR CONTEXT@ SWAP CONTEXT! TRUE ;

( {} )

FLASHPTR 002 00C ( ob T // {} F )

NOTcase DROPFALSE

( ob )

DUPTYPERRP? ( rrp T // ob F )

NOTcsdrpfls ( Sale con: FALSE )

( rrp )

TRUE ( rrp T )

;

Sin embargo, en dicho código no se toma en cuenta al directorio oculto, o a subdirectorios dentro del directorio oculto, o a otros directorios que tengan un nombre nulo. Para incluir a estos puedes cambiar ese código por este otro:

NULLNAME {}\_EsRuta? ( {} -> rrp T // F )

::

DUPNULL{}? ( {} flag )

case DROPFALSE ( Sale con: FALSE )

( {} )

DUP CARCOMP ( {} ob1 )

' xHOME EQUALNOT ( {} flag )

case DROPFALSE ( Sale con: FALSE )

( {} )

CDRCOMP ( {} )

DUPNULL{}? ( {} flag )

casedrop :: CONTEXT@ HOMEDIR CONTEXT@ SWAP CONTEXT! TRUE ;

( {} )

TRUE ( {} T )

CONTEXT@ ( {} T rrp\_inicial )

FLASHPTR 2LAMBIND ( {} )

HOMEDIR ( {} )

DUPLENCOMP ( {} #n )

SWAP ( #n {} )

>R ( #n )

ZERO\_DO (DO) ( )

RSWAP 'R RSWAP ( Obi )

:: DUPTYPEIDNT? ( Obi flag )

NOTcase :: DROPFALSE 2PUTLAM ExitAtLOOP ; ( OBS: Sale del LOOP )

( IDi )

SAFE@\_HERE ( OBi T // F )

NOTcase :: FALSE 2PUTLAM ExitAtLOOP ; ( OBS: Sale del LOOP )

( OBi )

DUPTYPERRP? ( RRPi T // OBi F )

NOTcase :: DROPFALSE 2PUTLAM ExitAtLOOP ; ( OBS: Sale del LOOP )

( RRPi )

CONTEXT! ( )

;

LOOP ( )

RDROP ( ) ( Borra los elem. del 1º nivel de la pila de retornos )

2GETLAM ( flag )

DUP ( flag flag )

IT :: CONTEXT@ SWAP ; ( rrp\_ruta T // F )

1GETABND ( rrp\_ruta T rrp\_inicial // F rrp\_inicial )

CONTEXT! ( rrp\_ruta T // F )

;

25.1.5 El Directorio Oculto

**Direcc. Nombre Descripción**

3714A SetHiddenRes ( 🡺 )

Establece al directorio oculto como el directorio actual y también como el directorio STOPSIGN.

370C3 WithHidden Run Stream:

( ob 🡺 )

Ejecuta el siguiente objeto del runstream en el directorio oculto.

370AF RclHiddenVar ( id 🡺 ob T )

( id 🡺 F )

Llama a la variable en el directorio oculto.

Equivale a :: WithHidden @ ;

37104 StoHiddenVar ( ob id 🡺 )

Guarda la variable en el directorio oculto.

Equivale a :: WithHidden STO ;

37118 PuHiddenVar ( id 🡺 )

Borra la variable del directorio oculto.

Equivale a :: WithHidden PURGE ;

De cualquiera de estas maneras puedes entrar al directorio oculto:

• HOMEDIR NULLID EVAL

• HOMEDIR EvalNULLID\_

• SetHiddenRes

Usando cualquiera de estas tres maneras, se ingresa al directorio oculto y además se establece este directorio oculto como el directorio STOPSIGN.

Para salir del directorio oculto correctamente, debemos devolver el directorio STOPSIGN a la normalidad, es decir, que este sea el directorio HOME. Por lo tanto, puedes salir del directorio oculto usando alguna de estas maneras:

• HOMEDIR SYSSTOPSIGN\_

• UPDIR SYSSTOPSIGN\_

• { xHOME ID DIR1 ID DIR2 ... ID DIRn } COMPEVAL SYSSTOPSIGN\_

• { HOMEDIR ID DIR1 ID DIR2 ... ID DIRn } COMPEVAL SYSSTOPSIGN\_

• CONTEXT! SYSSTOPSIGN\_ (cuando en la pila se encuentre un directorio)

¿Por qué al entrar al directorio oculto, no puedo usar los comandos User RPL al escribirlos con las teclas?

Cuando entramos al directorio oculto usando el comando SetHiddenRes, el directorio oculto también pasa a ser el directorio STOPSIGN. De esta manera, no se puede llamar al contenido de una variable que esté en el directorio HOME. Tampoco se pueden usar los comandos de User RPL escribiendolos con el teclado o abriendo el catálogo de comandos. Por ejemplo, escribe en la calculadora:

0 COS

y obtendrás en la pila:

0 ID COS

Cuando lo normal es obtener:

1.

Para evitar eso, puedes ejecutar el comando SYSSTOPSIGN\_ después de entrar al directorio oculto con SetHiddenRes para establecer al directorio HOME como el directorio STOPSIGN.

• HOMEDIR NULLID EVAL

• HOMEDIR EvalNULLID\_

• SetHiddenRes SYSSTOPSIGN\_

25.1.6 Memoria Temporal

Los objetos de la pila están en una zona llamada “memoria temporal”. Como su nombre lo dice, esta zona está destinada para el almacenamiento temporal. Cuando duplicas un objeto que está en la pila con DUP o presionando ENTER, en realidad no creas una copia de ese objeto, pues la pila contiene sólo punteros a objetos. Por lo tanto, sólo ese puntero es duplicado.

Cuando se modifica un objeto, la mayoría de los comandos automáticamente hace una nueva copia del objeto en cuestión y modifica la copia.

Por ejemplo, si ingresas una cadena en la pila, presionas ENTER y luego editas la cadena, tendrás dos cadenas diferentes ahora. Esto ha sucedido así, solamente porque una copia de la cadena fue hecha antes de editarla. Si la copia no hubiese sido hecha, las dos cadenas se habrían modificado simultáneamente, pues hubieran representado al mismo objeto en la memoria de la calculadora.

Existen pocos comandos que no hacen una copia del objeto antes de editarlo. Esto significa que todas las copias del objeto, que están en la pila, o incluso guardadas en la memoria serán modificadas al mismo tiempo. Esto es deseable a veces, pero a veces no. Estos comandos son llamados “tipo bang”. Cuando este tipo de comandos aparece en este libro, esto es notado en su descripcción.

Puedes usar los comandos TOTEMPOB o CKREF para hacer otra copia del objeto que está en el nivel uno de la pila. De esta manera, el objeto que está en el nivel uno de la pila no será el mismo que otro objeto que se encuentre en otro nivel de la pila, ni tampoco será una referencia a alguna variable global almacenada en la memoria de la calculadora. El comando TOTEMPOB siempre hace una copia del objeto, mientras que los comandos CKREF y FLASHPTR 001 3B2 sólo lo hacen si es necesario.

Por ejemplo, el comando INVGROB invierte los píxeles del grob que está en el nivel uno de la pila. El comando INVGROB es un comando “tipo bang”. Al ejecutar el programa:

:: "PALABRA" $>GROB ' ID A45 STO ID A45 INVGROB ;

Primero se guarda un grob en la variable 'ID A45'.

Luego ID A45 pone en la pila el contenido de la variable 'ID A45'. Esto es, en la pila se encuentra un grob (en realidad, un puntero que hace referencia al contenido de la variable global ‘A45’). El comando INVGROB invierte los píxeles de ese grob. Como INVGROB es un comando “tipo bang”, entonces no se hizo ninguna copia del objeto que estaba en la pila antes de editar ese objeto (invertir sus píxeles). El resultado es que tanto el grob de la pila como el contenido de la variable 'ID A45' han sido cambiados con INVGROB (pues en realidad, representaban al mismo objeto en la memoria de la calculadora).

Para evitar que también cambie el contenido guardado en la variable 'ID A45' puedes usar:

:: "PALABRA" $>GROB ' ID A45 STO ID A45 TOTEMPOB INVGROB ;

**Direcc. Nombre Descripción**

06657 TOTEMPOB ( ob 🡺 ob' )

Copia objeto a TEMPOB y retorna un puntero a la nueva copia.

35C90 TOTEMPSWAP ( ob1 ob2 🡺 ob2' ob1 )

Hace TOTEMPOB luego SWAP.

06B4E INTEMNOTREF? ( ob 🡺 ob flag )

Si el objeto está en TEMPOB, no está en un compuesto y no está referenciado, retorna el objeto y TRUE.

De lo contrario, retorna el objeto y FALSE.

25E9F CKREF ( ob 🡺 ob' )

Si el objeto está en TEMPOB, no está en un compuesto y no está referenciado, entonces no hace nada.

De lo contrario, copia a TEMPOB y retorna la copia.

3B2001 FLASHPTR 001 3B2 ( ob 🡺 ob' )

Llama a TOTEMPOB sólo si es necesario.

Hace: :: INTEMNOTREF? ?SEMI TOTEMPOB ;

3700A SWAPCKREF ( ob1 ob2 🡺 ob2 ob1' )

Hace SWAP luego CKREF.

01E0E8 ˜INTEMPOB? ( ob 🡺 ob flag )

06B3E (FREEINTEMP?) ( ob1 🡺 ob flag )

Prueba si el objeto está en TEMPOB y no está en un compuesto.

065D9 (PTRREFD?) ( ob 🡺 ob flag )

Pueba si el objeto está referenciado.

065E5 (REFERENCED?) ( ob 🡺 ob flag )

Prueba si el objeto está referenciado o en un compuesto.

06BC2 (NOTREF?) ( ob 🡺 ob flag )

Hace: :: REFERENCED?\_ NOT ;

06DDE (>TOPTEMP) ( ob 🡺 ob' )

Mueve el objeto a la parte superior de TEMPOB.

No hace recolección de basura.

064D6 (DOADJ1) ( ob1 ob2 🡺 ob1 ob' )

Mueve referencias desde ob2 hacia ob1 (ob1 en TEMPOB).

064E2 (DOADJ) ( ob1 ob2 🡺 ob1 ob' )

Mueve referencias desde ob2 hacia ob1 (ob1 en TEMPOB).

Referencias al cuerpo de ob2 son movidas también.

064BD (TOTEMPOBADJ) ( ob 🡺 ob ob' )

Hace una copia independiente, moviendo referencias a la nueva copia.

Hace: :: DUP TOTEMPOB SWAP DOADJ\_ ;

3B4001 FLASHPTR 001 3B4 ( ob 🡺 ob' )

Hace:

:: REFERENCED?\_ NOT?SEMI TOTEMPOBADJ\_ SWAPDROP ;

25.1.7 Puertos y Tarjeta SD

**Direcc. Nombre Descripción**

013002 FLASHPTR 002 013 ( #puerto 🡺 {tag} #mem )

Retorna una lista con las variables del puerto 0, 1 ó 2 y la memoria disponible en ese puerto.

Las bibliotecas son retornadas como enteros de cuatro cifras.

Usado por el comando **PVARS** de User RPL.

¿Cómo puedo saber si hay una tarjeta SD insertada en la calculadora?

\* Retorna TRUE si hay una tarjeta SD en la calculadora.

\* Retorna FALSE si NO hay una tarjeta SD.

NULLNAME HAY\_TAJETA\_SD? ( -> flag )

::

CODE

NIBHEX 8F41FF357080B468F65253

ENDCODE

;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **HP 49g** | **HP 48gII** | **HP 49g+** | **HP 50g** |
| **PUERTO 0** | SI | SI | SI | **SI** |
| **PUERTO 1** | SI | NO | SI | **SI** |
| **PUERTO 2** | SI | NO | SI | **SI** |
| **TARJETA SD** | NO | NO | SI | **SI** |
| **Tamaño Pantalla** | 131x64 | 131x64 | 131x80 | **131x80** |

Capítulo 26  
Tiempo y Alarmas

Este capítulo contiene una lista de entradas relacionadas al tiempo: hora, fecha y la lista interna de alarmas.

26.1 Referencia

**Direcc. Nombre Descripción**

26120 SLOW ( 🡺 )

Retraso de 15 milisegundos.

26125 VERYSLOW ( 🡺 )

Retraso de 300 milisegundos.

2F37E SORTASLOW ( 🡺 )

Retraso de 1.2 segundos (4 x VERYSLOW).

2612A VERYVERYSLOW ( 🡺 )

Retraso de 3 segundos.

2F2D4 dowait ( %segundos 🡺 )

Espera un número determinado de segundos.

3005E %>HMS ( % 🡺 %hms )

Convierte de formato decimal a formato H.MMSS

Equivale al comando **→HMS** de User RPL.

30912 %%H>HMS ( %% 🡺 %%hms )

Similar a %>HMS, pero para reales extendidos.

30077 %HMS> ( %hms 🡺 % )

Convierte de formato H.MMSS a formato decimal.

Equivale al comando **HMS→** de User RPL.

3008B %HMS+ ( %hms1 %hms2 🡺 %hms )

Suma dos horas en formato hms.

Equivale al comando **HMS+** de User RPL.

300B3 %HMS- ( %hms1 %hms2 🡺 %hms )

Resta dos horas en formato hms.

Equivale al comando **HMS-** de User RPL.

2EECF TOD ( 🡺 %hora )

Retorna la hora actual.

Equivale al comando **TIME** de User RPL.

2F388 VerifyTOD ( %hora 🡺 %hora )

Verifica que %hora sea una hora correcta.

Si no lo es, manda el mensaje de error: “Hora incorrecta”

2EED0 DATE ( 🡺 %date )

Retorna la fecha actual.

Equivale al comando **DATE** de User RPL.

2F03B (>DATE) ( %date 🡺 )

Establece la fecha indicada como la fecha actual.

Equivale al comando **→TIME** de User RPL.

2EED2 DATE+DAYS ( %fecha %días 🡺 %fecha' )

Suma a la fecha indicada el número de días.

Equivale al comando **DATE+** de User RPL.

**Direcc. Nombre Descripción**

2EED1 DDAYS ( %fecha1 %fecha2 🡺 %días )

Retorna el número de días entre dos fechas.

Equivale al comando **DDAYS** de User RPL.

2EED7 CLKTICKS ( 🡺 hxs )

Retorna la hora del sistema como el número de ticks en hxs (de 13 nibbles). Un tick = 1/8192 segundos.

aka: SysTime

Equivale al comando **TICKS** de User RPL.

2F153 CLKADJ\* ( %ticks 🡺 )

Ajusta la hora del sistema sumandole o restandole el número de ticks indicado. Suma si %ticks es positivo, de lo contrario, resta). Recuerda: un tick = 1/8192 segundos.

Equivale al comando **CLKADJ** de User RPL.

2EED3 TIMESTR ( %fecha %hora 🡺 "dia\_sem fecha hora" )

Retorna una cadena que representa el tiempo, usando el formato actual de hora y el formato actual de fecha.

Por ejemplo, para la fecha 24 de agosto de 2009 y la hora 00:00:45 am puede retornar alguna de estas cadenas:

"WED 08/24/09 12:00:45A"

"WED 24.08.09 00:00:45"

Equivale al comando **TSTR** de User RPL.

2F329 Date>d$ ( %fecha 🡺 $ )

Retorna la fecha indicada como un cadena, usando el formato actual de fecha.

2F381 TOD>t$ ( %hora 🡺 $ )

Retorna la hora indicada como una cadena, usando el formato actual de hora.

2F1AB Date>hxs13 ( %date 🡺 hxs )

Convierte una fecha a ticks en hxs (de 13 nibbles).

Es igual al número de ticks a la hora 0 de la fecha indicada.

26.1.1 Alarmas

La lista interna de las alarmas se almacena en el directorio oculto y tiene este formato:

{ { hxs1 acción1 } { hxs2 acción2 } ... }

La longitud de cada hxs es 24 nibbles. Los 13 nibbles menos significativos representan el tick para la fecha y la hora. Los siguientes 10 nibbles representan el intervalo de repetición, si hay alguno. El nibble más significativo representa el estado de la alarma (pendiente, reconocida, etc.).

**Direcc. Nombre Descripción**

2F178 ALARMS@ ( 🡺 {} )

Retorna una lista que contiene todas las alarmas en la forma:

{ { hxs1 acción1 } { hxs2 acción2 } ... }

2F003 (Ticks>Date) ( hxs 🡺 %fecha )

Retorna la fecha desde el hxs (de 24 nibbles) de una alarma.

2F002 (Ticks>TOD) ( hxs 🡺 %hora )

Retorna la hora desde el hxs (de 24 nibbles) de una alarma.

2F004 (Ticks>Rpt) ( hxs 🡺 %repet )

Retorna el intervalo de repetición (en ticks) desde el hxs (de 24 nibbles) de una alarma.

**Direcc. Nombre Descripción**

2F37F STOALM ( %fecha %hora acción %repet 🡺 % )

Guarda una alarma. %repet es el número de ticks entre cada repetición (%0 es sin repetición). Un segundo es 8192 ticks.

Una hora es 8192\*60\*60 = 29491200 ticks.

Retorna un número real que representa la posición de la alarma en la lista.

2F0AC PURGALARM% ( % 🡺 )

Borra la alarma cuyo índice en la lista es el indicado.

Equivale al comando **DELALARM** de User RPL.

2F314 RCLALARM% ( %n 🡺 {} )

Llama a la alarma cuyo índice se indica. La alarma devuelta tiene la forma { %fecha %hora acción %repet }

Equivale al comando **RCLALARM** de User RPL.

2F336 FindNext ( hxs 🡺 # )

El argumento es un hxs de 13 nibbles que representa una fecha y hora.

Retorna un bint que representa el número de orden de la primera alarma que debe aparecer después del tiempo especificado.

Si no se encuentra ninguna alarma que cumpla eso, retorna un bint que es igual al tamaño de la lista de alarmas más uno.

Es usado por FINDALARM{}

2F113 FINDALARM{} ( { %fecha %hora } 🡺 % )

Retorna un número real que representa el número de orden de la primera alarma que debe aparecer después del tiempo especificado.

Si no se encuentra ninguna alarma que cumpla eso, retorna 0.

25FA9 ALARM? ( 🡺 flag )

Retorna TRUE si una alarma está en camino.

Capítulo 27  
Funciones de Sistema

A continuación, una lista de comandos relacionadas con el sistema, como la configuración de algunos aspectos de la calculadora y el apagado de esta. Los comandos relacionados con las banderas de usuario y las banderas de sistema también se describen aquí.

27.1 Referencia

27.1.1 Banderas de Usuario y de Sistema

**Direcc. Nombre Descripción**

2614D SetSysFlag ( # 🡺 )

Activa la bandera de sistema con el número #.

26044 ClrSysFlag ( # 🡺 )

Desactiva la bandera de sistema con el número #.

26170 TestSysFlag ( # 🡺 flag )

Devuelve TRUE si la bandera de sistema está activada.

26152 SetUserFlag ( # 🡺 )

Activa la bandera de usuario con el número #.

26049 ClrUserFlag ( # 🡺 )

Desactiva la bandera de usuario con el número #.

26175 TestUserFlag ( # 🡺 flag )

Devuelve TRUE si la bandera de usuario está activada.

2F259 RCLSYSF ( 🡺 hxs )

Llama a las banderas de sistema desde 1 hasta 64.

2F25F (STOSYSF) ( hxs 🡺 )

Guarda las banderas de sistema desde 1 hasta 64.

2F23E DOSTOSYSF ( hxs 🡺 )

Guarda las banderas de sistema desde 1 hasta 64.

Luego, si el flag 55 está activado (No last args), borra los argumentos del último comando en LASTARG.

2F25A (RCLSYSF2) ( 🡺 hxs )

Llama a las banderas de sistema desde 65 hasta 128.

2F260 (STOSYSF2) ( hxs 🡺 )

Guarda las banderas de sistema desde 65 hasta 128.

2F25B RCLUSERF ( 🡺 hxs )

Llama a las banderas de usuario desde 1 hasta 64.

2F261 (STOUSERF) ( hxs 🡺 )

Guarda las banderas de usuario desde 1 hasta 64.

2F25C (RCLUSERF2) ( 🡺 hxs )

Llama a las banderas de usuario desde 65 hasta 128.

2F262 (STOUSERF2) ( hxs 🡺 )

Guarda las banderas de usuario desde 65 hasta 128.

2F3A9 (STOALLFcont) ( hxs\_sys hxs\_usr 🡺 )

Guarda banderas de sistema y banderas de usuario desde 1 hasta 64.

2F3AA (STOALLFcont2) ( hxs\_sys1 hxs\_usr1 hxs\_sys2 hxs\_usr2 🡺 )

Espera 4 hxs y los guarda como banderas de sistema banderas de usuario.

**Direcc. Nombre Descripción**

3B76C (DOSTOALLF) ( {} 🡺 )

Guarda banderas de sistema y banderas de usuario.

Espera una lista con dos o cuatro hxs.

Las primeras dos, son las banderas de sistema y de usuario respectivamente, desde 1 hasta 64.

Las últimas dos, si están presentes, son las banderas de sistema y de usuario, respectivamente, desde 65 hasta 128.

25F23 SaveSysFlags ( 🡺 )

Guarda las banderas de sistema (2 hxs) en la pila virtual.

Equivale a

RCLSYSF RCLSYSF2\_ BINT2 PushMetaVStack&Drop

25F22 RestoreSysFlags ( 🡺 )

Restaura las banderas de sistema (2 hxs) desde la pila virtual. Quita los dos hxs de la pila virtual.

Equivale a usar

PopMetaVStackDROP STOSYSF2\_ STOSYSF\_

2ABF0 RunSafeFlags Run Stream:

( ob 🡺 )

Evalua el siguiente objeto en el runstream, pero guarda y restaura las banderas de sistema alrededor de esta evaluación.

Si un error ocurre al evaluar, las banderas de sistema son restauradas a sus valores anteriores a la evaluación, luego se interrumpe el programa con un mensaje de error.

Usa DoRunSafe. Este comando es muy útil.

2AB69 RunInApprox Run Stream:

( ob 🡺 )

Evalúa el siguiente objeto en el runstream, con las banderas de sistema 20, 21, 100 activadas y 22, 105, 102, 120 activadas.

Guarda y restaura las banderas de sistema alrededor de esta evaluación.

Si un error ocurre al evaluar, las banderas de sistema son restauradas a sus valores anteriores a la evaluación, luego se interrumpe el programa con un mensaje de error.

2AC0E DoRunSafe ( ob 🡺 hxs1 hxs2 )

Evalua ob y pone en la pila las banderas del sistema como estaban antes de la evaluación.

Si un error ocurre al evaluar, las banderas de sistema son restauradas a sus valores anteriores a la evaluación, luego se interrumpe el programa con un mensaje de error.

Usado por RunSafeFlags y RunSafeFlagsNoError.

2ABD7 RunSafeFlagsNoError Run Stream:

( ob 🡺 )

Evalua el siguiente objeto en el runstream.

Guarda y restaura las banderas de sistema alrededor de esta evaluación, sólo si un error ocurre al evaluar; luego se interrumpe el programa con un mensaje de error.

Si no ocurre un error en la evaluación, entonces no restaura las banderas de sistema anteriores.

**Direcc. Nombre Descripción**

2EFA5 DOHEX ( 🡺 )

La manera de mostrar cadenas hexadecimales en la pila cambia a base hexadecimal.

2EFA8 DODEC ( 🡺 )

La manera de mostrar cadenas hexadecimales en la pila cambia a base decimal.

2EFA6 DOBIN ( 🡺 )

La manera de mostrar cadenas hexadecimales en la pila cambia a base binaria.

2EFA7 DOOCT ( 🡺 )

La manera de mostrar cadenas hexadecimales en la pila cambia a base octal.

2EFBF BASE ( 🡺 # )

Retorna un bint que representa la base en la que se muestran actualmente las cadenas hexadecimales.

Retorna BINT16 para base hexadecimal, BINT10 para base decimal, BINT8 para base octal o BINT2 para base binaria.

2605D DOSTD ( 🡺 )

Equivale al comando **STD** de User RPL.

26053 DOFIX ( # 🡺 )

Equivale al comando **FIX** de User RPL.

26058 DOSCI ( # 🡺 )

Equivale al comando **SCI** de User RPL.

2604E DOENG ( # 🡺 )

Equivale al comando **ENG** de User RPL.

261A7 savefmt1 ( 🡺 )

Guarda el actual formato de número de número (STD, FIX, SCI, ENG), y cambia a modo STD.

261A2 rstfmt1 ( 🡺 )

Restaura el formato de número guardado por savefmt1. Sólo un conjunto de banderas se puede usar, no hay anidación de estas entradas.

2FFDB SETRAD ( 🡺 )

Establece el modo angular a RAD (radianes).

Equivale a hacer BINT17 SetSysFlag

2FFBD SETDEG ( 🡺 )

Establece el modo angular a DEG (sexagesimales).

Equivale a hacer BINT18 ClrSysFlag BINT17 ClrSysFlag

2FFEF SETGRAD ( 🡺 )

Establece el modo angular a GRAD (centesimales).

Equivale a hacer BINT18 SetSysFlag BINT17 ClrSysFlag

25EF3 RAD? ( 🡺 flag )

Devuelve TRUE si el modo angular actual es RAD.

Equivale a hacer BINT17 TestSysFlag

25EBA DPRADIX? ( 🡺 flag )

Devuelve TRUE si el separador de decimales actual es "."

Equivale a hacer BINT51 TestSysFlag NOT

27.1.2 Funciones Generales

**Direcc. Nombre Descripción**

25EB2 DOBEEP ( %frecuencia %duración 🡺 )

Hace un Beep. Análogo al comando **BEEP** de User RPL.

Frecuencia en Hz, duración en segundos.

261AC setbeep ( #duración #frecuencia 🡺 )

También hace beep.

Frecuencia en Hz, duración en milisegundos.

05F42 GARBAGE ( 🡺 )

Fuerza a una recolección de basura.

05F61 MEM ( 🡺 #nib )

Retorna en la pila la cantidad de memoria libre en nibbles.

No hace recolección de basura (en cambio, el comando **MEM** de User RPL si lo hace).

05902 OSIZE ( ob 🡺 # )

Retorna el tamaño de ob en nibbles.

Fuerza a una recolección de basura.

05944 OCRC ( ob 🡺 #nib hxs )

Retorna el tamaño de ob en nibbles y su checksum como hxs.

2F257 OCRC% ( ob 🡺 hxs %bytes )

Retorna checksum en hxs y tamaño en bytes.

2F267 VARSIZE ( id 🡺 hxs %bytes )

Retorna el checksum en hxs y el tamaño en bytes del objeto guardado en id. Si el id no tiene algún valor, muestra el mensaje de error “Nombre no definido”

394C8 INHARDROM? ( ob 🡺 ob flag )

Retorna TRUE si ob es un puntero cuya dirección es menor que #80000h

Por ejemplo:

:: ' NEXTIRQ INHARDROM? ; retorna NEXTIRQ FALSE

:: ' VARSIZE INHARDROM? ; retorna VARSIZE TRUE

05AB3 CHANGETYPE ( ob #prólogo 🡺 ob' )

Cambia el prólogo del objeto, antes hace TOTEMPOB.

25F90 >LANGUAGE ( # 🡺 )

Cambia el idioma actual para mostrar los mensajes de error.

BINT0 para inglés, BINT1 para francés, BINT2 para español.

Versión interna del comando **→LANGUAGE** de User RPL.

25F95 LANGUAGE> ( 🡺 # )

Devuelve un bint que representa el idioma actual para mostrar los mensajes de error.

Versión interna del comando **LANGUAGE→** de User RPL.

27.1.3 UNDO y CMD

**Direcc. Nombre Descripción**

256A7 UNDO\_ON ( 🡺 )

256A6 UNDO\_OFF ( 🡺 )

256A2 UNDO\_ON? ( 🡺 flag )

25636 HISTON? ( 🡺 flag )

2EF6D GetLastEdit ( # 🡺 # $ T )

( # 🡺 # F )

27.1.4 Apagado, Alertas y Batería

**Direcc. Nombre Descripción**

041A7 TurnOff ( 🡺 )

Apaga la calculadora.

Equivale al comando **OFF** de User RPL.

041ED DEEPSLEEP ( 🡺 flag )

Apaga la calculadora. Al prenderla, no se muestra ninguna alerta, como por ejemplo “Batería baja”.

Retorna TRUE si se debe de inicializar los puertos de la calculadora (Invalid Card Data).

01118 LowBat? ( 🡺 flag )

Retorna TRUE si hay baterías bajas.

0426A ShowInvRomp ( 🡺 )

Muestra "Atención: Tarjeta: datos invál." con FlashWarning.

En inglés es: “Warning: Invalid Card Data”.

2EE5D ?FlashAlert ( 🡺 )

Muestra alerta del sistema (si la hay) con FlashWarning.

04544 (AlertStatus) ( 🡺 # )

04575 (Alert$) ( # 🡺 $ )

Capítulo 28  
Comunicación

Las entradas listadas aquí permiten al programador escribir programas que permitan la comunicación con otras máquinas.

28.1 Referencia

**Direcc. Nombre Descripción**

2EEBB SENDLIST ( {} 🡺 )

Equivale al comando **SEND** de User RPL cuando en la pila hay una lista.

2EEBC GETNAME ( $/id/lam 🡺 )

Equivale al comando **KGET** de User RPL cuando en la pila hay un nombre global o local o una cadena.

2EEBD DOFINISH ( 🡺 )

Equivale al comando **FINISH** de User RPL.

2EEBE DOPKT ( $ $' 🡺 )

Equivale al comando **PKT** de User RPL.

2EEC1 DOBAUD ( % 🡺 )

Especifica velocidad de transferencia de bits. Lo guarda como primer parámetro de **IOPAR**.

Llama a GetIOPAR y a StoIOPAR.

Equivale al comando **BAUD** de User RPL.

2EEC2 DOPARITY ( % 🡺 )

Establece el valor de paridad. Lo guarda como segundo parámetro de **IOPAR**.

Llama a GetIOPAR y a StoIOPAR.

Equivale al comando **PARITY** de User RPL.

2EEC3 DOTRANSIO ( % 🡺 )

Especifica una opción de traducción de caracteres en la transferencia de datos. Lo guarda como sexto parámetro de **IOPAR**.

Llama a GetIOPAR y a StoIOPAR.

Equivale al comando **TRANSIO** de User RPL.

2EEC4 DOKERRM ( 🡺 $ )

Equivale al comando **KERRM** de User RPL.

2EEC5 DOBUFLEN ( 🡺 % 0/1 )

Equivale al comando **BUFLEN** de User RPL.

2EEC6 DOSBRK ( 🡺 )

Equivale al comando **SBRK** de User RPL.

2EEC7 DOSRECV ( % 🡺 )

Equivale al comando **SRECV** de User RPL.

2EEC9 CLOSEUART ( 🡺 )

Equivale al comando **CLOSEIO** de User RPL.

2F130 (DOXMIT) ( $ 🡺 %1 )

( $ 🡺 $’ %0 )

Equivale al comando **XMIT** de User RPL.

**Direcc. Nombre Descripción**

2EECB DOCR ( 🡺 )

Equivale al comando **CR** de User RPL.

2EECC DOPRLCD ( 🡺 )

Imprime una imagen en pixeles por pixeles de la pantalla actual (excluyendo los anunciadores).

Equivale al comando **PRLCD** de User RPL.

2EECD DODELAY ( % 🡺 )

Especifica cuántos segundos espera la calculadora entre envíos de líneas de información a la impresora.

Equivale al comando **DELAY** de User RPL.

2F31A APNDCRLF ( $ 🡺 "$\0D\0A" )

Agrega retorno de carro y salto de línea a la cadena.

27A3A StdPRTPAR ( 🡺 {} )

PRTPAR por defecto: { 0. "" 80. "\0D\0A" }

281001 FLASHPTR 001 281 ( 🡺 %0 "" %80 "\0D\0A" )

Guarda en el directorio HOME una lista por defecto para la variable **PRTPAR** y pone sus elementos en la pila.

2F063 StoPRTPAR ( {} 🡺 )

Guarda la lista en el directorio HOME en la variable PRTPAR.

2716D StdIOPAR ( 🡺 {} )

**IOPAR** por defecto: { 9600. 0. 0. 0. 3. 1. }

0B1003 FLASHPTR 003 0B1 ( 🡺 {} )

**IOPAR** por defecto. En HP50G: { 115200. 0. 0. 0. 3. 1. }

280001 FLASHPTR 001 280 ( 🡺 %115200/%9600 %0 %0 %0 %3 %1 )

Guarda en el directorio HOME una lista por defecto para la variable **IOPAR** y pone sus elementos en la pila.

2EEBF GetIOPAR ( 🡺 %baud % % % % % )

Llama al **IOPAR** de HOME y pone sus elementos en la pila.

**IOPAR** debe ser una lista con seis elementos. De lo contrario, generará el error "IOPAR inválido".

Si **IOPAR** aun no existe, llama a FLASHPTR 001 280.

2F062 StoIOPAR ( {} 🡺 )

Guarda la lista de los parámetros IO en el directorio HOME en la variable **IOPAR**.

2F37B SetIOPARErr ( 🡺 )

Genera el error "IOPAR inválido".

2F34E KVIS ( $ 🡺 $' )

Traduce algunos caracteres especiales a digraphs para su posterior transferencia ASCII a una PC.

El comportamiento de este comando depende del sexto parámetro de **IOPAR** de la siguiente manera:

0 ó 1: ningún cambio es hecho.

2: Traduce caracteres con números del 128 al 159 (80-9F hex)

3: Traduce caracteres con números del 128 al 255.

Por ejemplo, si el sexto término de **IOPAR** es el real 3 y en la pila se encuentra la cadena:

"Σ(Ai) = n \* ñ \* ß"

Entonces el resultado será:

"\GS(Ai) = n \* \241 \* \Gb"

**Direcc. Nombre Descripción**

2F34F KVISLF ( $ 🡺 $' )

Como KVIS, pero además cambia el salto de línea de la HP (carácter 10d o 0Ah) a salto de línea de la PC, poniendo el carácter 13d (0Dh) delante de el.

2F34D KINVISLF ( $ 🡺 $' $'' )

Traduce digraphs en la cadena a caracteres de la HP.

También remueve los caracteres 13d (0Dh) que se encuentren al final de cada línea.

El comportamiento de este comando depende del sexto parámetro de **IOPAR**.

2F389 VERSTRING ( 🡺 $ )

Retorna una cadena con la versión. Por ejemplo “HPHP49-C”

Capítulo 29  
EL FILER

El FILER de la calculadora HP permite al programador escribir varias aplicaciones que manejan archivos.

Varias aplicaciones ya incorporadas en la ROM usan el filer:

* El administrador de archivos (tecla FILES).
* El visor de fuentes (tecla MODE 🡺 DISP 🡺 CHOOSE 🡺 Browse).
* Numeric Solver (tecla NUM.SLV 🡺 1.Solve equation.. 🡺 CHOOSE).
* Diff Equation Solver (tecla NUM.SLV 🡺 2.Solve diff eq.. 🡺 CHOOSE).
* Linear System Solver (tecla NUM.SLV 🡺 4.Solve lin sys.. 🡺 CHOOSE).
* Stat 1 var (tecla STAT 🡺 1.Single-var.. 🡺 CHOOSE)
* Etc.

29.1 Usando el Filer

La entrada general para llamar al filer es ˆFILER\_MANAGERTYPE. Este comando toma tres argumentos:

* Filer\_Tipos
* Filer\_RutaInicial
* Filer\_Comportamiento.

Este comando puede devolver en la pila lo siguiente:

**Devolución Manera en que se salió**

FALSE Al presionar ON cuando se explora algún directorio o puerto.

:0: { } Al presionar CANCL, ON o CHDIR cuando estás en el arbol de directorios.

……… TRUE Cuando presionas una tecla asignada a un programa personalizado que devuelve TakeOver como último elemento en la pila (sección 29.1.3.4)

29.1.1 El Argumento Filer\_Tipos

Este argumento permite seleccionar los tipos de objetos que se mostrarán en el filer. Esta es una lista que contiene los prólogos de los objetos permitidos. Los prólogos son bints diferentes para cada tipo de objeto. En la siguiente página se muestran los prólogos de los objetos de la calculadora HP.

Por ejemplo, si quieres que se muestren números reales, números complejos y nombres globales la lista que debes escribir es { # 2933 # 2977 # 2E48 }

Si quieres que se muestren objetos de todos los tipos en el filer, entonces este argumento deberá ser una lista que contenga al bint cero, es decir: { BINT0 }. Como hacer esto es muy común, existe un comando que abrirá el filer mostrando todos los objetos. Este comando es ˆFILER\_MANAGER. Usando este comando, solo debes ingresar los otros dos argumentos.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Código**  **Dispatch** | **Tipo**  **User** | **Tipo de objeto** | **Filer** | **Prólogo** | **Abrev. Prólogo** |
| BINT1=# 1 | 0. | Números reales | REAL | #2933 | TYPEREAL |
| BINT2=# 2 | 1. | Números complejos | CPLX | #2977 | TYPECMP\_ |
| BINT3=# 3 | 2. | Cadena | STRNG | #2A2C |  |
| BINT4=# 4 | 3. | Arreglos reales | ARRAY | #29E8 |  |
|  | 4. | Arreglos no reales | ARRAY | #29E8 |  |
|  | 29. | Matrices simbólicas | MATRX | #2686 | TYPEMATRIX\_ |
| BINT5=# 5 | 5. | Lista | LIST | #2A74 | TYPELIST\_ |
| BINT6=# 6 | 6. | Nombre global | GNAME | #2E48 | TYPEIDNT |
| BINT7=# 7 | 7. | Nomble Local | LNAME | #2E6D | TYPELAM\_ |
| BINT8=# 8 | 8. | Programa | PROG | #2D9D | TYPECOL\_ |
|  | 18. | Pointer comando de User RPL permitido en algebraicos | XLIB | #2D9D | TYPECOL\_ |
|  | 19. | Pointer comando de User RPL no permitido en algebraicos | XLIB | #2D9D | TYPECOL\_ |
| BINT9=# 9 | 9. | Simbólico | ALG | #2AB8 | TYPESYMB\_ |
| BINT10=# A |  | Clase simbólica: id, lam, symb, enteros. |  |  |  |
| BINT11=# B | 10. | Cadenas hexadecimales (hxs) | BIN | #2A4E |  |
| BINT12=# C | 11. | Gráfico (grob) | GROB | #2B1E |  |
| BINT13=# D | 12. | Etiquetado (tagged) | TAG | #2AFC |  |
| BINT14=# E | 13. | Unidades | UNIT | #2ADA | TYPEEXT\_ |
| BINT15=# F | 14. | Rompointer | XLIB | #2E92 |  |
| BINT31=#1F | 20. | Entero binario (bint) | SB | #2911 |  |
| BINT47=#2F | 15. | Directorio | DIR | #2A96 | TYPERRP\_ |
| #3F | 21. | Número real extendido | LREAL | #2955 | TYPEEREL\_ |
| #4F | 22. | Número complejo extendido | LCPLX | #299D |  |
| #5F | 23. | Arreglo vinculado | LARRY | #2A0A |  |
| #6F | 24. | Carácter | CHAR | #29BF |  |
| #7F | 25. | Code | CODE | #2DCC |  |
| #8F | 16. | Biblioteca | L•••• | #2B40 |  |
| #9F | 17. | BACKUP | BCKUP | #2B62 |  |
| #AF | 26. | Library Data | LBDAT | #2B88 |  |
| #BF | 27. | Access pointer (Extended Ptr) | EXPTR | #2BAA |  |
| #CF | 30. | Fuente normal | FONT | #2BCC |  |
| #DF | 27. | Minifuente | MFONT | #26FE |  |
| #EF | 27. | External object 4 (Extended 3) | EXT1 | #2C10 |  |
| BINT255d=#FF | 28. | Entero | INTG | #2614 | TYPEINT\_ |
| BINT0=# 0 | 27. | Flashpointer | FLSHP | #26AC |  |
| BINT0=# 0 | 27. | Pointer | EXTER |  |  |
| BINT0=# 0 | 27. | Aplet | EXTER | #26D5 |  |
| BINT0=# 0 | 27. | External object 3 (Extended 2) | AUNIT | #2BEE |  |

29.1.2 El Argumento Filer\_RutaInicial

Este argumento es una lista y especifica la ruta en la que empezará el Filer. Esta lista puede ser:

**Valor Significado Ejemplos**

{ } Empieza en HOME NULL{}

{ dir } Empieza en un subdirectorio de HOME { ID CASDIR }

{ ID DIR1 ID DIR2 }

:n:{ } Empieza en el puerto n TAG 1 { }

:n:{ dir } Empieza en directorio o backup en puerto n TAG 0 { ID DIR1 ID DIR2 }

TAG 1 { ID DIR1 ID DIR2 }

TAG 2 { ID DIR1 ID DIR2 }

TAG 3 { ID CARP1 ID CARP2 }

Si deseas que la ruta en la que empieza el filer sea el directorio actual, entonces sólo escribe: PATHDIR CDRCOMP.

29.1.3 El Argumento Filer\_Comportamiento

Este argumento es una lista y especifica las asignaciones que daremos a las teclas del menú y a los demás botones de la calculadora mientras dura la exploración con el filer. Cada asignación está representada por una lista que contiene entre tres y cinco elementos. La estructura general del argumento Filer\_Comportamiento es la siguiente:

{

\* Asignación 1

{ "Nombre"

Localización

Acción

[ Programa\_Extra (es necesario si 16 <= Acción <= 23) ]

[ Tecla\_Atajo ]

}

\* Asignación 2

{ "Nombre"

Localización

Acción

[ Programa\_Extra (es necesario si 16 <= Acción <= 23) ]

[ Tecla\_Atajo ]

}

\* Posiblemente más asignaciones

}

El argumento Filer\_Comportamiento también puede ser una lista etiquetada de la forma: TAG n { { … } { … } … } donde n es un número: 1, 2, etc y representa al número de filas del menú. De esta manera, al presionar la tecla PREV cuando estamos en la primera fila, pasamos a estar en la última fila. Además las listas que estén más allá del número de filas indicado con la etiqueta, sólo serán asignaciones de teclas que no sean del menú.

Cada uno de los elementos de las sublistas sera descrito a continuación:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Función ya definida**  **sin tecla atajo** | **Función ya definida**  **con tecla atajo** | **Función personalizada**  **sin tecla atajo** | **Función personalizada**  **con tecla atajo** |
| **Nombre** | Cadena  Bint  Grob 21x8  Programa (con TakeOver al inicio) que retorne cadena, bint o grob 21x8 | | | |
| **Localización** | BINT0  BINT1  BINT2  BINT3  BINT4 | | | |
| **Acción** | BINT0 a BINT15  BINT24 a BINT46 | BINT0 a BINT15  BINT24 a BINT46 | BINT16 a BINT23 | BINT16 a BINT23 |
| **Programa extra** |  | TakeOver | Programa | Programa |
| **Tecla atajo** |  | Bint |  | Bint |

**29.1.3.1 Nombre**

Especifica lo que será mostrado en el menu. Puede ser una cadena, un grob o un bint. Si es un bint, entonces se mostrará el mensaje de error (ver Apéndice E) correspondiente a ese bint. También puede ser un programa que devuelva un objeto de alguno de los tres tipos mencionados. Además, este programa siempre debe tener al comando TakeOver como su primer elemento.

Si el argumento **Nombre** es NULL$, entonces sólo podrá ser asignacion de una tecla que no es del menú.

**29.1.3.2 Localización**

Indica las circunstancias en las cuales la acción correspondiente a este menu puede ser ejecutada. Puede ser un bint o un programa que retorne un bint al ser evaluado. Hay cinco valores posibles, los cuales están listados en la tabla de abajo.

**Valor Constante Significado**

BINT0 fDondeSea La acción puede ejecutarse en cualquier lugar.

BINT1 fHome La acción puede ejecutarse solo si el usuario está explorando

dentro del directorio HOME o uno de sus subdirectorios.

BINT2 fNoLib La acción NO puede ejecutarse si el usuario está explorando los

comandos que forman parte de una biblioteca.

BINT3 fNoBackup La acción NO puede ejecutarse si el usuario está explorando

dentro de un directorio u objeto de respaldo en un puerto.

BINT4 fPuerto La acción puede ejecutarse solo en la raíz de un puerto, pero no

dentro de un directorio de este u objeto de respaldo.

**29.1.3.3 Acción**

Este define lo que sucederá cuando el usuario presione la tecla correspondiente al menú o a la tecla de atajo asignada (ver la sección 29.1.3.5). Este es un bint o un programa que retorna un bint al ser evaluado. Es posible llamar a una función ya definida del filer, o definir tu propia acción. La tabla de abajo muestra las acciones ya definidas del filer.

|  |  |  |  |
| --- | --- | --- | --- |
| **Valor** | **Constante** | **M. Error** | **Acción** |
| BINT0 | "BEEP" |  | Hace un sonido beep. |
| BINT3 |  | # DF25 | Para ver el objeto seleccionado. |
| BINT4 |  | # DF1D | Muestra el árbol de directorios. |
| BINT5 | "UP" |  | Mueve la selección hacia arriba. |
| BINT6 | "MaxUP" |  | Mueve la selección hacia arriba, hasta el 1º elemento. |
| BINT7 | "DOWN" |  | Mueve la selección hacia arriba. |
| BINT8 | "MaxDOWN" |  | Mueve la selección hacia abajo, hasta el último elemento. |
| BINT9 | "CHECK" | # DF2E | Marca o desmarca la variable seleccionada. |
| BINT10 | "UPDIR" |  | Para ir al directorio padre. |
| BINT11 | "DOWNDIR" |  | Para entrar en el directorio seleccionado. |
| BINT12 | "PREV" |  | Muestra la anterior fila del menú. |
| BINT13 | "NEXT" |  | Muestra la siguiente fila del menú. |
| BINT14 |  | # DF1C | Evalua el objeto seleccionado. |
| BINT15 |  | # DF27 | Alterna entre los dos tipos de cabecera disponibles. |
| BINT24 |  | # DF28 | Alterna entre mostrar información sobre las variables o sólo sus nombres. |
| BINT25 |  | # DF18 | Edita la variable seleccionada. |
| BINT26 |  | # DF19 | Copia las variables marcadas.  Si ninguna está marcada, copia la variable seleccionada. |
| BINT27 |  | # DF1A | Mueve las variables marcadas. Si ninguna está marcada, mueve la variable seleccionada. |
| BINT28 |  | # DF1B | Pone en la pila el contenido de las variables marcadas. Si ninguna está marcada, pone a la variable seleccionada. |
| BINT29 |  | # DF1E | Borra las variables marcadas.  Si ninguna está marcada, borra la variable seleccionada. |
| BINT30 |  | # DF1F | Renombra a la variable seleccionada. |
| BINT31 | "CRDIR" |  | Crea un directorio. |
| BINT32 |  | # DF21 | Reordena las variables con marcas en el directorio. |
| BINT33 |  | # DF22 | Envía la variable seleccionada usando Kermit. |
| BINT34 |  | # DF24 | Suspende el filer temporalmente. |
| BINT35 |  | # DF26 | Edita la variable seleccionada en el editor más apropiado. |
| BINT36 |  | # DF23 | Recibe una variable usando Kermit. |
| BINT37 | "CANCL" | # DF2C | Sale del filer dejando FALSE en la pila. |
| BINT38 | "PageUP" |  | Mueve el filer una pantalla hacia arriba. |
| BINT39 | "PageDOWN" |  | Mueve el filer una pantalla hacia abajo. |
| BINT40 |  | # DF20 | Crea una nueva variable. |
| BINT41 |  | # DF29 | Abre un cuadro de diálogo con varias opciones para ordenar las variables. |
| BINT42 | "INICIO" |  | Mueve la selección al elemento inicialmente seleccionado en el directorio. |
| BINT43 | "Puerto0" |  | Para ir a explorar en el puerto 0. |
| BINT44 | "Puerto1" |  | Para ir a explorar en el puerto 1. |
| BINT45 | "Puerto2" |  | Para ir a explorar en el puerto 2. |
| BINT46 | "Puerto3" |  | Para ir a explorar en el puerto 3 (tarjeta SD). |

Para ejecutar un programa personalizado, el valor del argumento Acción debe estar en el rango 16–23. Cada uno de los siete valores especifica cuales elementos estarán en la pila, es decir, cuales son los argumentos de tu programa.

En la siguiente tabla se muestran los métodos de llamado. En esta tabla la ruta devuelta es la actualmente explorada, en el mismo formato que en los ejemplos de la sección 29.1.2, es decir, es una lista o una lista etiquetada.

**Valor Descripción**

BINT16 Llama sólo a la ruta actualmente explorada.

1: Ruta

BINT17 Llama también al nombre y al contenido del objeto actualmente seleccionado.

3: Ruta

2: Objeto

1: Nombre

BINT18 Llama al nombre y al contenido de todos los objetos marcados.

Si ningún objeto está marcado, entonces llama al nombre y al contenido del objeto

actualmente seleccionado como si este fuera el único marcado.

2n + 2: Ruta

...

5: Objeto 2

4: Nombre 2

3: Objeto 1

2: Nombre 1

1: Número de objetos (bint)

BINT19 Este programa es llamado varias veces, una vez por cada objeto marcado.

Para cada objeto, llama a los mismos tres que obtenemos al usar el BINT17.

Si no hay ningún objeto marcado, es llamado una vez para el objeto seleccionado.

BINT20 Llama al nombre del objeto actualmente seleccionado.

2: Ruta

1: Nombre

BINT21 Llama a los nombres de todos los objetos marcados.

Si ningún objeto está marcado, entonces llama al nombre del objeto actualmente

seleccionado como si este fuera el único marcado.

n + 2: Ruta

...

3: Nombre 2

2: Nombre 1

1: Número de objetos (bint)

BINT22 Llama al objeto seleccionado solo en una cadena de direcciones.

2: Ruta

1: Cadena

BINT23 Llama a los objetos marcados en un cadena de direcciones.

2: Ruta

1: Cadena

La cadena devuelta cuando el argumento **Acción** es BINT22 o BINT23 no sera descrita aquí, pues tiene poco uso y es más dificil de usar.

Cuando un programa es llamado sobre una **biblioteca que está en un puerto**, algunas reglas especiales se aplican al nombre devuelto:

• Para llamadas 17 y 18, el nombre sera el título de la biblioteca como un id.

Por ejemplo: ‘Nosy 4.1+ by JNE Bos’.

• Para la llamada 19, el nombre sera una “L” seguida del número de la bilioteca, como un id.

Por ejemplo: ‘L1625’.

• Para las llamadas 20 y 21, el nombre será el número de la bilioteca como un número real.

**29.1.3.4 Programa\_Extra**

Cuando usas una función ya definida del filer y no le vas a asignar ninguna tecla de atajo, entonces los argumentos Programa\_Extra y Tecla\_Atajo no son necesarios.

Cuando usas una función ya definida del filer y le vas a asignar una tecla de atajo, entonces el argumento Programa\_Extra debe ser sólo TakeOver y el argumento Tecla\_Atajo debe ser un bint (explicado en la siguiente página).

Pero si vas a usar una función personalizada, entonces el argumento Programa\_Extra debe ser un programa que será llamado al presionar la tecla correspondiente.

Existen algunas características adicionales que pueden ser útiles:

• Si estás explorando el directorio HOME o uno de sus subdirectorios, entonces tu programa empezará a ejecutarse en ese directorio que está siendo explorado.

• Si estás explorando dentro de un puerto, entonces tu programa empezará a ejecutarse en el directorio en el que estabas antes de entrar al filer.

• Un programa personalizado sólo se podrá ejecutar si hay un objeto marcado o seleccionado, excepto para la llamada número 16, la cual se puede ejecutar en un directorio vacío.

• Por defecto, luego de la ejecución del programa, el directorio explorado será analizado nuevamente, la pantalla se actualizará y si habían objetos marcados estos quedarán desmarcados. Para evitar esto, tu puedes dejar FALSE en la pila (excepto para la llamada número 19). Por ejemplo:

{ "INFO" ( Nombre )

BINT0 ( Localización: BINT0=DondeSea )

BINT20 ( Acción: BINT20= Prog Pers que pone Ruta y Nombre )

:: ( Ruta Nombre )

SWAPDROP ( Nombre )

"Nombre seleccionado es:\0A" ( Nombre $ )

SWAP DO>STR &$ ( $ )

FlashWarning ( )

FALSE ( FALSE )

;

}

• Si deseas salir del filer después de la ejecución del programa, debes dejar “TakeOver” en la pila (al salir de esta manera el filer retornará TRUE en la pila). Por ejemplo:

{ "OK" ( Nombre )

BINT0 ( Localización: BINT0=DondeSea )

BINT16 ( Acción: BINT16= Prog Pers que pone sólo la Ruta )

:: ( Ruta )

DROP ' TakeOver ( TakeOver )

;

}

El cual es parecido a la función ya definida BINT37 (pero que retorna FALSE en la pila):

{ "CANCL" ( Nombre )

BINT0 ( Localización: BINT0=DondeSea )

BINT37 ( Acción: BINT37= Sale del filer dejando FALSE en la pila )

}

**29.1.3.5 Tecla\_Atajo**

Usar este argumento para asignar una función ya definida o programa personalizado a una tecla. Este argumento es un bint en la forma # axx, donde a es 0 o 1, lo cual significa **sin ALPHA** y **con ALPHA** (sin soltar) respectivamente. xx es el código de la tecla, al cual podemos sumar de manera opcional #40 para el cambio izquierdo o #C0 para el cambio derecho.

|  |  |
| --- | --- |
| Por ejemplo, si deseas asignar a tu programa las teclas cambio izquierdo + TOOL, entonces el número debe ser #049 el cual es la suma #40 + #9 (cambio izquierdo y tecla TOOL). |  |
| Otro ejemplo: si deseas asignar a tu programa las teclas cambio derecho + SPC, entonces el número debe ser #0F2 el cual es la suma #C0 + #32 (cambio derecho y tecla SPC). |  |
| Otro ejemplo: si deseas asignar a tu programa las teclas ALPHA + EEX, entonces el número debe ser #11B el cual es la suma de los números #100 + #1B (ALPHA y tecla EEX). En este caso, es necesario mantener presionada ALPHA. |  |
| Otro ejemplo: si deseas asignar a tu programa las teclas ALPHA + cambio izquierdo + TOOL, entonces el número debe ser #149 el cual es la suma de los números #100 + #40 + #9. No es necesario mantener las teclas presionadas. |  |
| Otro ejemplo: si deseas asignar a tu programa las teclas ALPHA + cambio derecho + TOOL, entonces el número debe ser #1F2 el cual es la suma de los números #100 + #C0 + #32. No es necesario mantener las teclas presionadas. |  |

**NOTA**: Este argumento debe ser el quinto de la lista. De manera que si estás usando una función ya definida y deseas ponerle un argumento Tecla\_Atajo, tendrás que ingresar algo como esto:

{ "EDIT" ( Nombre )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver el objeto seleccionado )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 11B ( Tecla\_Atajo: #100= alpha + #1B= tecla EEX )

}



Esto le asignará a la función ya definida la tecla ALPHA-EEX. Recuerda que debes mantener presionado ALPHA y luego, sin soltar, presionar EEX.

29.1.4 Atajos de tecla predefinidos

Existen atajos de tecla predefinidos en el filer y los mostramos abajo.

Recuerda que cuando haces un programa que llame al filer, estos atajos de tecla predefinidos siempre estarán presentes a menos que les asignes otra acción a estas teclas.

Si quieres que no se exploren ciertos directorios o quieres que sólo se explore el directorio actual, cambiá las acciones de las teclas de atajo predefinidas que cambian el directorio actual.

Si no quieres que exista la marcación de objetos, cambiá las acciones de las teclas de atajo predefinidas que marcan o desmarcan a un objeto.

**29.1.4.2 Atajos de tecla predefinidos para cambiar de directorio explorado**

|  |  |  |  |
| --- | --- | --- | --- |
| Teclas | Tecla\_Atajo |  | Acción |
|  | # 10 | "DOWNDIR" | Para entrar en el directorio seleccionado. |
|  | # 0E | "UPDIR" | Para ir al directorio padre. |
|  | # 4B | "UPDIR" | Para ir al directorio padre. |
|  | # CE |  | Muestra el arbol de directorios. |
|  | # 110 | "DOWNDIR" | Para entrar en el directorio seleccionado. |
|  | # 10E | "UPDIR" | Para ir al directorio padre. |

**29.1.4.3 Atajos de tecla predefinidos para marcar o desmarcar un objeto seleccionado**

|  |  |  |  |
| --- | --- | --- | --- |
| Teclas | Tecla\_Atajo |  | Acción |
|  | # 1C | "CHECK" | Marca o desmarca el objeto seleccionado. |
|  | # 33 | "CHECK" | Marca o desmarca el objeto seleccionado. |
|  | #D5 | "CLEAR" | Desmarca todos los objetos que están marcados. |
|  | # 133 | "CHECK" | Marca o desmarca el objeto seleccionado. |

**29.1.4.1 Atajos de tecla predefinidos para ordenar los objetos del directorio explorado**

|  |  |  |  |
| --- | --- | --- | --- |
| Teclas | Tecla\_Atajo |  | Acción |
|  | # 12 | Clasifica Nombre | Ordena en orden alfabético. |
|  | # 52 | Clasifica Nombre Inverso | Ordena en orden alfabético inverso. |
|  | # D2 | Clasifica Nombre | Ordena en orden alfabético. |
|  | # 18 | Clasifica Tamaño | Ordena de acuerdo al tamaño, de menor a mayor. |
|  | # 58 | Clasifica Tamaño Inverso | Ordena de acuerdo al tamaño, pero de mayor a menor. |
|  | # D8 | Clasifica Tamaño | Ordena de acuerdo al tamaño, de menor a mayor. |
|  | # 19 | Clasifica Tipo | Ordena de acuerdo al tipo de objeto. |
|  | # 59 | Clasifica Tipo Inverso | Ordena de acuerdo al tipo de objeto, pero inverso al anterior. |
|  | # D9 | Clasifica Tipo | Ordena de acuerdo al tipo de objeto. |

**29.1.4.4 Atajos de tecla predefinidos para desplazarse entre las variables del directorio explorado**

|  |  |  |  |
| --- | --- | --- | --- |
| Teclas | Tecla\_Atajo |  | Acción |
|  | # A | "UP" | Mueve la selección hacia arriba. |
|  | # 4A | "PageUP" | Mueve la selección hacia arriba, una página. |
|  | # CA | "MaxUP" | Mueve la selección hacia arriba, hasta el primer elemento. |
|  | # 10A | "UP" | Mueve la selección hacia arriba. |
|  | # F | "DOWN" | Mueve la selección hacia abajo. |
|  | # 4F | "PageDOWN" | Mueve la selección hacia abajo, una página. |
|  | # CF | "MaxDOWN" | Mueve la selección hacia abajo, hasta el último elemento. |
|  | # 10F | "DOWN" | Mueve la selección hacia abajo. |

**29.1.4.5 Otros atajos de tecla predefinidos**

|  |  |  |  |
| --- | --- | --- | --- |
| Teclas | Tecla\_Atajo |  | Acción |
|  | # D | "NEXT" | Muestra la siguiente fila del menú. |
|  | # 4D | "PREV" | Muestra la anterior fila del menú. |
|  | # 2F | "CANCL" | Sale del filer, dejando FALSE en la pila. |
|  | # EF | "OFF" | Apaga la calculadora. |
|  | # 20 | Search OFF | Desactiva el modo de búsqueda, si está activado. De lo contrario, espera a otras teclas. |
|  | # 120 | Search ON | Activa el modo de búsqueda. |

29.2 El Comando FLASHPTR BrowseMem.1

Con este comando podemos explorar el filer y mandar a la pila el contenido o el nombre de todos los objetos marcados (o sólo del objeto seleccionado, si no hay ningún objeto marcado) al presionar OK o ENTER. Los argumentos para este comando son cinco:

• Ob1

• Ob2

• Flag\_Tecla\_Check

• Tipos\_Permitidos

• Flag\_Objeto\_Nombre

a) Los argumentos ob1 y ob2 pueden ser cualquier objeto.

b) El argumento Flag\_Tecla\_Check puede tomar dos valores:

- TRUE para que haya una tecla de menú “CHECK”

- FALSE para que no haya la tecla de menú “CHECK”

La tecla de menú CHECK marca o desmarca al objeto seleccionado.

|  |  |
| --- | --- |
| **Con tecla de menú CHECK** | **Sin tecla de menú CHECK** |
|  |  |

c) El parámetro TiposPermitidos es una lista de bints, que representan los tipos permitidos de los objetos que se verán en el filer. Sólo se aceptan los bints mostrados en la siguiente tabla. Para que se vean los objetos de todos los tipos existentes en la calculadora HP, el parámetro TiposPermitidos debe ser una lista vacía o también MINUSONE.

|  |  |
| --- | --- |
| **BINT** | **TIPOS DE OBJETOS** |
| BINT0 | Reales |
| BINT1 | Complejos |
| BINT2 | Cadenas |
| BINT3 | Arreglos reales y matrices simbólicas |
| BINT4 | Arreglos no reales |
| BINT5 | Listas |
| BINT6 | Nombres globales |
| BINT7 | Nombres locales |
| BINT8 | Programas |
| BINT9 | Simbólicos |
| BINT10 | Cadenas hexadecimales |
| BINT11 | Gráfico (grob) |
| BINT13 | Unidades |
| BINT16 | Biblioteca (Library) |
| BINT17 | BACKUP |
| BINT25 | Code |
| BINT26 | Library Data |
| BINT30 | Fuente normal |
| BINT255d=#FF | Enteros |

d) El argumento flag\_Objeto\_Nombre puede tomar dos valores TRUE o FALSE:

- TRUE para que se devuelvan los contenidos de los objetos. Al presionar OK termina el filer dejando en la pila:

|  |  |
| --- | --- |
| Estado de la pila | Contexto |
| {ob1...obn} |  |
| TRUE | Si hay más de una variable marcada. |
| ob1 … obn son los contenidos de cada variable marcada. |  |
| ob |  |
| TRUE | Si hay una variable marcada o ninguna. |
| ob es el contendido de la única variable marcada (si ninguna está marcada, es el contenido de la variable seleccionada). |  |

- FALSE para que se devuelvan los nombres de los objetos. Al presionar OK termina el filer dejando en la pila:

|  |  |
| --- | --- |
| **Estado de la pila** | **Contexto** |
| {nombre1...nombreN}  TRUE | Si estamos explorando en HOME o uno de sus subdirectorios y hay más de una variable marcada. |
| Nombre  TRUE | Si estamos explorando en HOME o uno de sus subdirectorios y hay una variable marcada o ninguna (si ninguna está marcada, es el nombre de la variable seleccionada). |
| {:n:{ruta nomb1} … :n:{ruta nombN}}  TRUE | Si estamos explorando en algún puerto y había más de una variable marcada.  ‘n’ es el número del puerto. |
| :n:{ ruta nombre}  TRUE | Si estamos explorando en algún puerto y hay una variable marcada o ninguna (si ninguna está marcada, es el nombre de la variable seleccionada). |

Al salir del filer, el comando FLASHPTR BrowseMem.1 puede devolver en la pila lo siguiente:

**Pila Manera en que se salió**

FALSE Al presionar ON o “CANCEL” cuando se explora algún directorio o puerto.

:0: { } Al presionar ON o “CANCEL” cuando estás en el arbol de directorios.

ob TRUE Al presionar ENTER u “OK” cuando se explora algún directorio o puerto.

El objeto retornado antes de TRUE está explicado en los dos cuadros de arriba.

29.3 Referencia

**Direcc. Nombre Descripción**

06E004 ˆFILER\_MANAGERTYPE ( {Tipos} {RutaInicial} {Args} 🡺 F )

( {Tipos} {RutaInicial} {Args} 🡺 ……… T )

( {Tipos} {RutaInicial} {Args} 🡺 :0:{} )

{args} = { ítem1 ítem2 ... }

ítem = {name loc action [prog] [key]}

Filer personalizado, se verán objetos de los tipos señalados.

06D004 ˆFILER\_MANAGER ( {RutaInicial} {Args} 🡺 F )

( {RutaInicial} {Args} 🡺 ……… T )

( {RutaInicial} {Args} 🡺 :0:{} )

Filer personalizado, se verán objetos de cualquier tipo.

067004 ˆFiler ( 🡺 )

Llama al administrador de archivos estándar.

070004 ˆBrowseMem.1 ( ob ob’ flag Tipos flag' 🡺 ob'' T )

( ob ob’ flag Tipos flag' 🡺 F )

( ob ob’ flag Tipos flag' 🡺 :0:{} )

04C004 ˆIfGetPrlgFromTypes ( {#tipos} 🡺 {#prólogos} )

( NULL{} 🡺 { #0 } )

( #FFFFF 🡺 { #0 } )

Genera una lista de los prólogos de los tipos permitidos.

Es usado por ˆBrowseMem.1 para convertir su lista de tipos.

068004 ^Arbo ( rrp $titulo flag 🡺 rrp' T )

( rrp $titulo flag 🡺 pointer T )

( rrp $titulo flag 🡺 F )

Abre un entorno en el cual puedes seleccionar un directorio desde el árbol de directorios.

Si sales con ENTER, OK o flecha derecha, en la pila se retornará el directorio seleccionado (o el número de puerto) y TRUE.

Si sales con CANCL u ON, en la pila se retorna FALSE.

Si el flag es TRUE, estará disponible la tecla de menú CHDIR (si la presionas, entonces saldrás de la pantalla y el nuevo directorio actual será el seleccionado y en la pila se retornará FALSE).

Por ejemplo, el siguiente programa podría generar una pantalla como la indicada.

:: ( )

CONTEXT@ ( rrp )

"Escoge una ruta" ( rrp $ )

TRUE ( rrp $ T )

FLASHPTR Arbo ( ob T // F )

;



29.4 Ejemplos

Ejemplo 1 Filer

**Poner una tecla de menú que no haga nada.**

Para hacer esto la lista correspondiente a esa tecla de menú debe ser

{ NULL$ BINT0 BINT0 }

También funciona poner

{ NULL$ }

También funciona poner

{ }

Pero cuidado, no debes colocar

NULL{}

El siguiente código llama al filer con tres teclas de menú, la segunda de estas no hace nada y no se ve en la pantalla.

ASSEMBLE

CON(1) 8 ( Le dice al parseador que el comando es 'No algebraico' )

RPL

xNAME FilerMKeyEmpty ( -> F // :0:{} )

:: CK0

{ BINT0 } ( Filer\_Tipos: { BINT0 } = Todos los objetos )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* HAY 3 TECLAS DE MENU

\* La 1º es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* La 2º es una tecla que sólo hace BEEP. No se ve en pantalla.

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

}

\* La 3º es para renombrar al objeto seleccionado

{ # DF1F ( Nombre: # DF1F "RENOM" )

BINT1 ( Localización: BINT1= En HOME o uno de sus subdirectorios )

BINT30 ( Acción: BINT30= Renombra el objeto )

}

}

FLASHPTR FILER\_MANAGERTYPE

;

|  |
| --- |
|  |
|  |
|  |

Ejemplo 2 Filer

**Asignar un programa a una tecla que no sea una tecla de menú.**

Para hacer esto, la lista correspondiente a esa tecla de menú debe tener como primer elemento a una cadena vacía. Además esta lista debe estar después de las listas correspondientes a las teclas de menú.

Recuerda que si una lista tiene como **Nombre** a una cadena vacía, entonces no se verá en el menú en la pantalla.

El siguiente código llama al filer con una tecla de menú y una asignación de tecla que no se ve en el menú.

ASSEMBLE

CON(1) 8 ( Le dice al parseador que el comando es 'No algebraico' )

RPL

xNAME FilerKeyNoMenu ( -> F // :0:{} )

:: CK0

{ BINT0 } ( Filer\_Tipos: { BINT0 } = Todos los objetos )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* La 1º es una tecla de menú para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* La 2º es una tecla que no corresponde al menú.

\* Abre un cuadro de diálogo para ordenar las variables como "CLASI"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0=DondeSea )

BINT41 ( Acción: BINT41= Abre un cuadro de diálogo para ordenar variables )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT7 ( Tecla\_Atajo: BINT7 = #7 = tecla APPS )

}

}

FLASHPTR FILER\_MANAGERTYPE

;

Ejemplo 3 Filer

**Abrir el Filer para explorar sólo el directorio inicial o puerto inicial sin poder ir a otros directorios o puertos.**

Para hacer esto escoger la Ruta\_Inicial que es el directorio o puerto inicial que queremos. Ahora también será el único que podemos explorar.

En el código de abajo se escogió el directorio HOME como Ruta\_Inicial por medio de una lista vacía, pero también puedes poner cualquier otro directorio o puerto.

A las 6 teclas predefinidas que cambian de directorio explorado, se les asignó otra acción, la de hacer BEEP.

ASSEMBLE

CON(1) 8 ( Le dice al parseador que el comando es 'No algebraico' )

RPL

xNAME FilerSoloHome ( -> F )

:: CK0

{ BINT0 } ( Filer\_Tipos: { BINT0 } = Todos los objetos )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* SOLO 1 TECLA DEL MENU

\* Es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* LAS RESTANTES LISTAS SOLO SON ASIGNACIONES DE TECLAS QUE NO SON DEL MENU

\* Esta lista es para quitar la tecla izquierda "UPDIR"

{ NULL$ ( Nombre: NULL$ para no asignar a ninguna tecla del MENU )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT14 ( Tecla\_Atajo: BINT14 = #E = Tecla izquierda "UPDIR" )

}

\* Esta lista es para quitar la tecla derecha "DOWNDIR"

{ NULL$ ( Nombre: NULL$ para no asignar a ninguna tecla del MENU )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT16 ( Tecla\_Atajo: BINT16 = #10 = Tecla derecha "DOWNDIR" )

}

\* Esta lista es para quitar la tecla cambio izquierdo + VAR "UPDIR"

{ NULL$ ( Nombre: NULL$ para no asignar a ninguna tecla del MENU )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT75 ( Tecla\_Atajo: BINT75 = #4B = #40 + #B = cambio izq. + VAR "UPDIR" )

}

\* Esta lista es para quitar la tecla cambio derecho + izquierda "ARBOL"

{ NULL$ ( Nombre: NULL$ para no asignar a ninguna tecla del MENU )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# CE ( Tecla\_Atajo: #CE = #C0 + #E = cambio derecho + izquierda "ARBOL" )

}

\* Esta lista es para quitar la tecla ALPHA + izquierda "UPDIR"

{ NULL$ ( Nombre: NULL$ para no asignar a ninguna tecla del MENU )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 10E ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista es para quitar la tecla ALPHA + derecha "DOWNDIR"

{ NULL$ ( Nombre: NULL$ para no asignar a ninguna tecla del MENU )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 110 ( Tecla\_Atajo: # 110 = #100 + #10 = ALPHA + Tecla derecha "DOWNDIR" )

}

}

FLASHPTR FILER\_MANAGERTYPE

;

Ejemplo 4 Filer

**Abrir el Filer para explorar sólo dentro del directorio HOME o sólo dentro de algúno de los puertos. No se podrá ir al ARBOL.**

Para hacer esto escoger cualquier Ruta\_Inicial que queramos.

• Si escoges el directorio HOME o uno de sus subdirectorios, entonces sólo podrás explorar dentro de HOME.

• Si escoges algún puerto, o algún directorio o backup dentro de un puerto, entonces sólo podrás explorar dentro de ese puerto especificado.

En el código de abajo se escogió NULL{} (el directorio HOME), pero puedes escoger otro.

A las 3 teclas predefinidas que hacen “UPDIR” se las cambió por un programa que devuelve BINT10 ó BINT0 (hacer UPDIR o hacer BEEP) de acuerdo al contexto.

A la tecla predefinida que hace “ARBOL” se le asignó “BEEP”.

ASSEMBLE

CON(1) 8 ( Le dice al parseador que el comando es 'No algebraico' )

RPL

xNAME FilerHomeYSubD ( -> F )

:: CK0

{ BINT0 } ( Filer\_Tipos: { BINT0 } = Todos los objetos )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* SOLO 1 TECLA VISIBLE EN EL MENU. Es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* LAS RESTANTES LISTAS SOLO SON ASIGNACIONES DE TECLAS QUE NO SE VEN EN EL MENU

\* Esta lista es para asignar a la tecla izquierda "UPDIR" ó "BEEP"

{ NULL$ ( Nombre: NULL$ para no asignar a ninguna tecla del MENU )

BINT0 ( Localización: BINT0= En cualquier lugar )

UPDIR\_NO\_ARBOL ( Acción: Programa que devuelve BINT10 ó BINT0 )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT14 ( Tecla\_Atajo: BINT14 = #E = Tecla izquierda "UPDIR" )

}

\* Esta lista es para asignar a la tecla cambio izq. + VAR "UPDIR" ó "BEEP"

{ NULL$ ( Nombre: NULL$ para no asignar a ninguna tecla del MENU )

BINT0 ( Localización: BINT0= En cualquier lugar )

UPDIR\_NO\_ARBOL ( Acción: Programa que devuelve BINT10 ó BINT0 )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT75 ( Tecla\_Atajo: BINT75 = #4B = #40 + #B = cambio izq. + VAR "UPDIR" )

}

\* Esta lista es para asignar a la tecla ALPHA + izquierda "UPDIR" ó "BEEP"

{ NULL$ ( Nombre: NULL$ para no asignar a ninguna tecla del MENU )

BINT0 ( Localización: BINT0= En cualquier lugar )

UPDIR\_NO\_ARBOL ( Acción: Programa que devuelve BINT10 ó BINT0 )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 10E ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista es para quitar la tecla cambio derecho + izquierda "ARBOL"

{ NULL$ ( Nombre: NULL$ para no asignar a ninguna tecla del MENU )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# CE ( Tecla\_Atajo: #CE = #C0 + #E = cambio derecho + izquierda "ARBOL" )

}

}

FLASHPTR FILER\_MANAGERTYPE

;

NULLNAME UPDIR\_NO\_ARBOL ( -> BINT0/BINT10 )

:: LAM 'FPPath ( {}/tag )

STRIPTAGS ( {} )

DUPNULL{}? ( {} flag )

casedrop BINT0 ( Sale con: BINT0 )

CARCOMP ( id/lam )

TYPELAM? ( flag )

ITE BINT0 BINT10 ( Sale con: BINT0 ó BINT10 )

;

Ejemplo 5 Filer

**Abrir el Filer para seleccionar sólo un objeto y que no se permita la marcación de objetos.**

En este ejemplo no se permite la marcación de objetos, por lo tanto, sólo se puede seleccionar un objeto.

Al presionar la tecla de menú “CANCL” o la tecla ON, entonces sale del filer y deja FALSE en la pila.

Al presionar la tecla de menú “OK” o la tecla ENTER, entonces sale del filer y deja el objeto seleccionado y TRUE en la pila.

Por supuesto, si estás en el arbol y presionas CANCL, ON o CHDIR, entonces sales del programa y se devuelve :0: {}

A las 3 teclas predefinidas que marcan o desmarcan a un objeto, se les asigno otra acción (se asignó un programa personalizado a la tecla ENTER y se asignó BEEP a las otras dos).

ASSEMBLE

CON(1) 8 ( Le dice al parseador que el comando es 'No algebraico' )

RPL

xNAME FilerSel1Obj ( -> ob T // F // :0:{} )

:: CK0

{ BINT0 } ( Filer\_Tipos: { BINT0 } = Todos los objetos )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* HAY 6 TECLAS DEL MENU

\* La 1º es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* Esta lista es para quitar la tecla +/- "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT28 ( Tecla\_Atajo: BINT28 = #1C = Tecla +/- "CHECK" )

}

\* Esta lista es para quitar la tecla ALPHA + ENTER "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 133 ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista no hace nada

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

}

\* Esta lista es para la tecla de menú F5 "CANCL"

{ # DF2C ( Nombre: # DF2C "CANCL" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT37 ( Acción: BINT37= Sale del filer dejando FALSE en la pila )

}

\* Esta lista es para la tecla de menú F6 "OK"

\* También es para asignarla a la tecla ENTER

{ # DF2D ( Nombre: # DF2D "OK" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT17 ( Acción: BINT17= Prog Pers que pone Ruta, Objeto y Nombre )

:: ( Ruta Objeto Nombre )

DROPSWAPDROP ( Objeto )

' TakeOver ( Objeto TakeOver )

; ( Programa\_Extra: Sale del filer, deja en la pila el Objeto y TRUE )

# 33 ( Tecla\_Atajo: BINT51 = #33 = Tecla ENTER )

}

}

FLASHPTR FILER\_MANAGERTYPE

;

Ejemplo 6 Filer

**Abrir el Filer para seleccionar sólo un objeto y que no se permita la marcación de objetos.**

**Además no se podrá devolver un directorio en la pila.**

Este ejemplo es similar al anterior.

La única diferencia es que al presionar la tecla de menú “OK” o la tecla ENTER, entonces sale del filer y deja el objeto seleccionado y TRUE en la pila, sólo si el objeto seleccionado no es un directorio. Si el objeto seleccionado es un directorio, sólo emite un sonido.

Sólo se ha cambiado el Programa\_Extra de la tecla OK del menú.

ASSEMBLE

CON(1) 8 ( Le dice al parseador que el comando es 'No algebraico' )

RPL

xNAME FilerSel1ObjND ( -> ob T // F // :0:{} )

:: CK0

{ BINT0 } ( Filer\_Tipos: { BINT0 } = Todos los objetos )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* HAY 6 TECLAS DEL MENU

\* La 1º es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* Esta lista es para quitar la tecla +/- "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT28 ( Tecla\_Atajo: BINT28 = #1C = Tecla +/- "CHECK" )

}

\* Esta lista es para quitar la tecla ALPHA + ENTER "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 133 ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista no hace nada

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

}

\* Esta lista es para la tecla de menú F5 "CANCL"

{ # DF2C ( Nombre: # DF2C "CANCL" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT37 ( Acción: BINT37= Sale del filer dejando FALSE en la pila )

}

\* Esta lista es para la tecla de menú F6 "OK"

\* También es para asignarla a la tecla ENTER

{ # DF2D ( Nombre: # DF2D "OK" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT17 ( Acción: BINT17= Prog Pers que pone Ruta, Objeto y Nombre )

:: ( Ruta Objeto Nombre )

DROPSWAPDROP ( Objeto )

DUPTYPERRP? ( Objeto flag )

caseDrpBadKy ( )

' TakeOver ( Objeto TakeOver )

; ( Programa\_Extra: Sale del filer, deja en la pila el Objeto y TRUE )

# 33 ( Tecla\_Atajo: BINT51 = #33 = Tecla ENTER )

}

}

FLASHPTR FILER\_MANAGERTYPE

;

Capítulo 30  
Verificación de Argumentos

En System RPL, es muy importante verificar que todos los argumentos requeridos por un programa se encuentren en la pila, y también verificar si esos objetos son de un tipo válido, cuando ese programa sea accesible al usuario. En User RPL no tenemos que preocuparnos de esto, pues es hecho de manera automática. En System RPL, muy pocos comandos hacen esto, de tal manera que esta tarea es dejada para el programador. A primera vista, esto podría parecer una desventaja, pero esto es en realidad una ventaja: tu sólo necesitas verificar los argumentos sólo una vez, al inicio del programa. Esto generará un código rápido, contrariamente al User RPL, donde los argumentos son verificados por cada comando.

30.1 Número de Argumentos

Para verificar por un número específico de argumentos, usa uno de los siguientes comandos. Estos comandos verifican si hay suficientes argumentos en la pila y generan el error: “Muy pocos argumentos” si esto no es asi.

**Comando Cuando usarlos**

CK0, CK0NOLASTWD No se requieren argumentos

CK1, CK1NOLASTWD Un argumento es requerido

CK2, CK2NOLASTWD Dos argumentos son requeridos

CK3, CK3NOLASTWD Tres argumentos son requeridos

CK4, CK4NOLASTWD Cuatro argumentos son requeridos

CK5, CK5NOLASTWD Cinco argumentos son requeridos

Los comandos CK<n> guardan el nombre del comando en el cual ellos se están ejecutando, y si un error ocurre, este nombre es mostrado (para más detalles ver capítulo 23). Esto significa que los comandos CK<n> deben ser usados solo en bibliotecas, porque si ellos no son parte de una biblioteca y sucede un error, el error será mostrado de una forma parecida a esta: “XLIB 1364 36 Error:”. En programas que no son parte de una biblioteca, debes usar CK<n>NOLASTWD, los cuales no guardan el nombre del comando.

Además de verificar por el número especificado de argumentos, estos comandos también marcan la pila de una forma tal, que si un error sucede, los objetos que fueron puestos en la pila por tu programa serán removidos y de esta manera no se dejan trastos en la pila al ocurrir el error.

Esto funciona “marcando” la pila sobre el nivel “n”, donde “n” es el número de argumentos requeridos. Por ejemplo, si tu programa usa CK2 o CK2NOLASTWD y hay tres argumentos en la pila, puedes imaginarte la pila de esta manera:

3: 10.

2: 3.

1: 5.5

Esta marca no esta fija en ese nivel. En vez de eso, esta marca se mueve si unos elementos son agregados o retirados de la pila. Aquí está la pila después de que el programa pone el bint1:

4: 10.

3: 3.

2: 5.5

1: ¤ 1h

Ahora, si un error sucede en el programa, entonces los objetos que estaban debajo de la marca antes de ejecutar el comando, son devueltos a sus posiciones iniciales. Pero esto no sucederá con los objetos que estaban encima de la marca. De esta forma, si queremos que la pila en su totalidad vuelva a su estado inicial (cuando un error sucede y estamos usando un comando CK<n>), no debemos manipular los objetos que están por encima de la marca.

|  |  |
| --- | --- |
| Comando que origina error y no usa CK<n> | Comando que origina error y usa CK<n> |
| xNAME ErrorSinCKn ( %a %b -> ERROR )  :: ( %a %b )  \* Verifica nº argumentos sin CK2  DEPTH BINT2 #<case SETSTACKERR  \* Verifica 2 números reales  OVER TYPEREAL?  OVER TYPEREAL?  AND  NcaseTYPEERR  \* Continua el programa  ( %a %b )  %+ ( %a+b )  %10 ( %a+b %10 )  %\* ( %10[a+b] )  %12 ( %10[a+b] %12 )  %0 ( %10[a+b] %12 %0 )  %/  ; | xNAME ErrorConCKn ( %a %b -> ERROR )  :: ( %a %b )  \* Verifica nº argumentos con CK2  CK2  \* Verifica 2 números reales  OVER TYPEREAL?  OVER TYPEREAL?  AND  NcaseTYPEERR  \* Continua el programa  ( %a %b )  %+ ( %a+b )  %10 ( %a+b %10 )  %\* ( %10[a+b] )  %12 ( %10[a+b] %12 )  %0 ( %10[a+b] %12 %0 )  %/  ; |
| Estado de la pila antes del comando | Estado de la pila antes del comando |
| NIVEL 3: 6.  NIVEL 2: 7.  NIVEL 1: 8. | NIVEL 3: 6.  NIVEL 2: 7.  NIVEL 1: 8. |
| Estado de la pila después del comando | Estado de la pila después del comando  (con flag 55 desactivado: save last args) |
|  |  |

Además de verificar por un número de argumentos en la pila y proveerlos para un restablecimiento de esta en caso de errores, estos comandos también guardan los argumentos como los últimos argumentos, los que se pueden llamar por medio del comando LASTARG de User RPL (cuando el flag 55 está desactivado). Si un error ocurre y flag 55 está desactivado, entonces los argumentos son restaurados automáticamente.

Para programas accesibles al usuario y que no tomen argumentos de la pila, deberás sin embargo, usar el comando CK0 (o CK0NOLASTWD si el programa no es parte de una biblioteca), para marcar todos los objetos de la pila como de propiedad del usuario y marcar la pila para un restablecimiento de esta en caso de errores. Si tu programa usa un número de argumentos definido por un número situado en el nivel 1 de la pila (como DROPN), usa los comandos CKN o CKNNOLASTWD. Estos comandos primero verifican que haya un número real en el nivel 1 de la pila, y luego verifican que exista el número especificado de objetos en la pila. La pila es marcada en el nivel 2, pero sólo el número real es guardado en LAST ARG. El real es convertido a bint.

30.2 Tipo de Argumento

Los comandos CK&DISPATCH1 y CK&DISPATCH0 son usados para permitir que tu programa haga acciones diferentes basadas en el tipo de argumentos presentes en la pila. Estos comandos se usan de la siguiente manera:

::

CK&DISPATCH0 ( En su lugar puede ir CK&DISPATCH1 )

#tipo1

acción1

#tipo2

acción2

#tipo3

acción3

...

...

#tipo\_n

acción\_n

;

Si después de despachar los argumentos y realizar su acción correspondiente, quieres hacer más acciones para todos los tipos de argumentos, deberás encerrar el bloque mostrado en un objeto programa y realizar las acciones posteriores fuera de ese programa.

Asi es como funciona el comando CK&DISPATCH0: verifica si los objetos de la pila corresponden con los exigidos en #tipo1. Si es asi, entonces acción1 es ejecutada y después se pasa por alto el resto del programa (es decir, la ejecución del programa pasa a estar después de SEMI). Cada acción debe ser un objeto único, de manera que si quieres realizar varias acciones, todas estas deben estar dentro de un objeto programa (Es decir, encerradas por los delimitadores :: y ;). Si los objetos presentes en la pila no corresponden a los exigidos en #tipo1, entonces se comprueba si corresponden a los exigidos en #tipo2 y asi sucesivamente. Si los objetos que están en la pila no corresponden a ninguno de los tipos exigidos, entonces se genera el error “Argumento incorrecto”.

Aun si tu programa acepta solo una combinación de argumentos, este comando es útil para verificar si los argumentos son del tipo deseado.

La diferencia entre CK&DISPATCH0 y CK&DISPATCH1 es que el último, después de comparar los argumentos sin éxito, quita las etiquetas de los argumentos y compara nuevamente. Si todavía no tiene éxito, convierte los enteros a reales y compara nuevamente. Sólo después de no tener éxito en la última comparación, manda el mensaje de error “Argumento incorrecto”. Cada definición de tipo es un bint de la forma #nnnnn. Cada n es un número hexadecimal que representa a un objeto en alguna posición en la pila, de acuerdo a la tabla mostrada más adelante. El primer n representa el objeto en el nivel 5, el segundo n representa el objeto en el nivel 4, y asi sucesivamente. De esta forma, #00201 representa un número complejo en el nivel 3, cualquier objeto en el nivel 2 y un número real en el nivel 1. Por otra parte, #000A6 representa un hxs en el nivel 2 y un id en el nivel 1. También hay objetos cuyo tipo tiene representación hexadecimal con dos dígitos, el último de estos es F. Cada vez que usas uno de estos, el número de objetos que puede ser verificado se reduce. Por ejemplo, #13F4F representa un número real en el nivel 3, un real extendido en el nivel 2 y un complejo extendido en el nivel 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Código**  **Dispatch** | **Tipo**  **User** | **Tipo de objeto** | **Filer** | **Prólogo** | **Abrev.**  **Prólogo** |
| BINT1=# 1 | 0. | Números reales | REAL | #2933 | TYPEREAL |
| BINT2=# 2 | 1. | Números complejos | CPLX | #2977 | TYPECMP\_ |
| BINT3=# 3 | 2. | Cadena | STRNG | #2A2C |  |
| BINT4=# 4 | 3. | Arreglos reales | ARRAY | #29E8 |  |
|  | 4. | Arreglos no reales | ARRAY | #29E8 |  |
|  | 29. | Matrices simbólicas | MATRX | #2686 | TYPEMATRIX\_ |
| BINT5=# 5 | 5. | Lista | LIST | #2A74 | TYPELIST\_ |
| BINT6=# 6 | 6. | Nombre global | GNAME | #2E48 | TYPEIDNT |
| BINT7=# 7 | 7. | Nomble Local | LNAME | #2E6D | TYPELAM\_ |
| BINT8=# 8 | 8. | Programa | PROG | #2D9D | TYPECOL\_ |
|  | 18. | Pointer comando de User RPL permitido en algebraicos | XLIB | #2D9D | TYPECOL\_ |
|  | 19. | Pointer comando de User RPL no permitido en algebraicos | XLIB | #2D9D | TYPECOL\_ |
| BINT9=# 9 | 9. | Simbólico | ALG | #2AB8 | TYPESYMB\_ |
| BINT10=# A |  | Clase simbólica: id, lam, symb, enteros. |  |  |  |
| BINT11=# B | 10. | Cadenas hexadecimales (hxs) | BIN | #2A4E |  |
| BINT12=# C | 11. | Gráfico (grob) | GROB | #2B1E |  |
| BINT13=# D | 12. | Etiquetado (tagged) | TAG | #2AFC |  |
| BINT14=# E | 13. | Unidades | UNIT | #2ADA | TYPEEXT\_ |
| BINT15=# F | 14. | Rompointer | XLIB | #2E92 |  |
| BINT31=#1F | 20. | Entero binario (bint) | SB | #2911 |  |
| BINT47=#2F | 15. | Directorio | DIR | #2A96 | TYPERRP\_ |
| #3F | 21. | Número real extendido | LREAL | #2955 | TYPEEREL\_ |
| #4F | 22. | Número complejo extendido | LCPLX | #299D |  |
| #5F | 23. | Arreglo vinculado | LARRY | #2A0A |  |
| #6F | 24. | Carácter | CHAR | #29BF |  |
| #7F | 25. | Code | CODE | #2DCC |  |
| #8F | 16. | Biblioteca | L•••• | #2B40 |  |
| #9F | 17. | BACKUP | BCKUP | #2B62 |  |
| #AF | 26. | Library Data | LBDAT | #2B88 |  |
| #BF | 27. | Access pointer (Extended Ptr) | EXPTR | #2BAA |  |
| #CF | 30. | Fuente normal | FONT | #2BCC |  |
| #DF | 27. | Minifuente | MFONT | #26FE |  |
| #EF | 27. | External object 4 (Extended 3) | EXT1 | #2C10 |  |
| BINT255d=#FF | 28. | Entero | INTG | #2614 | TYPEINT\_ |
| BINT0=# 0 | 27. | Flashpointer | FLSHP | #26AC |  |
| BINT0=# 0 | 27. | Pointer | EXTER |  |  |
| BINT0=# 0 | 27. | Aplet | EXTER | #26D5 |  |
| BINT0=# 0 | 27. | External object 3 (Extended 2) | AUNIT | #2BEE |  |

Existen también los comandos CK<n>&Dispatch, donde <n> es un número del 1 al 5. Estos comandos combinan CK<n> con CK&DISPATCH1. Debido a que estos comandos usan CK<n> (y de esta manera guardan el nombre del último comando), deberán usarse solamente en comandos de biblioteca.

30.2.1 Comando TYPE

Descompilando y estudiando los comandos ya incorporados en la ROM, puedes aprender mucho. No solo sobre verificación de argumentos, sino también sobre muchas otras cosas. El comando TYPE de User RPL nos provee un ejemplo de como despachar objetos. Aquí está su descompilación:

\* Comando TYPE de User RPL ( ob -> %tipo )

::

CK1

::

CK&DISPATCH0

BINT1 %0 ( real )

BINT2 %1 ( complejo )

BINT3 %2 ( cadena )

BINT4 ( arreglo o matriz simbólica )

:: DUP

TYPERARRY?

case

%3 ( arreglo real )

DUPTYPEMATRIX?\_

NOTcase

%4 ( arreglo no real )

%29\_ ( matriz simbólica )

;

BINT5 %5 ( lista )

BINT6 %6 ( id )

BINT7 %7 ( lam )

BINT8

:: DUPROM-WORD?

NOTcase

%8 ( ob programa )

?OKINALG

case

%18 ( comando user permitido en algebraicos )

%19 ( comando user no permitido en algebraicos )

;

BINT9 %9 ( simbólico )

BINT11 %10 ( hxs )

BINT12 %11 ( grob )

BINT13 %12 ( ob etiquetado )

BINT14 %13 ( ob unidad )

BINT15 %14 ( rompointer )

BINT47 %15 ( directorio ) ( BINT47 es # 2F )

#8F\_ %16 ( biblioteca )

backup\_ %17 ( backup ) ( backup\_ apunta a # 9F )

BINT31 %20 ( bint ) ( BINT31 es # 1F )

# 3F %21 ( real extendido )

# 4F %22 ( complejo extendido )

# 5F %23 ( arreglo vinculado )

# 6F %24 ( carácter )

# 7F %25 ( objeto code )

# AF %26 ( Library Data )

BINT255d %28\_ ( Entero ) ( BINT255d apunta a # FF )

# CF %30\_ ( fuente de sistema )

BINT0 %27 ( Otros: minifuente, FlashPtr, AccessPtr, ... )

;

SWAPDROP

;

En este caso, pudo haberse usado el comando CK&DISPATCH1 en lugar de CK&DISPATCH0, porque los objetos etiquetados y los enteros son despachados en el programa.

Debido a que el último ítem despachado es para un objeto de cualquier tipo (#0), entonces el número real 27 es retornado cuando en la pila hay un objeto que no haya sido despachado antes.

Debido a que el comando TYPE de User RPL es parte de una biblioteca ya incorporada en la ROM de la calculadora, el comando CK1&Dispatch pudo haber sido usado. La razón por la cual no se ha usado ese comando es para ahorrar espacio en la ROM. El compuesto interno es actualmente el cuerpo del comando XEQTYPE. De esta forma, los programadores de System RPL pueden llamar a una función para retornar el tipo del objeto, sin necesidad de verificar si ya hay un objeto en la pila, y sin necesidad de duplicar el mecanismo de despachar objetos.

30.3 Referencia

**Direcc. Nombre Descripción**

262B0 CK0 ( 🡺 )

Guarda comando actual a LASTCKCMD.

Marca la pila debajo del nivel 1 a STACKMARK.

262B5 CK1 ( ob 🡺 ob )

Guarda comando actual a LASTCKCMD.

Verifica que exista por lo menos un objeto en la pila.

Si no es así, genera el error "Muy pocos argumentos".

Guarda marca de la pila a STACKMARK.

Si Last Arg está activado (flag 55 desactivado), entonces guarda el argumento.

262BA CK2 ( ob1 ob2 🡺 ob1 ob2 )

Como CK1, pero verifica por al menos dos argumentos.

262BF CK3 ( ob1...ob3 🡺 ob1...ob3 )

Como CK1, pero verifica por al menos tres argumentos.

262C4 CK4 ( ob1...ob4 🡺 ob1...ob4 )

Como CK1, pero verifica por al menos cuatro argumentos.

262C9 CK5 ( ob1...ob5 🡺 ob1...ob5 )

Como CK1, pero verifica por al menos cinco argumentos.

262CE CKN ( ob1...obn %n 🡺 ob1..obn #n )

Verifica por un número real en el nivel 1 de la pila.

Luego verifica por ese número de argumentos.

Finalmente, convierte ese real a bint.

26292 CK0NOLASTWD ( 🡺 )

Como CK0, pero no guarda el nombre del comando actual.

26297 CK1NOLASTWD ( ob 🡺 ob )

Como CK1, pero no guarda el nombre del comando actual.

2629C CK2NOLASTWD ( ob1 ob2 🡺 ob1 ob2 )

Como CK2, pero no guarda el nombre del comando actual.

262A1 CK3NOLASTWD ( ob1...ob3 🡺 ob1...ob3 )

Como CK3, pero no guarda el nombre del comando actual.

262A6 CK4NOLASTWD ( ob1...ob4 🡺 ob1...ob4 )

Como CK4, pero no guarda el nombre del comando actual.

262AB CK5NOLASTWD ( ob1...ob5 🡺 ob1...ob5 )

Como CK5, pero no guarda el nombre del comando actual.

25F25 CKNNOLASTWD ( ob1...obn %n 🡺 ob1..obn #n )

Como CKN, pero no guarda el nombre del comando actual.

2631E CK&DISPATCH0 ( 🡺 )

Despacha acciones de acuerdo a los argumentos en la pila.

26328 CK&DISPATCH1 ( 🡺 )

Despacha acciones de acuerdo a los argumentos en la pila, quitando las etiquetas y convirtiendo enteros a reales si es necesario.

26323 CK&DISPATCH2 ( 🡺 )

Equivalente a CK&DISPATCH1.

26300 CK1&Dispatch ( 🡺 )

Combina CK1 con CK&DISPATCH1.

26305 CK2&Dispatch ( 🡺 )

Combina CK2 con CK&DISPATCH1.

**Direcc. Nombre Descripción**

2630A CK3&Dispatch ( 🡺 )

Combina CK3 con CK&DISPATCH1.

2630F CK4&Dispatch ( 🡺 )

Combina CK4 con CK&DISPATCH1.

26314 CK5&Dispatch ( 🡺 )

Combina CK5 con CK&DISPATCH1.

25F9A 0LASTOWDOB! ( 🡺 )

Limpia el nombre del comando guardado por el último comando CK<n>.

aka: 0LastRomWrd!

2EF6C AtUserStack ( 🡺 )

:: CK0NOLASTWD 0LASTOWDOB! ;

25E9E CK1NoBlame ( 🡺 )

:: 0LASTOWDOB! CK1NOLASTWD ;

354CB 'RSAVEWORD ( 🡺 )

Guarda el primer objeto del compuesto que está sobre el actual a LASTCKCMD.

aka: 'RSaveRomWrd

26319 EvalNoCK ( comp 🡺 ? )

Evalúa el compuesto sin guardarlo como el comando actual.

Si el primer comando es CK<n>&Dispatch este es reemplazado por CK&DISPATCH1.

Si el primer comando es CK<n>, éste es pasado por alto.

Cualquier otro comando es también pasado por alto.

25F29 (EvalNoCK:) Run Stream:

( ob 🡺 )

Hace EvalNoCK con el siguiente objeto del runstream como argumento.

aka: 'EvalNoCK:, 'EvalNoCK:\_sup\_

2A9E9 RunRPN: Run Stream:

( ob 🡺 )

Evalua el siguiente objeto del runstream en modo RPN (flag 95 desactivado). Después de la evaluación el flag 95 es restaurado a su valor antiguo.

26D006 ˆCK1TONOext ( ob 🡺 ob' )

Applies prg to ob, recursively for lists. prg is fetched from runstream.

30.3.1 Verificando el Tipo de un Objeto

**Direcc. Nombre Descripción**

36B7B CKREAL ( % 🡺 % )

( Z 🡺 % )

Verifica que se trate de un número real.

Si es un número real, no hace nada.

Si es un número entero, lo convierte a real.

Para objetos de otro tipo, se genera el error “Argumento incorrecto”

184006 ˆCK1Z ( $/Z/hxs 🡺 Z )

Verifica por un número entero en la pila.

Si hay un entero, no hace nada.

Si hay una cadena apropiada, la convierte a entero.

También convierte algunos hxs.

**Direcc. Nombre Descripción**

185006 ˆCK2Z ( ob ob' 🡺 Z Z' )

Como ˆCK1Z, pero para dos objetos.

186006 ˆCK3Z ( ob ob' ob'' 🡺 Z Z' Z'' )

Como ˆCK1Z, pero para tres objetos.

3D2B4 CKSYMBTYPE ( ob 🡺 ob )

Si ob es un simbólico que contiene solo a un nombre global o local, no hace nada.

De lo contrario, genera el error “Argumento incorrecto”.

Por ejemplo, si en la pila se encuentra uno de estos simbólicos:

SYMBOL ID X ;

SYMBOL LAM P24 ;

entonces no manda mensaje de error alguno.

2EF07 nmetasyms ( ob1...obn #n 🡺 ob1...obn )

Verifica que los n objetos de la pila sean de tipo real, complejo, unidad o de clase simbólica (id, lam, symb o entero). Si esto no se cumple, genera el error “Argumento incorrecto”.

03C64 TYPE ( ob 🡺 #prolog )

Retorna la dirección del prólogo del objeto, para reales, complejos, unidades, objetos de clase simbólica (id, lam, symb o enteros), matrices simbólicas, listas, programas, directorios y reales extendidos.

No usar este comando si en la pila hay un objeto de otro tipo.

Por ejemplo:

:: 35.36 TYPE TOTEMPOB ; retorna # 2933

3BC43 XEQTYPE ( ob 🡺 ob %tipo )

Hace lo mismo que el comando TYPE de User RPL.

La diferencia es que XEQTYPE mantiene a ob en la pila.

3511D TYPEREAL? ( ob 🡺 flag )

¿Número real?

35118 DUPTYPEREAL? ( ob 🡺 ob flag )

¿Número real?

aka: DTYPEREAL?

3512C TYPECMP? ( ob 🡺 flag )

¿Número complejo?

35127 DUPTYPECMP? ( ob 🡺 ob flag )

¿Número complejo?

3510E TYPECSTR? ( ob 🡺 flag )

¿Cadena?

35109 DUPTYPECSTR? ( ob 🡺 ob flag )

¿Cadena?

aka: DTYPECSTR?

3513B TYPEARRY? ( ob 🡺 flag )

¿Arreglo?

35136 DUPTYPEARRY? ( ob 🡺 ob flag )

¿Arreglo?

aka: DTYPEARRY?

35292 TYPERARRY? ( ob 🡺 flag )

¿Arreglo real?

352AD TYPECARRY? ( ob 🡺 flag )

¿Arreglo complejo?

**Direcc. Nombre Descripción**

35195 TYPELIST? ( ob 🡺 flag )

¿Lista?

35190 DUPTYPELIST? ( ob 🡺 ob flag )

¿Lista?

aka: DTYPELIST?

3504B TYPEIDNT? ( ob 🡺 flag )

¿Nombre global?

35046 DUPTYPEIDNT? ( ob 🡺 ob flag )

¿Nombre global?

350E1 TYPELAM? ( ob 🡺 flag )

¿Nombre local?

350DC DUPTYPELAM? ( ob 🡺 ob flag )

¿Nombre local?

194006 ˆTYPEIDNTLAM? ( ob 🡺 flag )

¿Nombre global o local?

Retorna TRUE si ob es un nombre global o un nombre local.

191006 ˆIDNTLAM? ( ob 🡺 ob flag )

¿Nombre global o local?

Agrega TRUE si ob es un nombre global o un nombre local.

2F0D4 (NotIDorLAM?) ( ob 🡺 ob flag )

¿NO ES nombre global o local?

Agrega TRUE si ob no es nombre global ni local.

35168 TYPESYMB? ( ob 🡺 flag )

¿Objeto simbólico?

35163 DUPTYPESYMB? ( ob 🡺 ob flag )

¿Objeto simbólico?

350FF TYPEHSTR? ( ob 🡺 flag )

¿Cadena hexadecimal?

350FA DUPTYPEHSTR? ( ob 🡺 ob flag )

¿Cadena hexadecimal?

35186 TYPEGROB? ( ob 🡺 flag )

¿Objeto gráfico (grob)?

35181 DUPTYPEGROB? ( ob 🡺 ob flag )

¿Objeto gráfico (grob)?

351A4 TYPETAGGED? ( ob 🡺 flag )

¿Objeto etiquetado?

3519F DUPTYPETAG? ( ob 🡺 ob flag )

¿Objeto etiquetado?

351B3 TYPEEXT? ( ob 🡺 flag )

¿Objeto unidad?

351AE DUPTYPEEXT? ( ob 🡺 ob flag )

¿Objeto unidad?

3514A TYPEROMP? ( ob 🡺 flag )

¿Rompointer?

35145 DUPTYPEROMP? ( ob 🡺 ob flag )

¿Rompointer?

350F0 TYPEBINT? ( ob 🡺 flag )

¿Entero binario (bint)?

350EB DUPTYPEBINT? ( ob 🡺 ob flag )

¿Entero binario (bint)?

**Direcc. Nombre Descripción**

35159 TYPERRP? ( ob 🡺 flag )

¿Directorio?

35154 DUPTYPERRP? ( ob 🡺 ob flag )

¿Directorio?

3505A TYPEBAK?\_ ( ob 🡺 flag )

¿Objeto de respaldo (Backup)?

35055 DUPTYPEBAK?\_ ( ob 🡺 ob flag )

¿Objeto de respaldo (Backup)?

3503C TYPECHAR? ( ob 🡺 flag )

¿Caracter?

35037 DUPTYPECHAR? ( ob 🡺 ob flag )

¿Caracter?

35069 TYPELIB?\_ ( ob 🡺 flag )

¿Biblioteca?

35064 DUPTYPELIB?\_ ( ob 🡺 ob flag )

¿Biblioteca?

351C2 TYPEEXT0?\_ ( ob 🡺 flag )

¿Library Data?

351BD DUPTYPEEXT0?\_ ( ob 🡺 ob flag )

¿Library Data?

02F0E7 ˜UTTYPEEXT0? ( ob 🡺 flag )

¿Library Data?

35177 TYPECOL? ( ob 🡺 flag )

¿Programa o pointer que contiene un programa?

35172 DUPTYPECOL? ( ob 🡺 ob flag )

¿Programa o pointer que contiene un programa?

aka: DTYPECOL?

350D2 TYPEAPLET? ( ob 🡺 flag )

¿Aplet?

350CD DUPTYPEAPLET? ( ob 🡺 ob flag )

¿Aplet?

35087 TYPEFLASHPTR? ( ob 🡺 flag )

¿Flashpointer?

35082 DUPTYPEFLASHPTR? ( ob 🡺 ob flag )

¿Flashpointer?

350C3 TYPEFONT? ( ob 🡺 flag )

¿Fuente de sistema?

350BE DUPTYPEFONT? ( ob 🡺 ob flag )

¿Fuente de sistema?

35078 TYPEMATRIX? ( ob 🡺 flag )

¿Matriz simbólica?

35073 DUPTYPEMATRIX? ( ob 🡺 ob flag )

¿Matriz simbólica?

aka: DTYPEMATRIX?

350B4 TYPELNGCMP? ( ob 🡺 flag )

¿Número complejo extendido?

350AF DUPTYPELNGCMP? ( ob 🡺 ob flag )

¿Número complejo extendido?

350A5 TYPELNGREAL? ( ob 🡺 flag )

¿Número real extendido?

350A0 DUPTYPELNGREAL? ( ob 🡺 ob flag )

¿Número real extendido?

**Direcc. Nombre Descripción**

35096 TYPEZINT? ( ob 🡺 flag )

¿Entero?

35091 DUPTYPEZINT? ( ob 🡺 ob flag )

¿Entero?

182006 ˆTYPEZ? ( ob 🡺 flag )

¿Entero?

183006 ˆDUPTYPEZ? ( ob 🡺 ob flag )

¿Entero?

196006 ˆTYPEREALZINT? ( ob 🡺 flag )

¿Número real, entero o hxs?

195006 ˆREAL? ( ob 🡺 ob flag )

¿Número real, entero o hxs?

25E77 ?OKINALG ( ob 🡺 ob flag )

¿Es un objeto permitido en algebraicos?

¿Número real, complejo, unidad, clase simbólica (id, lam, symb o entero) o comandos de User RPL (pointers o rompointers) permitidos en algebraicos?

192006 ˆFLOAT? ( ob 🡺 ob flag )

Agrega TRUE, si ob es real, complejo, real extendido o complejo extendido.

Agrega FALSE, si ob es una lista o un objeto de clase simbólica (id, lam, symb o entero).

Para objetos de otro tipo genera el error “Argumento incorrecto”

181006 ˆCKALG ( ob 🡺 ob )

Si ob es real, complejo, unidad o de clase simbólica (id, lam, symb o entero) no hace nada.

Si ob es un objeto de otro tipo, genera el error

“Argumento incorrecto”

193006 ˆCKSYMREALCMP ( ob 🡺 ob )

Si ob es real, complejo o de clase simbólica (id, lam, symb o entero) no hace nada.

Si ob es un objeto de otro tipo, genera el error

“Argumento incorrecto”

3F33F CKARRY\_ ( ob 🡺 ob )

Si ob es un arreglo o una matriz simbólica, no hace nada.

Si ob es un objeto de otro tipo, manda el mensaje de error “Argumento incorrecto”

3F3C1 CKLIST\_ ( ob 🡺 ob )

Si ob es una lista, no hace nada.

Si ob es un objeto de otro tipo, manda el mensaje de error “Argumento incorrecto”

30.4 Ejemplos con Verificación de Argumentos

**Ejemplo 1 Verificación**

**Lista con números reales.**

Con el siguiente programa puedes saber si el objeto del nivel 1 de la pila es una lista cuyos elementos son todos números reales.

\* Retorna TRUE si en la pila hay una lista no vacía que

\* contiene como elementos sólo a números reales

NULLNAME ListaReal ( ob -> flag )

::

DUPTYPELIST? ( ob flag )

NOTcase DROPFALSE

DUPNULL{}? ( {} flag )

casedrpfls

( {} )

' :: TYPEREAL? NOT ; ( {} ProgTest\_1arg )

Find1stTrue ( ob T // F ) ( TRUE si un obj de la lista cumple test )

ITE

DROPFALSE

TRUE

( flag )

;

**Ejemplo 2 Verificación**

**Lista con números reales o enteros.**

Con el siguiente programa puedes saber si el objeto del nivel 1 de la pila es una lista cuyos elementos son todos números reales o enteros. Si se cumple eso, los números enteros son convertidos a reales y retornados junto a TRUE. De lo contrario, sólo se retorna FALSE.

NULLNAME ListaRealZ ( {} -> {%} T // F )

:: ( ob )

DUPTYPELIST? ( ob flag )

NOTcase DROPFALSE

( {} )

DUPNULL{}? ( {} flag )

casedrpfls

( {} )

TRUE ( {} T )

1LAMBIND ( {} )

INNERDUP ( ob1,...,obn #n #n )

ZERO\_DO ( ... #n )

ROLL ( ... obi )

DUPTYPEZINT? ( ... obi flag )

IT

FLASHPTR Z>R

( ... obi )

DUPTYPEREAL? ( ... obi flag )

ISTOP@ ( ... obi flag #n )

SWAP ( ... obi #n flag )

NOT\_IT

:: ExitAtLOOP FALSE 1PUTLAM ;

( ... obi #n )

LOOP

( ob1,...,obn #n )

1GETABND ( ob1,...,obn #n flag )

ITE

:: {}N TRUE ;

NDROPFALSE

;

**Ejemplo 3 Verificación**

**Lista con números positivos, reales o enteros.**

Con el siguiente programa puedes saber si el objeto del nivel 1 de la pila es una lista cuyos elementos son todos números reales o enteros positivos. Si se cumple eso, los números enteros son convertidos a reales y retornados junto a TRUE. De lo contrario, sólo se retorna FALSE.

NULLNAME ListaRealZ>0 ( {} -> {%} T // F )

:: ( {} )

DUPTYPELIST? ( ob flag )

NOTcase DROPFALSE

DUPNULL{}? ( {} flag )

casedrpfls

TRUE ( {} T )

1LAMBIND ( {} )

INNERDUP ( ob1,...,obn #n #n )

ZERO\_DO ( ... #n )

ROLL ( ... obi )

DUPTYPEZINT? ( ... obi flag )

IT

FLASHPTR Z>R

( ... obi )

DUPTYPEREAL? ( ... obi flag )

ISTOP@ ( ... obi flag #n )

SWAP ( ... obi #n flag )

ITE

:: OVER ( ... %i #n %i )

%0> ( ... %i #n flag )

NOT\_IT

:: ExitAtLOOP FALSE 1PUTLAM ;

;

:: ExitAtLOOP FALSE 1PUTLAM ;

( ... obi #n )

LOOP

( ob1,...,obn #n )

1GETABND ( ob1,...,obn #n flag )

ITE

:: {}N TRUE ;

NDROPFALSE

;

**Ejemplo 4 Verificación**

**Usando el NULLNAME TodosTrueComp del capítulo 11 para hacer verificaciones de los elementos de una lista.**

El NULLNAME TodosTrueComp ( {} Test\_1arg 🡪 flag ) del capítulo 11 evalúa el test de un argumento (pointer, rompointer o programa) a cada elemento de la lista y si para todos los elementos el resultado fue TRUE, retorna TRUE en la pila. Si para algún elemento el resultado de la evaluación fue FALSE, retorna FALSE.

En la pila debe haber una lista no vacía.

Podemos usar este NULLNAME para saber si todos los elementos de una lista cumplen una condición (todos reales, todos reales o enteros, todos positivos, etc).

\* Retorna TRUE si todos los elementos de la lista son reales

\* Debe haber una lista no vacía en la pila

NULLNAME {}%? ( {} -> flag )

::

' TYPEREAL?

TodosTrueComp

;

\* Retorna TRUE si todos los elementos de la lista son reales o enteros

\* Debe haber una lista no vacía en la pila

NULLNAME {}%Z? ( {} -> flag )

::

' :: DUPTYPEREAL? SWAP TYPEZINT? OR ;

TodosTrueComp

;

\* Retorna TRUE si todos los elementos de la lista son

\* reales o enteros positivos.

\* Debe haber una lista no vacía en la pila.

NULLNAME {}%Z>0? ( {} -> flag )

::

' :: DUPTYPEZINT? ( ob flag )

IT

FLASHPTR Z>R ( %/ob )

DUPTYPEREAL? ( %/ob )

NOTcaseFALSE ( SALE CON FALSE )

( % )

%0> ( flag )

;

TodosTrueComp

;

\* Retorna TRUE si todos los elementos de la lista son

\* reales o enteros del intervalo [90;180]

\* Debe haber una lista no vacía en la pila.

NULLNAME {}%Z[90,180]? ( {} -> flag )

::

' :: DUPTYPEZINT? ( ob flag )

IT

FLASHPTR Z>R ( %/ob )

DUPTYPEREAL? ( %/ob )

NOTcaseFALSE ( SALE CON FALSE )

( % )

DUP ( % % )

% 90. %>= ( % flag )

SWAP ( flag % )

% 180. %<= ( flag flag' )

AND ( flag'' )

;

TodosTrueComp

;

**Ejemplo 5 Verificación**

**El comando CK&DISPATCH0**

El siguiente ejemplo muestra el uso del comando CK&DISPATCH0.

El comando CK1 verifica que exista un argumento en la pila, marca la pila y guarda el nombre del comando EjCKDisp0.

El comando CK&DISPATCH0, despacha las acciones de la pila de acuerdo al objeto presente en la pila.

Si hay un real, retorna su valor absoluto.

Si hay un complejo, retorna el valor absoluto de su parte real.

Si hay un objeto de otro tipo, genera el error “Argumento incorrecto”.

Finalmente el programa muestra el resultado en la parte superior y congela la pantalla.

Si se hubiera usado el comando CK&DISPATCH1 en lugar de CK&DISPATCH0 y si en la pila hubiera un entero, este sería convertido a real (pues no se ha puesto ninguna acción para un objeto entero en el programa) y luego se hubiera tomado su valor absoluto.

Observa que el comando CK1 debe ser el primer objeto de EjCKDisp0 para que pueda guardar el nombre del comando.

\* Retorna TRUE si todos los elementos de la lista son

\* reales o enteros del intervalo [90;180]

\* Debe haber una lista no vacía en la pila.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME EjCKDisp0 ( %/C% -> )

::

CK1 ( Verifica si hay 1 objeto en la pila )

CLEARLCD ( Limpia toda la pantalla )

:: CK&DISPATCH0

BINT1

%ABS

BINT2

:: C>Re% %ABS ;

;

DO>STR

DISPROW1 ( Muestra cadena en la pantalla )

SetDAsTemp ( Congela toda la pantalla )

;

Capítulo 31  
Control del Teclado

Hay varias formas en que un programa hecho con System RPL puede conseguir lo que ingresa un usuario:

• Desde la pila.

• Esperando la opresión de botones en el teclado.

• Usando la versión interna del comando INPUT.

• Usando la version interna del comando INFORM.

• Creando aplicaciones con el Bucle Externo Parametrizado (llamado también POL).

• Y con otros métodos.

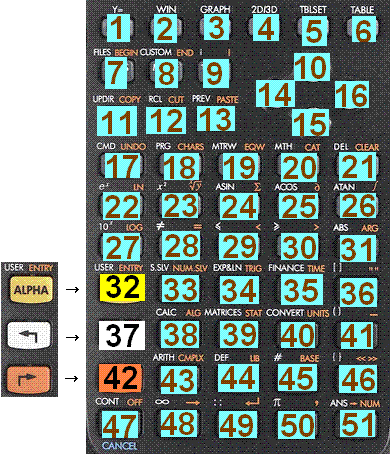
Ya viste antes como conseguir datos desde la pila. El uso de InputLine, POL y formularios de entrada serán vistos en los siguientes capítulos. En este capítulo aprenderemos como leer las pulsaciones de botones desde el teclado.

31.1 Localización de Teclas

En User RPL, las representaciones de las pulsaciones de teclas tienen la forma %fc.p.

En System RPL, estas están representadas por dos enteros binarios.

El primer bint, frecuentemente llamado #CodigoTecla, tiene valores desde 1 (tecla F1) hasta 51 (tecla ENTER), y representa cada tecla, en orden , de izquierda a derecha y de arriba a abajo, como se indica en el gráfico.



El segundo bint, llamado #Plano, representa el modificador de estado, de acuerdo a la tabla:

|  |  |  |
| --- | --- | --- |
| **#plano** | **Modificador** |  |
| 1 | Ninguno |  |
| 2 | Shift izquierdo | **TECLA CAMBIO IZQUIERDA AZUL** |
| 3 | Shift derecho | **TECLA CAMBIO DERECHA NARANJA** |
| 4 | Alpha | **TECLA ALPHA** |
| 5 | Alpha+ Shift izquierdo | **TECLA ALPHATECLA CAMBIO IZQUIERDA AZUL** |
| 6 | Alpha+ Shift derecho | **TECLA ALPHATECLA CAMBIO DERECHA NARANJA** |

Puedes convertir de una representación a otra usando los comandos:

Ck&DecKeyLoc ( %fc.p 🡺 #ct #Plano )

CodePl>%rc.p ( #ct #Plano 🡺 %fc.p )

Algunas veces, las teclas shift no son tratadas como modificadores por otras teclas, sino como teclas propias. En estos casos, las teclas shift tienen los códigos de tecla 40h (shift izquierdo), C0h (shift derecho), y 80h (alpha).

31.2 Planos Shift Hold

Existen cinco planos más además de los mostrados en la tabla anterior. Estos planos adicionales son los planos shift-hold. En estos casos, una tecla shift es presionada primero y se mantiene presionada (sin soltar) mientras otra tecla es presionada. En User RPL, estas teclas son denotadas agregando 0.01 a la representación %fc.p. Por ejemplo, el código de tecla 11.21 en User RPL significa mantener presionado shift izquierdo mientras se presiona la tecla F1.

En System RPL, las teclas shift-hold pueden ser codificadas de dos maneras.

La primera forma (la cual llamaremos codificación A) deja el código de tecla #ct sin cambios y usa nuevos planos #8,#9,#A,#B,#C.

La segunda forma (codificación B) usa planos en el rango #1...#6 y agrega el código de tecla de la tecla shift a #ct.

La siguiente tabla muestra las diferentes codificaciones para todas las posibles maneras de presionar la tecla F1 en la HP 50g.

**User RPL System RPL A System RPL B**

**Plano Shift Keys %fc.p #ct #p #ct #p**

1 Unshifted 11.1 1h 1h 1h 1h

2 Shift izquierdo 11.2 1h 2h 1h 2h

3 Shift derecho 11.3 1h 3h 1h 3h

4 Alpha 11.4 1h 4h 1h 4h

5 Alpha, shift izquierdo 11.5 1h 5h 1h 5h

6 Alpha, shift derecho 11.6 1h 6h 1h 6h

7 Sin uso

8 Shift-hold izquierdo 11.21 1h 8h 41h 2h

9 Shift-hold derecho 11.31 1h 9h C1h 3h

10 Alpha-hold 11.41 1h Ah 81h 4h

11 Alpha, shift-hold izquierdo 11.51 1h Bh 41h 5h

12 Alpha, shift-hold derecho 11.61 1h Ch C1h 6h

La mayoría de veces pero no siempre, los comandos de System RPL que tratan con teclas manejan las teclas shift-hold. La sección de referencia de abajo tiene información sobre este asunto para algunos comandos relevantes. Los comandos de System RPL que esperan #ct y #p como argumentos aceptan ambas formas de codificación (A y B). Los comandos que retornan #ct o #ct y #p, todas usan la codificación B.

Para convertir codificación B a codificación A, puedes usar lo siguiente:

NULLNAME CodifB>CodifA ( #ct\_B #plano\_B -> #ct\_A #plano\_A )

:: ( #ct\_B #plano\_B )

SWAP ( #plano\_B #ct\_B )

64 ( #plano\_B #ct\_B 64 )

#/ ( #plano\_B #r #q )

ROTSWAP ( #r #plano\_B #q )

#0=?SKIP

#6+

( #r #plano\_A )

;

31.3 Esperando una tecla

Un comando que podemos usar para esperar la pulsación de una tecla es WaitForKey. Este comando pone la calculadora en un modo de baja potencia y espera hasta que una tecla sea presionada. Luego retorna el código de tecla en el nivel 2 y el plano en el nivel 1.

Hay otros comandos, listados abajo, los cuales son usados en otras circunstancias.

Desafortunadamente, el comando WaitForKey no trata con las teclas shift-hold. Por lo tanto, mostramos a continuación unos programas cuyo comportamiento es similar a WaitForKey pero que pueden leer teclas shifthold y retornan la codificación A o la codificación B.

Para retornar la codificación A puedes usar el siguiente programa:

\* Devuelve código de tecla y plano de la tecla presionada a continuación.

\* Devuelve la codificación A

\* No funciona en los siguientes 8 casos:

\* shifthold izquierdo + ON

\* shifthold derecho + ON

\* alpha + shifthold izquierdo + ON

\* alpha + shifthold derecho + ON

\* shifthold izquierdo + alpha

\* shifthold derecho + alpha

\* alpha + shifthold izquierdo + alpha

\* alpha + shifthold derecho + alpha

\* En estos 8 casos el ShiftHold es leído como SHIFT

NULLNAME WaitForKey\_A ( -> #ct\_A #plano\_A )

::

POLSaveUI

ERRSET

:: ' ClrDAsOK ( ... ) ( AppDisplay )

' :: ( #ct\_B #p #ct #p #ct #p )

ModifierKey? ( #ct\_B #p #ct #p flag )

caseFALSE

( #ct\_B #p #ct #p )

' TakeOver ( #ct\_B #p #ct #p TakeOver )

3PICK3PICK ( #ct\_B #p #ct #p TakeOver #ct #p )

7PICK ( #ct\_B #p #ct #p TakeOver #ct #p #ct\_B )

BINT63 #> ( #ct\_B #p #ct #p TakeOver #ct #p flag )

IT #6+

( #ct\_B #p #ct #p TakeOver #ct #p\_A )

BINT3 ::N ( #ct\_B #p #ct #p prog )

TrueTrue ( #ct\_B #p #ct #p prog T T )

AppExitCond! ( #ct\_B #p #ct #p prog T )

; ( ... ) ( AppKeys )

TrueTrue ( ... ) ( NonAppKeyOK? y DoStdKeys? )

FALSE ( ... ) ( AppMenu )

BINT0 ( ... ) ( #AppMenuPage )

FalseFalse ( ... ) ( SuspendOK? y ExitCond )

'ERRJMP ( ... ) ( AppError )

POLSetUI

POLKeyUI

;

ERRTRAP

POLResUI&Err

POLRestoreUI

;

Para retornar la codificación B puedes usar el siguiente programa:

\* Devuelve código de tecla y plano de la tecla presionada a continuación.

\* Devuelve la codificación B

\* No funciona en los siguientes 8 casos:

\* shifthold izquierdo + ON

\* shifthold derecho + ON

\* alpha + shifthold izquierdo + ON

\* alpha + shifthold derecho + ON

\* shifthold izquierdo + alpha

\* shifthold derecho + alpha

\* alpha + shifthold izquierdo + alpha

\* alpha + shifthold derecho + alpha

\* En estos 8 casos el ShiftHold es leído como SHIFT

NULLNAME WaitForKey\_B ( -> #ct\_B #plano\_B )

::

POLSaveUI

ERRSET

:: ' ClrDAsOK ( ... ) ( AppDisplay )

' :: ( #ct\_B #p #ct #p #ct #p )

ModifierKey? ( #ct\_B #p #ct #p flag )

caseFALSE

( #ct\_B #p #ct #p )

' TakeOver ( #ct\_B #p #ct #p TakeOver )

5PICK ( #ct\_B #p #ct #p TakeOver #ct\_B )

3PICK ( #ct\_B #p #ct #p TakeOver #ct\_B #p )

BINT3 ::N ( #ct\_B #p #ct #p prog )

TrueTrue ( #ct\_B #p #ct #p prog T T )

AppExitCond! ( #ct\_B #p #ct #p prog T )

; ( ... ) ( AppKeys )

TrueTrue ( ... ) ( NonAppKeyOK? y DoStdKeys? )

FALSE ( ... ) ( AppMenu )

BINT0 ( ... ) ( #AppMenuPage )

FalseFalse ( ... ) ( SuspendOK? y ExitCond )

'ERRJMP ( ... ) ( AppError )

POLSetUI

POLKeyUI

;

ERRTRAP

POLResUI&Err

POLRestoreUI

;

31.4 Referencia

31.3.1 Conversión de Códigos de Tecla

**Direcc. Nombre Descripción**

25EA7 Ck&DecKeyLoc ( %fc.p 🡺 #ct\_B #p\_B )

Convierte la representación de una tecla de User a System B.

Maneja teclas shift-hold.

25EA9 CodePl>%rc.p ( #ct\_A #p\_A 🡺 %fc.p )

Convierte la representación de una tecla de System A a User.

Maneja teclas shift-hold.

25EDC H/W>KeyCode ( # 🡺 #' )

Convierte representación de teclas shift (retornada por algunos comandos) a sus códigos de tecla.

( 80h 🡺 32 )

( 40h 🡺 37 )

( C0h 🡺 42 )

25EEA ModifierKey? ( #ct #p 🡺 flag )

¿Representa la tecla alguno de los tres modificadores shift izquierdo, shift derecho o alpha?

( 37 #p 🡺 T )

( 42 #p 🡺 T )

( 32 #1 🡺 T )

( 32 #4 🡺 T )

31.3.2 Esperando Teclas

**Direcc. Nombre Descripción**

261CA FLUSHKEYS ( 🡺 )

Vacía el buffer del teclado.

aka: FLUSH

04708 CHECKKEY ( 🡺 #ct T )

( 🡺 F )

Retorna la siguiente tecla del buffer de teclado (si hay alguna), pero no la quita del buffer.

Maneja teclas shift-hold.

04714 GETTOUCH ( 🡺 #ct T )

( 🡺 F )

Quita la siguiente tecla del buffer de teclado (si hay alguna), y la retorna en la pila.

Maneja teclas shift-hold.

Es la version interna del comando KEY de User RPL.

25ED6 GETKEY ( 🡺 #ct T )

( 🡺 F )

Consigue el código de tecla desde el buffer de teclado, espera cuando el buffer está vacío. La tecla es retornada con TRUE.

Si la tecla presionada fue CANCEL, retorna FALSE.

Si una excepción ocurre, retorna FALSE. La excepción no es manejada (pero los objetos retornados por esa excepción son puestos en la pila cuando se han terminado de ejecutar los programas actuales).

Maneja teclas shift-hold.

**Direcc. Nombre Descripción**

25ED7 GETKEY\* ( 🡺 #ct T )

( 🡺 %alarma T F ) ( cuando es tiempo de una alarma )

( 🡺 F F )

( 🡺 ??? F F ) ( cuando existe la variable STARTOFF )

Consigue el código de tecla desde el buffer de teclado, espera cuando el buffer está vacío. La tecla es retornada con TRUE.

Si la tecla presionada fue CANCEL, retorna BINT47 TRUE.

Si una excepción ocurre (error, alarma u otra), la excepción es manejada y retorna FALSE.

Maneja teclas shift-hold.

25ED9 GetKeyOb ( 🡺 ob )

( 🡺 ??? ob ) (cuando existe la variable STARTOFF )

Espera por una tecla y retorna el objeto asociado con esa tecla.

25EDD H/WKey>KeyOb ( #ct 🡺 ob )

Retorna el objeto asociado con una tecla.

25EC5 DoKeyOb ( ob 🡺 )

Ejecuta ob como si este hubiera sido asignado a la tecla y la tecla ha sido presionada.

25EE3 KEYINBUFFER? ( 🡺 flag )

Espera por una tecla y retorna TRUE si hay alguna tecla en el buffer de teclado.

25F0B WaitForKey ( 🡺 #ct #plano )

Retorna el código de tecla y el plano.

No maneja teclas shift-hold.

2F268 Wait/GetKey ( %posit 🡺 ? )

( %noposit 🡺 %fc.p )

Si el argumento es positivo, espera el tiempo especificado en segundos con DOWAIT. Si se presiona la tecla ON, el programa es abortado y un error es generado.

Si el argumento no es positivo, espera por una tecla. Si la tecla presionada es ON o alpha+ON, el programa es abortado y un error es generado.

No maneja teclas shift-hold.

Equivale al comando WAIT de User RPL.

31.3.3 El flag ATTN

**Direcc. Nombre Descripción**

25FAE ATTN? ( 🡺 flag )

Retorna TRUE si CANCEL ha sido presionado.

25E70 ?ATTNQUIT ( 🡺 )

Si CANCEL ha sido presionado, ABORTA el programa.

25E9D CK0ATTNABORT ( 🡺 )

Ejecutado por los delimitadores de programas de User RPL: x<< y x>> y por xUNTIL. Principalmente hace ?ATTNQUIT.

25EED NoAttn?Semi ( 🡺 )

Si CANCEL no ha sido presionado, pasa por alto el resto del programa.

05040 ATTNFLG@ ( 🡺 # )

Llama al contador de la tecla CANCEL (número de veces que ha sido presionada la tecla CANCEL).

05068 ATTNFLGCLR ( 🡺 )

Limpia el contador de la tecla CANCEL. No afecta el buffer del teclado.

31.3.4 Bad Keys

**Direcc. Nombre Descripción**

25EBF DoBadKey ( 🡺 )

Beeps.

Hace :: TakeOver BINT70 # 151 setbeep SetDAsNoCh ;

25ECD DropBadKey ( ob 🡺 )

Beeps.

25E6E 2DropBadKey ( ob ob' 🡺 )

Beeps.

31.3.5 Teclado de Usuario

Si ninguna tecla está asignada, la lista interna de teclas de usuario es una lista vacía.

Si hay una o más asignaciones, la lista contiene 52 sublistas, las primeras 51 de estas representan una tecla. Cada una de estas listas puede ser una lista vacía (si la tecla no tiene asignaciones), o contener 12 elementos: la representación para cada plano. Los planos son dados en la tabla de la sección 31.1. Para planos sin asignaciones, el elemento respectivo es una lista vacía. La séptima lista está siempre vacía.

**Direcc. Nombre Descripción**

25F09 UserKeys? ( 🡺 flag )

¿Está activo el teclado del usuario?

Hace BINT62 TestSysFlag.

25967 GetUserKeys ( 🡺 {} )

Retorna las asignaciones de tecla del usuario (formato interno).

2F3B3 (StoUserKeypatch) ( ob #ct\_B #p\_B 🡺 )

Asigna un objeto a la tecla.

25617 (SetNUsrKeyOK) ( 🡺 )

Las teclas sin asignaciones, harán su acción por defecto.

Equivale a usar ‘S’ STOKEYS en User RPL.

2561C (ClrNUsrKeyOK) ( 🡺 )

Las teclas sin asignaciones, sólo harán BEEP al ser presionadas.

Equivale a usar ‘S’ DELKEYS en User RPL.

25EE5 Key>StdKeyOb ( #ct\_B #p\_B 🡺 ob )

Llama a la acción que realiza la tecla por defecto.

Esta es la acción que hace cuando está desactivado el teclado de usuario.

Maneja teclas shift hold.

25E76 ?Key>UKeyOb ( #ct\_B #p\_B 🡺 ob T )

( #ct\_B #p\_B 🡺 F )

Si el teclado de usario está activado, retorna la acción asociada a la tecla (asignación user o beep) si la hay y TRUE. Si no la hay, retorna FALSE.

Si el teclado de usario está desactivado, sólo retorna FALSE.

25EE6 Key>U/SKeyOb ( #ct\_B #p\_B 🡺 ob )

Si el teclado de usuario está activado y hay una asignación user para esa tecla, la retorna.

De otro modo, retorna la acción por defecto de la tecla.

255006 ˆKEYEVAL ( %fc.p 🡺 ? )

Evalúa el objeto asociado con la tecla. Si % es negativo, la asignación por defecto es siempre evaluada.

Equivale al comando KEYEVAL de User RPL.

31.5 Ejemplos Control de Teclado

**Ejemplo 1**

**Retornar objeto relacionado con la tecla presionada.**

Con el siguiente programa puedes retornar el objeto relacionado con cualquier tecla de la calculadora. Primero debes ejecutar el programa, luego debes presionar una tecla.

El resultado es el objeto relacionado a la tecla presionada.

Este programa también funciona para teclas shift y para teclas shift hold.

Por ejemplo, si presionas Shift hold izquierdo + TOOL, devuelve el programa:

:: TakeOver BINT103 DUP SysITE ClrSysFlag SetSysFlag ;

el cual cambia la calculadora entre los modos real y complejo.

Otro ejemplo: si presionas Shift hold izquierdo + F1, devuelve el programa:

:: PTR 275DA FPTR 3 9D ;

El cual permite abrir el formulario de entrada “TABLE SETUP”.

\* Retorna el objeto asociado a cualquier tecla de la calculadora.

\* ( 🡪 ob )

\* ( 🡪 … … … ) ( cuando una excepción ocurre: alarmas, errores u otros )

\* OBSERVACIONES:

\* Algunos códigos de tecla retornados por el comando GETKEY\* son:

\* Shift izquierdo: # 40

\* Shift izquierdo: # C0

\* ALPHA: # 80

:: CK0NOLASTWD ( ) ( ningún argumento es requerido )

AppMode? ( flag ) ( ¿Hay algún POL activo? )

?SEMI

BINT95 TestSysFlag

case

:: "Sólo funciona en modo RPN" FlashWarning ;

FALSE 1LAMBIND ( ) ( crea entorno temporal )

CLEARLCD ( ) ( limpia la pantalla )

"Presiona una tecla" ( $ )

DISPROW1 ( muestra en la línea 1 )

( )

BEGIN

:: GETKEY\* ( #ct T // ... F )

NOTcase

:: "Excepción ocurrida (error, alarma u otra)" FlashWarning

"Ejecuta el programa nuevamente" FlashWarning

TRUE

;

( #ct )

DUP BINT7 #< ( #ct flag )

case

:: GETPROC TRUE ;

( #ct )

DUP ( #ct #ct )

H/WKey>KeyOb ( #ct ob )

OVER ( #ct ob #ct )

# 40 #/ ( #ct ob #r #q )

DROP ( #ct ob #r )

#0<> ( #ct ob flag ) ( TRUE si es ALPHA, shift izq o der )

3PICK # 80 #= 1GETLAM AND ( #ct ob flag flag' )

ORcase

SWAPDROPTRUE

( #ct ob ) ( Es #40, #C0 ó # 80 )

EVAL ( #ct )

# 80 #<> ( flag ) ( TRUE si es shift izq o der )

IT

:: 1GETLAM NOT 1PUTLAM ;

FALSE

;

UNTIL

ABND

ClrDAsOK

;

Capítulo 32  
Usando InputLine

El comando InputLine hace algo similar al comando INPUT de User RPL.

• Muestra una cabecera en la parte de arriba de la pantalla.

• Inicia el modo de entrada desde el teclado.

• Inicializa la línea de edición.

• Acepta una entrada hasta que la tecla ENTER sea presionada.

• Convierte la cadena a objeto, evalua el contenido de la cadena, o sólo devuelve la cadena ingresada por el usuario.

• Retorna TRUE si el entorno fue terminado con ENTER o FALSE si fue abortado con ON/CANCEL.

La pila debe contener los siguientes 10 parámetros:

**Nombre Descripción**

$Prompt La cabecera que se mostrará en la parte superior.

$EditLine La línea de edición inicial.

CursorPos La posición inicial del cursor.

Puede ser un bint que indica la posición absoluta. El bint #0 indica que el cursor se movera al final del texto.

Puede ser una lista con dos bints que representan el número de fila y de columna. Si el primer bint es cero, el cursor estará en la última fila. Si el segundo bint es cero, el cursor estará al final de la fila.

#Ins/Rep El modo inicial del cursor.

• #0 modo actual

• #1 modo insertar

• #2 modo reemplazo

#Entry El modo de entrada inicial.

• #0 modo de entrada actual (según el modo algebraico o RPN).

• #1 modo de entrada algebraico desactivado

• #2 modo de entrada algebraico activado

#Alphalock El modo ALPHA inicial.

• #0 modo actual

• #1 modo ALPHA activado

• #2 modo ALPHA desactivado

ILMenu El menú inicial, en el formato especificado abajo.

También puede ser FALSE, lo cual indica que el menu actual no será cambiado.

#ILMenu El número de fila para el menú inicial mostrado.

#1 para la primera página, #7 para la segunda, #13 para la tercera…

AttnAbort? Un flag:

• TRUE CANCEL aborta inmediatamente la edición.

• FALSE CANCEL limpia la línea de edición y si se presiona nuevamente,aborta la edición.

#Parse Indica como se procesará la línea de edición.

• #0 retorna la línea de edición como una cadena (sin evaluar).

• #1 retorna la línea de edición como una cadena y también convierte esa

cadena a objeto.

• #2 evalúa el contenido de la cadena.

De acuerdo al valor del argumento#Parse, diferentes valores son retornados de acuerdo a la siguiente tabla:

**#Parse Pila Descripción**

#0 $LineaEdición TRUE Sólo la línea de edición (sin evaluar)

#1 $LineaEdición obs TRUE Línea de edición y objeto(s)

Cuando hay más de un objeto, estos son retornados en un único objeto programa. Este programa es diferente cuando la calculadora está en modo algebraico a cuando está en modo RPN.

#2 ob1 ... obn TRUE Objeto(s) resultante(s) en la pila.

FALSE Cuando se aborta con CANCEL.

32.1 Asignaciones de las Teclas de Menu

Puedes especificar un menú inicial por medio del parámetro ILMenu. Este menú será mostrado cuando se inicie el InputLine. Las teclas de menu pueden tener asignaciones para la opresión sin modificador, con shift izquierdo y con shift derecho. Cuando el InputLine termina, el menú previo es restaurado intacto.

El parámetro ILMenu es una lista (también puede ser un programa que retorne una lista), en el formato descrito en la sección 38.1. También puedes fijar este parámetro como FALSE, si no quieres que el menú actual sea cambiado al iniciarse el InputLine. Nota que cada acción debe iniciarse con el comando TakeOver para indicar que estas acciones serán ejecutadas con la línea de edición activa.

32.2 Los LAMS de InputLine

InputLine usa internamente 24 lams, de tal manera que si tu quieres llamar a otros lams que tu hayas creado anteriormente, es preferible que los hayas creado como lams con nombre. Los LAMs de InputLine son los siguientes:

**LAM Contenido**

1LAM Un flag. Si su valor es fijado como TRUE, el POL finalizará.

Nota: el POL también termina si se finaliza el editor. Por ejemplo usando el

comando DEL\_CMD.

2LAM Un bint. Es el parámetro #Parse

3LAM Un flag. Es el parámetro AttnAbort?

4LAM Una cadena. Es el parámetro $Prompt

32.3 Referencia

**Direcc. Nombre Descripción**

2EF5F InputLine ( args 🡺 $ T )

( args 🡺 $ ob1..obn T )

( args 🡺 ob1..obn T )

( args 🡺 F )

args= $cabec $line Cpos #Ins/Rep #Alg #alpha

menu #fila attn #parse

2F154 (Ck&Input1) ( $cabecera $inicial 🡺 $ )

Equivale al comando INPUT de User RPL cuando en la pila hay 2 cadenas.

2F155 (Ck&Input2) ( $cabecera {} 🡺 $ )

Equivale al comando INPUT de User RPL cuando en la pila hay una cadena y una lista.

2F300 DispILPrompt ( 🡺 )

Redibuja la cadena $Prompt en la parte superior de la pantalla (pero debajo del área de estado), cuando la línea de edición no es muy grande.

Si deseas usar este comando fuera del entorno InputLine, deberás colocar una cadena en 4LAM.

2F344 InputLAttn ( 🡺 )

( 🡺 F )

Equivale a presionar la tecla ON.

2F345 InputLEnter ( 🡺 )

( 🡺 $ T )

( 🡺 $ obs T )

( 🡺 ob1...obn T )

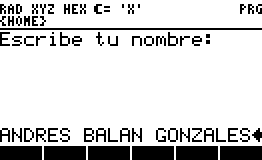
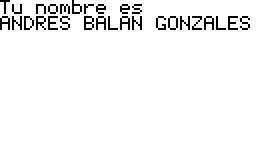
Equivale a presionar la tecla ENTER.

32.4 Ejemplos InputLine

**Ejemplo 1 InputLine**

**Pedir el ingreso de una cadena.**

En este ejemplo se usa el comando InputLine, para pedir el ingreso de una cadena, la cual será mostrada luego en la pantalla si el usuario no aborta el programa.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME EjempInputLine ( -> )

:: CK0

"Escribe tu nombre:" ( $ ) ( $cabec )

NULL$ ( $ $ ) ( $inicial )

BINT0 ( ... ) ( CursPos: cursor al final )

BINT1 ( ... ) ( #Ins/Rep: modo insertar )

BINT1 ( ... ) ( #ALG: modo algebraico desactivado )

BINT1 ( ... ) ( #alpha: modo alpha activado )

NULL{} ( ... ) ( no menu )

ONE ( ... ) ( fila del menu )

FALSE ( ... ) ( CANCEL limpia linea de edición )

BINT0 ( ... ) ( retorna sólo cadena sin evaluar )

InputLine

( $ T // F )

NOT?SEMI

( $ )

"Tu nombre es\0A" ( $ $' )

SWAP&$ ( $'' )

CLEARLCD ( $'' ) ( limpia la pantalla )

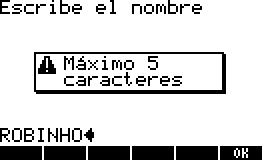
%1 DODISP ( ) ( muestra la cadena en la primera línea )

SetDAsTemp ( ) ( congela la pantalla )

;

**Ejemplo 2 InputLine**

**Pedir el ingreso de una cadena con un número máximo de caracteres.**

En este ejemplo se usa el comando InputLine, para pedir el ingreso de una cadena.

Si la cadena es vacía, se muestra el mensaje “Escribe una cadena” y se continua en la línea de edición.

Si la cadena tiene más de 5 caracteres, se muestra el mensaje “Máximo 5 caracteres” y se continúa en la línea de edición.

De esta forma, siempre se retorna una cadena que tenga entre 1 y 5 caracteres y TRUE o sólo FALSE.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME Input$5 ( -> $ T // F )

:: CK0

"Escribe el nombre"

NULL$

\* PROGRAMA DE LA FORMA ( ob -> flag )

' :: DUPNULL$? ( $ flag )

casedrop

:: "Escribe una cadena" FlashWarning FALSE ;

LEN$ ( #n )

BINT5

#> ( flag )

case

:: "Máximo 5 caracteres" FlashWarning FALSE ;

TRUE ( T )

;

GetHeader ( $cabec $inicial prog #header )

BINT0 SetHeader ( )

' NULLLAM BINT4 NDUPN DOBIND

( )

BEGIN

:: 4GETLAM ( $ ) ( $cabec )

3GETLAM ( $ $ ) ( $EditLine )

BINT0 ( ... ) ( CursPos: cursor al final )

BINT1 ( ... ) ( #Ins/Rep: modo insertar )

BINT2 ( ... ) ( #ALG: modo algebraico activado )

BINT1 ( ... ) ( #alpha: modo alpha activado )

NULL{} ( ... ) ( no menu )

ONE ( ... ) ( fila del menu )

FALSE ( ... ) ( CANCEL limpia linea de edición )

BINT0 ( ... ) ( retorna sólo cadena sin evaluar )

InputLine ( $ T // F )

NOTcase FalseTrue

( $ )

DUP ( $ $ )

2GETEVAL ( $ flag )

case

TrueTrue

( $ )

3PUTLAM ( )

FALSE ( F )

;

UNTIL

( $ T // F )

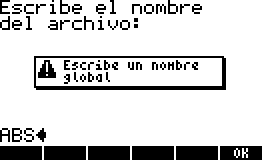
1GETABND ( $ T #header // F #header )

SetHeader ( $ T // F )

;

**Ejemplo 3 InputLine**

**Pedir el ingreso de un nombre global.**

El siguiente programa pide escribir un nombre global.

Si uno escribe algo incorrecto se muestra el mensaje “sintaxis incorrecta” y se continúa en la línea de edición.

Si uno escribe algo que que no es un nombre global, aparece el mensaje “Escribe un nombre global” y se continúa en la línea de edición.

De esta forma, siempre se retorna un nombre global y TRUE o sólo FALSE.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME InputNombGlobal ( -> id T // F )

:: CK0

"Escribe el nombre\0Adel archivo:"

NULL$

\* PROGRAMA DE LA FORMA ( ob -> flag )

\* id -> T

\* ob -> F

' :: ( ob )

TYPEIDNT? ( flag )

NOTcase

:: "Escribe un nombre global" FlashWarning FALSE ;

( )

TRUE ( T )

;

GetHeader ( $cabec $inicial prog #header )

' NULLLAM BINT4 NDUPN DOBIND

( )

BINT0 SetHeader ( )

BEGIN

:: 4GETLAM ( $ ) ( $cabec )

3GETLAM ( $ $ ) ( $inicial )

BINT0 ( ... ) ( CursPos: cursor al final )

BINT1 ( ... ) ( #Ins/Rep: modo insertar )

BINT2 ( ... ) ( #ALG: modo algebraico activado )

BINT1 ( ... ) ( #alpha: modo alpha activado )

NULL{} ( ... ) ( no menu )

BINT1 ( ... ) ( fila del menu )

FALSE ( ... ) ( CANCEL limpia linea de edición )

BINT1 ( ... ) ( retorna $ y objeto convertido )

RunSafeFlags

:: BINT72 ClrSysFlag BINT73 ClrSysFlag BINT95 ClrSysFlag

InputLine

;

( $ ob T // F )

NOTcase FalseTrue

( $ ob )

DUP ( $ ob ob )

2GETEVAL ( $ ob flag ) ( $ id/ob T/F )

case

:: SWAPDROP ( id )

TrueTrue ( id T T )

;

( $ ob )

DROP ( $ )

3PUTLAM ( ) ( guarda línea de edición inicial )

FALSE ( F )

;

UNTIL

( id T // F )

1GETABND ( id T #header // F #header )

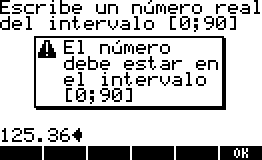
SetHeader ( id T // F )

;

**Ejemplo 4 InputLine**

**Pedir el ingreso de un número real de un intervalo deseado.**

El siguiente programa pide escribir un número real.

Si uno escribe algo incorrecto se muestra el mensaje “sintaxis incorrecta” y se continúa en la línea de edición.

Si uno escribe algo que que no es un número real, aparece el mensaje “Escribe un número real” y se continúa en la línea de edición.

Si uno escribe un número real que no está en el intervalo [0;90], aparece el mensaje “El número debe estar en el intervalo [0;90]” y se continúa en la línea de edición.

De esta forma, siempre se retorna un número del intervalo deseado y TRUE o sólo FALSE.

xNAME InputRealInterv ( -> % T // F )

:: CK0

"Escribe un número real\0Adel intervalo [0;90]"

NULL$

\* PROGRAMA DE LA FORMA ( ob -> flag )

' :: DUPTYPEZINT? ( ob flag )

IT FLASHPTR Z>R

( ob )

DUPTYPEREAL? ( ob flag )

NOTcasedrop

:: "Escribe un número real" FlashWarning FALSE ;

( % )

DUP %0>= SWAP 90. %<= AND ( flag )

NOTcase

:: "El número debe estar en el intervalo [0;90]" FlashWarning FALSE ;

TRUE ( T )

;

GetHeader ( $cabec $inicial prog #header )

' NULLLAM BINT4 NDUPN DOBIND ( )

BINT0 SetHeader ( )

BEGIN

:: 4GETLAM ( $ ) ( $cabec )

3GETLAM ( $ $ ) ( $inicial )

BINT0 ( ... ) ( CursPos: cursor al final )

BINT1 ( ... ) ( #Ins/Rep: modo insertar )

BINT2 ( ... ) ( #ALG: modo algebraico activado )

BINT2 ( ... ) ( #alpha: modo alpha desactivado )

NULL{} ( ... ) ( no menu )

BINT1 ( ... ) ( fila del menu )

FALSE ( ... ) ( CANCEL limpia linea de edición )

BINT1 ( ... ) ( retorna $ y objeto convertido )

RunSafeFlags

:: BINT72 ClrSysFlag BINT73 ClrSysFlag BINT95 ClrSysFlag

InputLine

; ( $ ob T // F )

NOTcase FalseTrue

( $ ob )

DUP ( $ ob ob )

2GETEVAL ( $ ob flag )

case

:: SWAPDROP ( %/Z )

CKREAL ( % )

TrueTrue ( % T T )

;

( $ ob )

DROP ( $ )

3PUTLAM ( ) ( guarda línea de edición inicial )

FALSE ( F )

;

UNTIL

( % T // F )

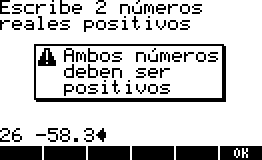
1GETABND ( % T #header // F #header )

SetHeader ( % T // F )

;

**Ejemplo 5 InputLine**

**Pedir el ingreso de dos números reales positivos.**

El siguiente programa pide escribir 2 números reales positivos.

Si uno escribe algo incorrecto se muestra el mensaje “sintaxis incorrecta” y se continúa en la línea de edición.

Si uno escribe algo que que no sean dos números reales, aparece el mensaje “Escribe dos números reales” y se continúa en la línea de edición.

Si uno escribe dos números reales que no sean ambos positivos, aparece el mensaje “Ambos números deben ser positivos” y se continúa en la línea de edición.

De esta forma, siempre se retornan 2 números positivos y TRUE o sólo FALSE.

\* Necesita el subprograma PROG\_Input2%Posit

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME Input2%Posit ( -> % %' T // F )

:: CK0

"Escribe 2 números\0Areales positivos"

NULL$

' PROG\_Input2%Posit

GetHeader ( $cabec $inicial prog #header )

' NULLLAM BINT4 NDUPN DOBIND

( )

BINT0 SetHeader ( )

BEGIN

:: 4GETLAM ( $ ) ( $cabec )

3GETLAM ( $ $ ) ( $inicial )

BINT0 ( ... ) ( CursPos: cursor al final )

BINT1 ( ... ) ( #Ins/Rep: modo insertar )

BINT2 ( ... ) ( #ALG: modo algebraico activado )

BINT2 ( ... ) ( #alpha: modo alpha desactivado )

NULL{} ( ... ) ( no menu )

BINT1 ( ... ) ( fila del menu )

FALSE ( ... ) ( CANCEL limpia linea de edición )

BINT1 ( ... ) ( retorna $ y objeto convertido )

RunSafeFlags

:: BINT72 ClrSysFlag BINT73 ClrSysFlag BINT95 ClrSysFlag

InputLine

;

( $ ob T // F )

NOTcase FalseTrue

( $ ob )

DUP ( $ ob ob )

2GETEVAL ( $ ob flag )

case

:: SWAPDROP ( prog )

INCOMPDROP ( ob1 ob2 )

SWAP CKREAL ( ob2 %1 )

SWAP CKREAL ( %1 %2 )

TrueTrue ( %1 %2 T T )

;

( $ ob )

DROP ( $ )

3PUTLAM ( ) ( guarda línea de edición inicial )

FALSE ( F )

;

UNTIL

( %1 %2 T // F )

1GETABND ( %1 %2 T #header // F #header )

SetHeader ( %1 %2 T // F )

;

\* PROGRAMA DE LA FORMA ( ob -> flag )

NULLNAME PROG\_Input2%Posit

:: ( ob )

:: DUPTYPECOL? ( ob flag )

NOTcase DROPFALSE

( prog )

DUPLENCOMP ( prog #n )

#2= ( prog flag )

NOTcase DROPFALSE

( prog )

INCOMPDROP ( ob1 ob2 )

DUPTYPEZINT? ( ob1 ob2 flag )

IT FLASHPTR Z>R

( ob1 ob2 )

SWAP ( ob2 ob1 )

DUPTYPEZINT? ( ob2 ob1 flag )

IT FLASHPTR Z>R

( ob2 ob1 )

DUPTYPEREAL? ( ob2 ob1 flag )

3PICK TYPEREAL? ( ob2 ob1 flag flag' )

ANDNOTcase

2DROPFALSE

( %1 %2 )

TRUE ( %1 %2 T // F )

;

NOTcase

:: "Escribe dos números reales" FlashWarning FALSE ;

( %1 %2 )

%0> SWAP %0> AND ( flag )

NOTcase

:: "Ambos números deben ser positivos" FlashWarning FALSE ;

TRUE ( T )

;

Capítulo 33  
El Bucle Externo Parametrizado (POL)

El Bucle Externo Parametrizado es una estructura System RPL que permite crear una aplicación completa, la cual lee las pulsaciones de teclas y hace diferentes acciones de acuerdo a la tecla que fue presionada. Esto es repetido tantas veces como sea necesario, hasta que ocurre una condición de salida. La mayoría de las veces, hay una tecla que detiene al bucle, como CANCL, OK o DROP. Generalmente, el POL es usado para hacer programas que manipulan la pantalla. Complejos usos del POL incluyen formularios de entrada (Capítulos 37 y 38) y los browsers (capítulos 34, 35 y 36). Nota que el POL es una construcción muy general y por esta razón requiere argumentos elaborados.

Aplicaciones más simples pueden hacerse más fácil y compactamente usando un bucle alrededor del comando WaitForKey (sección 31.3) en lugar de usar un POL.

Para usar un POL, nueve parámetros son necesarios:

**Parámetro Descripción**

AppDisplay Este objeto es evaluado antes de cada opresión de tecla.

Este objeto debería manejar aspectos de la actualización de la pantalla no manejados por las acciones de las teclas y también debería realizar manejos especiales de errores.

AppKeys Contiene las asignaciones de las teclas en el formato descrito abajo.

NonAppKeyOK? Un flag.

TRUE: las teclas no asignadas realizan sus acciones normales.

FALSE: las teclas no asignadas sólo hacen beep.

DoStdKeys? Un flag. Sólo tiene sentido cuando NonAppKeyOK? es TRUE.

TRUE: las teclas no asignadas realizan su acción por defecto.

FALSE: las teclas no asignadas realizan la definición del teclado de usuario en caso de existir esta, o la acción por defecto si no existe esta.

AppMenu Contiene al menu, en cualquiera de las especificaciones descritas en la sección 40.1, o FALSE para dejar al menú actual sin cambios.

#AppMenuPage La página inicial del menu.

Normalmente es BINT1 para mostrar la primera página.

#1 para la primera página, #7 para la segunda, #13 para la tercera…

SuspendOK? Un flag.

Si es FALSE, no se permitirá la ejecución de cualquier comando que cree un entorno suspendido y en su lugar se generará un error.

ExitCond Este objeto es evaluado antes de cada actualización de pantalla y lectura de tecla. Si su resultado es TRUE, el bucle es finalizado.

AppError Este objeto es un manejador de errores y es evaluado cuando un error ocurre durante la evaluación de una tecla.

Después de poner los 9 argumentos en la pila, puedes llamar al comando ParOuterLoop. Este comando no genera ningún resultado por si mismo, pero puedes retornar resultados en la pila manipulando alguno de los argumentos.

Principalmente, puedes manipular los argumentos AppKeys o ExitCond para retornar resultados en la pila o mostrar resultados de cualquier otra forma.

33.1 Comandos del POL

El POL está formado por llamadas (con apropiado manejo de errores) a los comandos de la siguiente tabla.

De estos comandos, el único que pide argumentos es POLSetUI, los mismos requeridos por el comando ParOuterLoop.

De estos comandos, el único que puede retornar resultados en la pila es POLKeyUI, estos resultados son especificados en algunos de los argumentos del POL, principalmente el argumento AppKeys.

**Comando Acción**

POLSaveUI Guarda la actual interfaz de usuario (la anterior al POL que vamos a programar) en la pila virtual.

POLSetUI Fija la nueva interfaz de usuario (la interfaz del POL que deseamos programar). Pide 9 argumentos.

POLKeyUI Este comando es el centro del POL. Es un bucle que actualiza la pantalla, espera una tecla y evalúa la asignación de dicha tecla. Si un error ocurre durante la evaluación de esa asignación, evalúa al argumento AppError. Estas acciones son realizadas hasta que la evaluación del argumento ExitCond retorne FALSE.

POLRestoreUI Restaura la interfaz de usuario guardada por POLSaveUI.

POLResUI&Err Restaura la interfaz de usuario guardada por POLSaveUI y llama al último error. Este comando es usado cuando hay un error no manejado con AppError (es decir, un error que no ocurre durante la evaluación de una tecla).

La descompilación del comando ParOuterLoop es:

::

POLSaveUI ( Guarda la interfaz actual en la pila virtual )

ERRSET

::

POLSetUI ( Fija la nueva interfaz, necesita 9 argumentos )

POLKeyUI ( bucle que evalúa condición de salida, actualiza la )

( pantalla, espera una tecla y realiza su acción )

( correspondiente. Si un error ocurre en esta )

( última acción, llama a AppError y lo evalúa )

;

ERRTRAP

POLResUI&Err ( si un error ocurre, restaura la interfaz guardada )

( por POLSaveUI y luego llama al error que ocurrió )

POLRestoreUI ( restaura la interfaz guardada por POLSaveUI )

;

Si usas los comandos mostrados en la tabla en lugar del comando ParOuterLoop, debes proporcionar el mismo nivel de protección que el mostrado en el código de arriba.

Algunas aplicaciones basadas en el POL, tales como formularios de entrada o browsers, crean un entorno temporal después de usar el comando POLSaveUI y lo abandonan antes de restaurar la interfaz de usuario guardada. Esto significa que no podrás usar comandos que operen con entornos temporales creados anteriormente, tales como 1GETLAM dentro del bucle. Para usar los entornos temporales creados anteriormente al llamado a POLSaveUI, deberás crear esos entornos temporales usando variables “con nombre”.

33.2 La Pantalla

En el POL, el programador es responsible de mostrar la pantalla y actualizarla. No hay una pantalla por defecto. La pantalla puede ser actualizada de dos maneras: con el parámetro AppDisplay o con asignaciones de teclas. Por ejemplo, cuando el usuario presiona una tecla para mover el cursor, la asignación de tecla puede pasar alguna información a AppDisplay, para que esta manipule la actualización de pantalla; o también la asignación de tecla puede manejar por si misma la pantalla. Cualquiera de estos dos métodos es más eficiente que el otro de acuerdo a la situación. En el ejemplo 1 de abajo, AppKeys solo fija la posición del grob en lams, y AppDisplay dibuja el grob.

33.3 Asignaciones de Teclas

En el POL, a cualquier tecla de los seis planos básicos (ver sección 31.1) se le puede asignar una nueva función. El parámetro AppKeys especifica las teclas que serán asignadas y sus respectivas acciones.

Si una tecla no tiene una asignación en el POL (dada por AppKeys), la acción a realizarse cuando se presiona esta tecla depende de los parámetros NonAppKeyOK? y DoStdKeys?.

Si el parámetro NonAppKeyOK? es TRUE, entonces:

* Si DoStdKeys? es TRUE, la definición por defecto de la tecla es ejecutada.
* Si DoStdKeys? es FALSE, se realiza la definión del teclado de usuario y si no hay esta definición, entonces se realiza la acción por defecto de la tecla.

Si el parámetro NonAppKeyOK? es FALSE, sólo se produce un beep y nada más.

La mayoría de las veces, NonAppKeysOK? se fija como FALSE.

El parámetro AppKeys es un programa, el cual toma como argumento el código de tecla y el plano, y retorna la asignación (generalmente un programa, comado o rompointer) de la tecla deseada y TRUE, o FALSE si la aplicación no maneja esta tecla. Por lo tanto, el diagrama de pila es como sigue:

( #CodigoTecla #Plano 🡺 AsignacionTecla TRUE )

( #CodigoTecla #Plano 🡺 FALSE )

Una forma sugerida para el parámetro AppKeys es:

BINT1 #=casedrop

:: (process unshifted plane) ;

BINT2 #=casedrop

:: (process left-shifted plane) ;

...

2DROPFALSE

Y cada manejador de plano generalmente tiene la forma:

BINT7 ?CaseKeyDef :: TakeOver <proceso tecla APPS> ;

BINT9 ?CaseKeyDef :: TakeOver <proceso tecla TOOL> ;

...

DROPFALSE

El comando ?CaseKeyDef es muy práctico en este caso, porque es equivalente a

#=casedrop :: ' <keydef> TRUE ;

El diagrama de pila de este comando es:

( # #' 🡺 :: ' ob1 T ; )

( # #' 🡺 :: # <ob2> <rest> ; )

?CaseKeyDef compara dos bints. Si son iguales, los quita de la pila y coloca el siguiente objeto del programa seguido por TRUE y pasa por alto el resto del programa.

Si son diferentes, quita el bint del nivel 1 de la pila y pasa por alto el siguiente objeto del programa.

Si quieres manejar teclas shift hold, si lo puedes hacer. La codificación B de la tecla (ver sección 31.1) es provista al programa AppKeys en la pila en los niveles 5 y 6. Todo lo que necesitas hacer es empezar AppKeys con el código

4DROP 2DUP 2DUP

Y luego operar normalmente.

33.4 Asignaciones de Teclas de Menú

Puedes especificar un menu para ser mostrado cuando el POL se inicia. El formato del parámetro AppMenu es esencialmente el mismo que el del parámetro ILMenu del comando InputLine, descrito en la sección 38.1.

La diferencia es que TakeOver no es necesario en este caso, puesto que la línea de entrada no está activa.

También, debido a que las asignaciones de teclas de AppKeys tienen prioridad sobre las asignaciones de teclas de menú de AppMenu, deberás poner este código en el parámetro AppKeys, en cada definición de plano:

DUP#<7 casedrpfls

como se indica en el ejemplo 1 de este capítulo. Esto pondrá FALSE cuando una tecla cuyo código sea menor que 7 (que es una tecla de menú), ha sido presionada. El FALSE forzará que la asignación por defecto de dicha tecla sea ejecutada y esta acción por defecto es la acción correspondiente en el menú (definida con el parámetro AppMenu).

Para que esto pueda funcionar, el parámetro NonAppKeysOK? debe ser TRUE, De este modo, las teclas de menú cumplirán su función, esto es, harán la acción especificada por el parámetro AppMenu.

33.5 Previniendo Entornos Suspendidos

Tu aplicación podría requerir la evaluación de comandos arbitrarios y argumentos ingresados por el usuario, pero puede no ser deseable que el entorno actual sea suspendido por comandos como HALT o PROMPT. El parámetro SuspendOK?, cuando es FALSE, hará que este y otros commandos capaces de suspender el entorno, no se ejecuten normalmente y en su lugar, se generará el error “HALT no permitido”, el cual puede ser manejado por el parámetro AppError.

Si el parámetro SuspendOK? es TRUE, la aplicación debe estar preparada para manejar las consecuencias. “Los peligros aquí son muchos y muy severos” como está escrito en RPLMAN.DOC.

Casi todas las aplicaciones deben de fijar al parámetro SuspendOK? como FALSE.

33.6 La Condición de Salida

El parámetro ExitCond es un objeto que es evaluado antes de cada evaluación de tecla. Si al evaluar este objeto, el resultado es TRUE, el bucle es finalizado. Si es FALSE, el bucle continua.

Es muy común que el parámetro ExitCond sea una lam, por ejemplo ' LAM exit. Puedes asignar una tecla de salida, de manera que al presionarse se ejecute el código

TRUE ' LAM exit STO

y el bucle será finalizado. De hecho, deberás crear antes el lam y fijar su valor inicial como FALSE.

33.7 Manejador de Errores

Si un error ocurre durante la ejecución de la acción asignada a la tecla presionada, entonces el parámetro AppError es ejecutado. AppError debería determinar que error ha ocurrido y actuar de acuerdo a este.

También puedes fijar al parámetro AppError, simplemente como ERRJMP, lo cual significa que tu aplicación no manejará ningún error.

33.8 Referencia

33.8.1 POL

**Direcc. Nombre Descripción**

2B475 ParOuterLoop ( Disp Keys NonAppKeys? DoStdKeys?

menu #row suspendOK? ExitCond AppErr 🡺 )

2B4AC POLSaveUI ( 🡺 )

Guarda la interfaz actual en la pila virtual.

2B542 POLSetUI ( Disp Keys NonAppKeys? DoStdKeys?

menu #row suspendOK? ExitCond AppErr 🡺 )

Establece un Nuevo POL, require los mismos argumentos que el comando ParOuterLoop.

2B628 POLKeyUI ( 🡺 ??? )

Actualiza la pantalla, lee una tecla, evalúa su acción hasta que la condición de salida sea TRUE.

2B6CD POLRestoreUI ( 🡺 )

Restaura la interfaz guardada en la pila virtual por POLSaveUI

2B6B4 POLResUI&Err ( 🡺 )

Restaura la interfaz guardada en la pila virtual por POLSaveUI y ejecuta ERRJMP.

25690 AppMode? ( 🡺 flag )

Retorna TRUE si hay actualmente un POL activo.

25695 SetAppMode ( 🡺 )

2569A ClrAppMode ( 🡺 )

27E72 nohalt ( 🡺 ob )

:: LAM 'nohalt ;

33.8.2 Argumentos del POL

**Direcc. Nombre Descripción**

29F25 AppDisplay! ( ob 🡺 )

29F35 AppDisplay@ ( 🡺 ob )

29F55 AppKeys! ( ob 🡺 )

29F65 PTR 29F65 ( 🡺 ob )

Debería llamarse AppKeys@

2564D SetNAppKeyOK ( 🡺 )

25652 (ClrNAppKeyOK) ( 🡺 )

25648 PTR 25648 ( 🡺 flag )

Debería llamarse NAppKeyOK?

2565F SetDoStdKeys ( 🡺 )

25664 (ClrDoStdKeys) ( 🡺 )

2565A DoStdKeys? ( 🡺 flag )

25671 SetAppSuspOK ( 🡺 )

25676 ClrAppSuspOK ( 🡺 )

2566C (AppSuspOK?) ( 🡺 flag )

25F04 SuspendOK? ( 🡺 flag )

Retorna TRUE si el parámetro SuspendOK? es TRUE y el lam 'nohalt no existe.

2A055 AppExitCond! ( ob 🡺 )

2A065 AppExitCond@ ( 🡺 ob )

2A145 AppError! ( ob 🡺 )

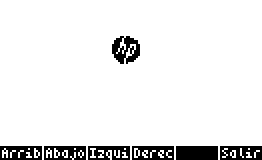
2A158 AppError@ ( 🡺 ob )

33.9 Ejemplos

**Ejemplo 1 POL**

**Mover un gráfico a través de la pantalla.**

El siguiente programa es un ejemplo de una aplicación que usa un POL para crear un entorno en donde el usuario puede mover un pequeño gráfico sobre la pantalla. Puedes usar las teclas de dirección para mover el gráfico un píxel, o puedes hacerlo también con las teclas del menú. En ambos casos puedes presionar antes el shift izquierdo para mover la figura 10 píxeles, o el shift derecho para mover la figura a los extremos de la pantalla. El código se asegura de que la figura no se salga de la pantalla.



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME PolEjemplo1

::

CK0

\* Prepara la pantalla

RECLAIMDISP ( fija a ABUFF como activa, lo limpia y lo redimensiona )

ClrDA1IsStat ( suspende la presentación del reloj temporalmente )

\* Grob con el símbolo de HP. Su tamaño es de 14x14

GROB 00042 E0000E00000B10CD70EDF1ECF1F813FA537B637DA3B5A3E7C3E7F1CBF08B700B00

HARDBUFF ( grob14x14 GROB131x80 )

TOTEMPOB ( grob14x14 GROB131x80 )

BINT56 BINT18 ( grob GROB #x #y ) ( coord inic x,y para mostrar grob )

FALSE ( grob GROB #x #y F ) ( condición inicial de salida del POL )

{ LAM grobHP LAM GROB LAM x LAM y LAM exit? } BIND

\* El siguiente compuesto es el argumento AppDisplay del POL

\* Dibuja el símbolo de hp en la posición indicada con LAM x y LAM y

' :: LAM GROB ( GROB )

TOTEMPOB ( GROB )

LAM grobHP ( GROB grob )

OVER ( GROB grob GROB )

LAM x LAM y ( GROB grob GROB #x #y )

GROB! ( GROB )

HARDBUFF ( GROB ABUFF )

ZEROZERO ( GROB ABUFF #0 #0 )

GROB! ( ) ( dibuja en la pantalla )

DispMenu.1 ( ) ( muestra el menú )

; ( prg ) ( AppDisplay )

' AppKeys\_EJEMPLO1 ( ... ) ( AppKeys )

TrueTrue ( ... ) ( NonAppKeyOK? y DoStdKeys? )

MENU\_EJEMPLO1 ( ... ) ( menú )

ONEFALSE ( ... ) ( #AppMenuPage y SuspendOK? )

' LAM exit? ( ... ) ( ExitCond )

'ERRJMP ( ... ) ( AppError )

ParOuterLoop ( ) ( ejecuta el POL )

ClrDAsOK ( ) ( redibuja la pantalla )

;

\* Este es el argumento AppKeys del POL

NULLNAME AppKeys\_EJEMPLO1 ( #ct #p -> ob T // F )

:: ( #ct #p )

BINT1 #=casedrop

:: ( #ct ) ( PLANO 1 )

BINT10 ?CaseKeyDef

ACCION\_ARRIBA

BINT15 ?CaseKeyDef

ACCION\_ABAJO

BINT14 ?CaseKeyDef

ACCION\_IZQUIERDA

BINT16 ?CaseKeyDef

ACCION\_DERECHA

BINT47 ?CaseKeyDef

:: TRUE ' LAM exit? STO ;

DUP#<7 casedrpfls

BINT37 #=casedrpfls

BINT42 #=casedrpfls

DROP 'DoBadKeyT

;

( #ct #p )

BINT2 #=casedrop

:: ( #ct ) ( PLANO 2 )

BINT10 ?CaseKeyDef

ACCION\_ShiftIzq\_ARRIBA

BINT15 ?CaseKeyDef

ACCION\_ShiftIzq\_ABAJO

BINT14 ?CaseKeyDef

ACCION\_ShiftIzq\_IZQUIERDA

BINT16 ?CaseKeyDef

ACCION\_ShiftIzq\_DERECHA

DUP#<7 casedrpfls

BINT37 #=casedrpfls

BINT42 #=casedrpfls

DROP 'DoBadKeyT

;

( #ct #p )

BINT3 #=casedrop

:: ( #ct ) ( PLANO 3 )

BINT10 ?CaseKeyDef

ACCION\_ShiftDer\_ARRIBA

BINT15 ?CaseKeyDef

ACCION\_ShiftDer\_ABAJO

BINT14 ?CaseKeyDef

ACCION\_ShiftDer\_IZQUIERDA

BINT16 ?CaseKeyDef

ACCION\_ShiftDer\_DERECHA

DUP#<7 casedrpfls

BINT37 #=casedrpfls

BINT42 #=casedrpfls

DROP 'DoBadKeyT

;

( #ct #p ) ( PLANOS 4,5,6 )

2DROP ( )

'DoBadKeyT ( 'DoBadKey T )

;

\* Este es el menú del POL

NULLNAME MENU\_EJEMPLO1

{

{ "Arrib" { ACCION\_ARRIBA ACCION\_ShiftIzq\_ARRIBA ACCION\_ShiftDer\_ARRIBA } }

{ "Abajo" { ACCION\_ABAJO ACCION\_ShiftIzq\_ABAJO ACCION\_ShiftDer\_ABAJO } }

{ "Izqui" { ACCION\_IZQUIERDA ACCION\_ShiftIzq\_IZQUIERDA ACCION\_ShiftDer\_IZQUIERDA } }

{ "Derec" { ACCION\_DERECHA ACCION\_ShiftIzq\_DERECHA ACCION\_ShiftDer\_DERECHA } }

NullMenuKey

{ "Salir" :: TRUE ' LAM exit? STO ; }

}

NULLNAME ACCION\_ARRIBA

:: LAM y DUP BINT1 #<ITE :: DROP ERRBEEP ; :: #1- ' LAM y STO ; ;

NULLNAME ACCION\_ShiftIzq\_ARRIBA

:: LAM y DUP BINT10 #<ITE :: DROPZERO ERRBEEP ; :: BINT10 #- ; ' LAM y STO ;

NULLNAME ACCION\_ShiftDer\_ARRIBA

:: ERRBEEP BINT0 ' LAM y STO ;

NULLNAME ACCION\_ABAJO

:: LAM y DUP BINT57 #>ITE :: DROP ERRBEEP ; :: #1+ ' LAM y STO ; ;

NULLNAME ACCION\_ShiftIzq\_ABAJO

:: LAM y DUP BINT48 #>ITE :: DROP BINT58 ERRBEEP ; #10+ ' LAM y STO ;

NULLNAME ACCION\_ShiftDer\_ABAJO

:: ERRBEEP BINT58 ' LAM y STO ;

NULLNAME ACCION\_IZQUIERDA

:: LAM x DUP BINT1 #<ITE :: DROP ERRBEEP ; :: #1- ' LAM x STO ; ;

NULLNAME ACCION\_ShiftIzq\_IZQUIERDA

:: LAM x DUP BINT10 #<ITE :: DROPZERO ERRBEEP ; :: BINT10 #- ; ' LAM x STO ;

NULLNAME ACCION\_ShiftDer\_IZQUIERDA

:: ERRBEEP BINT0 ' LAM x STO ;

NULLNAME ACCION\_DERECHA

:: LAM x DUP BINT116 #>ITE :: DROP ERRBEEP ; :: #1+ ' LAM x STO ; ;

NULLNAME ACCION\_ShiftIzq\_DERECHA

:: LAM x DUP BINT107 #>ITE :: DROP BINT117 ERRBEEP ; #10+ ' LAM x STO ;

NULLNAME ACCION\_ShiftDer\_DERECHA

:: ERRBEEP BINT117 ' LAM x STO ;

Si en el POL del ejemplo 1 cambias el AppKeys por el mostrado en esta página, podrás mover el grob por la pantalla varias veces si mantienes presionadas (sin soltar) las teclas de dirección.

\* Este es el argumento AppKeys del POL

\* Si usas este AppKeys podrás mover el gráfico por la pantalla

\* varias veces, si presionas una tecla de dirección sin soltarla.

NULLNAME AppKeys\_EJEMPLO1 ( #ct #p -> ob T // F )

:: ( #ct #p )

BINT1 #=casedrop

:: ( #ct ) ( PLANO 1 )

BINT10 ?CaseKeyDef

:: REPEATER 10 :: ACCION\_ARRIBA AppDisplay@ EVAL ; ;

BINT15 ?CaseKeyDef

:: REPEATER 15 :: ACCION\_ABAJO AppDisplay@ EVAL ; ;

BINT14 ?CaseKeyDef

:: REPEATER 14 :: ACCION\_IZQUIERDA AppDisplay@ EVAL ; ;

BINT16 ?CaseKeyDef

:: REPEATER 16 :: ACCION\_DERECHA AppDisplay@ EVAL ; ;

BINT47 ?CaseKeyDef

:: TRUE ' LAM exit? STO ;

DUP#<7 casedrpfls

BINT37 #=casedrpfls

BINT42 #=casedrpfls

DROP 'DoBadKeyT

;

( #ct #p )

BINT2 #=casedrop

:: ( #ct ) ( PLANO 2 )

BINT10 ?CaseKeyDef

:: REPEATER 10 :: ACCION\_ShiftIzq\_ARRIBA AppDisplay@ EVAL ; ;

BINT15 ?CaseKeyDef

:: REPEATER 15 :: ACCION\_ShiftIzq\_ABAJO AppDisplay@ EVAL ; ;

BINT14 ?CaseKeyDef

:: REPEATER 14 :: ACCION\_ShiftIzq\_IZQUIERDA AppDisplay@ EVAL ; ;

BINT16 ?CaseKeyDef

:: REPEATER 16 :: ACCION\_ShiftIzq\_DERECHA AppDisplay@ EVAL ; ;

DUP#<7 casedrpfls

BINT37 #=casedrpfls

BINT42 #=casedrpfls

DROP 'DoBadKeyT

;

( #ct #p )

BINT3 #=casedrop

:: ( #ct ) ( PLANO 3 )

BINT10 ?CaseKeyDef

ACCION\_ShiftDer\_ARRIBA

BINT15 ?CaseKeyDef

ACCION\_ShiftDer\_ABAJO

BINT14 ?CaseKeyDef

ACCION\_ShiftDer\_IZQUIERDA

BINT16 ?CaseKeyDef

ACCION\_ShiftDer\_DERECHA

DUP#<7 casedrpfls

BINT37 #=casedrpfls

BINT42 #=casedrpfls

DROP 'DoBadKeyT

;

( #ct #p ) ( PLANOS 4,5,6 )

2DROP ( )

'DoBadKeyT ( 'DoBadKey T )

;

Capítulo 34  
Browser 49

El browser nos permite ver varios objetos en una lista y seleccionar alguno de ellos. Las cajas de selección creadas por el comando **CHOOSE** de User RPL están basadas en el browser. Sin embargo, el browser puede hacer muchas cosas más que ese comando **CHOOSE**.

En la calculadora HP 50g hay dos maneras de hacer un browser: la antigua, la cual está presente desde los modelos HP 48, y la nueva, presente a partir de la HP 49g. Este capítulo describirá el nuevo browser, el cual es más fácil de usar. En el próximo capítulo veremos el browser antiguo. Las ventajas del browser antiguo son la selección de varios objetos a la vez y también la manera más sencilla de conseguir la pantalla completa.

La principal diferencia con el comando **CHOOSE** de User RPL es que puedes especificar un message handler, el cual puede ser usado para proporcionar un menú personalizado, para asignarles acciones a las teclas y para algunas otras cosas.

El principal comando del browser 49 es **FLASHPTR Choose3\_**. Este comando tiene los siguientes diagrama de pila:

( meta $título #inicial ::message 🡺 ob TRUE )

( meta $título #inicial ::message 🡺 FALSE )

Esto depende de si el usuario selecciona algo o cancela el browser.

Como una alternativa, puedes reemplazar el comando **FLASHPTR Choose3\_** con el comando **FLASHPTR Choose3Index\_**. Las diferencias son que este último comando no guarda una copia del meta original en la pila virtual y que en lugar del objeto seleccionado, este comando retorna el índice.

34.1 El Meta de Ítems

**meta** es un objeto meta (ver capítulo 12) que contiene los items que serán mostrados en la caja de selección. Todos los tipos de objetos son permitidos, y estos serán descompilados para ser mostrados en la pantalla.

34.2 La Cadena del Título

**$título** es el título. Esta será mostrada en un pequeño rectángulo en la parte alta de la caja de selección. Si la cadena es una cadena vacía, entonces no se mostrará título alguno. Esto puede ser útil cuando el contenido de la caja de selección no necesite explicación. Esto también te permite tener una fila más para mostrar otro item adicional.

34.3 El Ítem Inicial

Cuando el browser se inicia, un ítem ya está resaltado. Usualmente este es el primer item, pero puedes seleccionar a otro como el ítem inicial con el parámetro **#inicial**.

La numeración empieza con cero, no con uno.

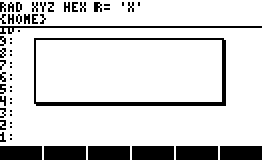
34.4 El Message Handler

**::message** es un programa: el message handler. Un message handler permite disponer de muchas opciones cuando uno programa una aplicación.

La aplicación puede llamar al message handler con diferentes “mensajes” (normalmente un bint) en el nivel uno de la pila, y a veces argumentos adicionales en otros niveles de la pila. El message handler puede decidir si maneja ese mensaje o no. Si lo maneja, este debe hacer ejecutar sus funciones y retornar TRUE. Si decide ignorar el mensaje, entonces sólo debe borrar al bint y poner FALSE en la pila. Un message handler vacío (que no maneja ningún mensaje) es **DROPFALSE** (el cual puedes poner en la pila llamando al comando **'DROPFALSE**). El comando **CHOOSE** de User RPL llama al browser 49 con un message handler vacío y por eso no puede usar todas las funciones disponibles en el browser 49.

Los mensajes que puede manejar el browser 49 son seis:

34.4.1 Message Handler número 1: MsgDispBox



La misión de este es mostrar el perímetro de la caja de selección en la pantalla.

Su diagrama de pila es:

( #1 🡺 TRUE )

( #1 🡺 FALSE )

34.4.2 Message Handler número 2: MsgDispTitle

La misión de este es mostrar el título del browser en la pantalla.

Si no es llamado, el título es mostrado a partir del argumento **$título**.

Su diagrama de pila es:

( #2 🡺 TRUE )

( #2 🡺 FALSE )

Los message handlers 1 y 2 son llamados sólo al inicio (después de guardados los parámetros del browser 49 y fijados los lams **FSize** y **DispLines**), uno a continuación de otro.

Además de mostrar el perímetro de la caja de selección y el título, estos message handlers deben fijar los valores de los lams **YTop** (8LAM) e **YHeight** (4LAM) de la siguiente manera:

**YTop**: Es un bint que define la posición 'Y' del primer ítem mostrado del browser.

**YHeight**: Es un bint y debes fijarlo a **FSize**\***DispLines** – 1 (aunque este no será su valor definitivo, pues después será fijado por la propia calculadora a otro valor: **FSize**\*(**DispLines**-1) – 1, restando **FSize** al valor que tu fijaste).

Si no llamas a los mensajes 1 y 2, se realizarán sus acciones por defecto, para las cuales mostramos un message handler que imita dichas acciones para comprender mejor lo que hacen:

NULLNAME MH\_B49\_1y2\_POR\_DEFECTO

::

BINT1 #=casedrop

:: ( )

7GETLAM ( #DispLines )

9GETLAM ( #DispLines #FSize )

3GETLAM ( #DispLines #FSize flag ) ( TRUE si el título es no vacío )

FLASHPTR 2 63 ( grob ) ( crea un grob con un rectángulo: el contorno )

HARDBUFF ( grob GROB )

BINT16 ( grob GROB 16 )

IsBigApple\_

ITE

# 49 ( grob GROB 16 73 ) ( 73 en la HP 50g )

# 39

4PICK ( grob GROB 16 79 grob )

GROBDIM ( grob GROB 16 79 #h #w )

DROP ( grob GROB 16 79 #h )

DUP ( grob GROB 16 79 #h #h )

#7-\_ ( grob GROB 16 79 #h #h-7 )

4PUTLAM ( grob GROB 16 79 #h )

#- ( grob GROB 16 79-h )

#2/ ( grob GROB 16 [79-h]/2 )

DUP ( grob GROB 16 [79-h]/2 [79-h]/2 )

#3+ ( grob GROB 16 [79-h]/2 [79-h]/2+3 )

8PUTLAM ( grob GROB 16 [79-h]/2 )

GROB! ( )

TRUE ( T )

;

BINT2 #=casedrop

:: ( )

3GETLAM ( flag )

case

:: HARDBUFF ( GROB )

BINT19 ( GROB 19 )

8GETLAM ( GROB 19 #YTop )

2GETLAM ( GROB 19 #YTop $título )

BINT90 ( GROB 19 #YTop $título 90 )

LEFT$3x5 ( GROB' )

BINT19 ( GROB' 19 )

8GETLAM ( GROB' 19 #YTop )

#6+ ( GROB' 19 #YTop+6 )

BINT109 ( GROB' 19 #YTop+6 109 )

OVER ( GROB' 19 #YTop+6 109 #YTop+6 )

LineB ( GROB'' )

DROP ( )

8GETLAM ( #YTop )

#9+ ( #YTop+9 )

8PUTLAM ( )

4GETLAM ( #YHeight )

#9-\_ ( #YHeight-9 )

4PUTLAM ( )

TRUE ( T )

;

8GETLAM ( #YTop )

#1+ ( #YTop+1 )

8PUTLAM ( )

4GETLAM ( #YHeight )

#1- ( #YHeight-1 )

4PUTLAM ( )

TRUE ( T )

;

DROPFALSE

;

34.4.3 Message Handler número 3: MsgEndInit

Este mensaje es ejecutado después de asignados los valores de los lams del browser 49 y dibujados el contorno de la caja y el título, pero antes del dibujado de los ítems y del inicio del POL.

Su diagrama de pila es:

( #3 🡺 TRUE )

( #3 🡺 FALSE )

34.4.4 Message Handler número 4: MsgKeyPress

La misión de este es asignar alguna acción a las teclas. Cuando el usuario presiona una tecla, el message handler es llamado con el código de tecla y el plano (ver sección 31.1), y el bint 4 en la pila. Este debe retornar una asignación para esa tecla (comando o programa), TRUE y nuevamente TRUE.

Si no se dará ninguna asignación a la tecla, debes poner FALSE en la pila.

Su diagrama de pila es:

( #ct #pl #4 🡺 AcciónTecla TRUE TRUE ) (¡Recuerdalo, dos veces TRUE!)

( #ct #pl #4 🡺 #ct #pl FALSE )

34.4.5 Message Handler número 5: MsgMenu

Este debe retornar el menu que sera mostrado al usuario durante la selección. El valor retornado por este mensaje es evaluado para conseguir el menú. El menu no es actualizado automáticamente cuando mueves la selección, pero el mensaje #6 puede ser usado para forzar una actualización. Si el menu tiene más de una página, debes dar asignaciones a las teclas NEXT y PREV en el message handler #4 (pues NEXT y PREV no tienen una asignación predefinida en el browser 49).

Su diagrama de pila es:

( #5 🡺 { menu list } TRUE )

( #5 🡺 ::prog\_que\_retorna\_lista TRUE )

( #5 🡺 FALSE )

34.4.6 Message Handler número 6: MsgEndEndDisp

Este mensaje es llamado inmediatamente antes del inicio del POL. También es llamado después del redibujado del browser 49 cuando cambias al ítem seleccionado. Puedes usar este mensaje para forzar una actualización del menú cada vez que cambias el ítem seleccionado. Para esto debes fijar a 24LAM como FALSE.

Su diagrama de pila es:

( #6 🡺 TRUE )

( #6 🡺 FALSE )

34.5 Los LAMs del Browser 49

El browser 49 usa internamente 24 lams, de tal manera que si tu quieres llamar a otros lams que tu hayas creado anteriormente, es preferible que los hayas creado como lams con nombre. Los LAMs del browser 49 son los siguientes:

**LAM Nombre Contenido**

1LAM **Quit** Un flag. Debes fijar su valor como TRUE para que finalice el POL.

2LAM **DispOffset** Un bint. Es el índice del item seleccionado con respecto al primero visible.

**DispOffset** = 0,1,2,…, **DispLines**-1

Al principio (cuando lo llaman los message handlers 1 y 2) es una cadena que representa al título del browser.

3LAM **DispTop** Un bint. Es el índice del primer ítem que está visible en la pantalla.

Al principio (cuando lo llaman los message handlers 1 y 2) es un flag.

TRUE cuando el título del browser es una cadena no vacía.

FALSE cuando el título del browser es una cadena vacía.

4LAM **YHeight** Un bint. Indica la altura total en píxeles desde el segundo ítem mostrado hasta el último mostrado.

**YHeight** = **FSize** \* (**DispLines** – 1) – 1

5LAM **YTop3** Un bint.

**Ytop3** = **YHeight** + **YTop** + 1

6LAM **YTop2** Un bint. Posición 'Y' del segundo ítem mostrado del browser.

**Ytop2** = **YTop** + **FSize**

7LAM **DispLines** Un bint. Número de elementos mostrados en la pantalla.

8LAM **YTop** Un bint. Posición 'Y' del primer ítem mostrado del browser.

9LAM **FSize** Un bint. Altura de la fuente más uno. Puede ser 7, 8 ó 9.

Depende del flag 90 y/o de la altura de la fuente de sistema actual.

10LAM **Pos** Un bint. Es el índice del ítem seleccionado. Empieza con cero.

11LAM **NbElm** Un bint. Número de ítems, es decir, número de elementos del meta.

12LAM **InvertLeft** Un bint. Es el ancho máximo en píxeles permitido que tendrán los ítems en la pantalla. Su valor por defecto es 86 ó 90 (si hay barra de desplazamiento o no).

13LAM **Lift?** Un flag. Su valor es TRUE cuando **NbElm**>**DispLines**, lo cual permitirá mostrar una barra de desplazamiento a la derecha.

14LAM **Display** Un programa o comando. Su función es dibujar en la pantalla los ítems. Su diagrama de pila debe ser el mismo de **ERASE&LEFT$5x7**:

grob #x #y $ #w 🡺 grob

15LAM **Tme** Un hxs.

16LAM **Str** Una cadena.

17LAM **Message** El message handler.

18LAM Un bint. Ancho en píxeles para la actualización de los ítems a partir de la posición de 19LAM cuando hay una barra de desplazamiento.

Debe ser mayor o igual a **InvertLeft**.

Su valor por defecto es 89.

19LAM Un bint. Posición 'X' para resaltar ítem seleccionado. Generalmente es el valor de 21LAM menos uno. Su valor por defecto es 19.

20LAM Un bint. Posición 'X' para mostrar la barra de desplazamiento.

Su valor por defecto es 107.

21LAM Un bint. Posición 'X' para mostrar un ítem. Su valor por defecto es 20.

22LAM Un bint. **YHeight** +**FSize** +1

23LAM Un programa o comando. Este será evaluado cuando finalice el POL del browser 49.

24LAM Un flag. Fijalo como FALSE cada vez que quieras redibujar el menú.

34.6 Accediendo al Ítem Seleccionado

Para usos del browser que hagan más que sólo seleccionar un ítem, debes escribir programas que sean accesibles por medio del manejador de teclas (mensaje 4) o del menú (mensaje 5). Una de las tareas más comunes en estos programas será poner a disposición del programa el ítem seleccionado.

El browser 49 guarda dos copias de los ítems en la Pila Virtual, y puedes usar estos para conseguir el ítem actual. En el nivel uno de la Pila Virtual (el nivel más reciente), la lista está invertida y los ítems se han convertido a cadenas (la forma como los ítems se convierten a cadenas depende del flag 85). En el nivel 2 de la Pila Virtual están los datos del entorno guardado anterior guardados por el comando PolSaveUI, y estos objetos no tienen relación alguna con el browser actual. En el nivel 3 de la Pila Virtual hay una copia exacta de la lista original.

Por ejemplo, si el meta de los ítems es el siguiente:

%11

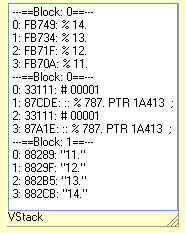
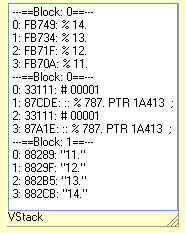
%12

%13

%14

BINT4

Los niveles de la pila virtual serán como se indica en la figura de la izquierda cuando usas el comando **FLASHPTR Choose3\_**. Pero si en lugar de ese comando usas el comando **FLASHPTR Choose3Index\_**, entonces los ítems originales no son guardados en la Pila Virtual y sólo se guardan las cadenas que representan a los ítems, como se muestra en la figura de la derecha.



El índice del ítem actual es disponible por medio del código:

:: 2GETLAM 3GETLAM #+ ;

El índice empieza en cero.

Para conseguir una cadena que representa al ítem actual (desde el nivel 1 de la Pila Virtual), puedes usar el siguiente código:

:: 2GETLAM 3GETLAM #+ GetElemBotVStack ;

Para conseguir el ítem actual original (desde el nivel 3 de la Pila Virtual), puedes usar el siguiente código:

::

GetVStackProtectWord ( #pw1 )

PopMetaVStack ( #pw1 meta1 )

GetVStackProtectWord ( #pw1 meta1 #pw2 )

PopMetaVStack ( #pw1 meta1 #pw2 meta2 )

2GETLAM 3GETLAM #+ ( #pw1 meta1 #pw2 meta2 #índice )

GetElemTopVStack ( #pw1 meta1 #pw2 meta2 ob )

1LAMBIND ( #pw1 meta1 #pw2 meta2 )

PushMetaVStack&Drop ( #pw1 meta1 #pw2 )

SetVStackProtectWord ( #pw1 meta1 )

PushMetaVStack&Drop ( #pw1 )

SetVStackProtectWord ( )

1GETABND ( ob )

;

Para conseguir el ítem actual original, también puedes haber guardado previamente una copia de los ítems originales en una lista, por ejemplo en un LAM MiLista. Si tu haces esto antes de iniciar el browser, podrás conseguir el actual ítem original con el siguiente código:

:: LAM MiLista 2GETLAM 3GETLAM #+ #1+ NTHCOMPDROP ;

34.7 Guardando y Restaurando la Pantalla

Si quieres usar una tecla de menu o alguna otra tecla para hacer una acción que muestre algo en la pantalla, debes guardar y restaurar la pantalla actual alrededor de esa acción. Eso debe hacerse porque el POL del browser 49 no actualiza la totalidad de la pantalla alrededor de una acción que se ejecute. Hay dos comandos flashpointers que puedes usar para guardar y restaurar la pantalla:

**FLASHPTR SaveHARDBUFF\_** Guarda toda la pantalla actual (incluyendo al menú).

**FLASHPTR RestoreHARDBUFF\_** Restaura la pantalla guardada por **^SaveHARDBUFF\_**.

Nota que estos comandos usan un lugar específico para guardar la pantalla actual, por lo tanto estos comandos no pueden ser usados por un browser 49 ejecutado dentro de otro browser 49 que ya haya usado este método para guardar la pantalla antes de llamar al nuevo browser 49. En este caso, necesitarás guardar y restaurar copias de HARDBUFF y HARDBUFF2.

34.8 Referencia

**Direcc. Nombre Descripción**

072002 (ˆChoose3) ( meta $título #pos ::handler 🡺 ob T )

( meta $título #pos ::handler 🡺 F )

El principal comando del browser 49.

074002 (ˆChoose3Index) ( meta $título #pos ::handler 🡺 #i T )

( meta $título #pos ::handler 🡺 F )

Parecido a ˆChoose3, pero retorna el índice del ítem seleccionado en lugar del ítem en sí.

06E002 (ˆChoose2) ( meta $título #pos 🡺 ob T )

( meta $título #pos 🡺 F )

Llama a ^Choose3 con un message handler vacío.

Hace :: 'DROPFALSE FLASHPTR Choose3\_ ;

070002 (ˆChoose2Index) ( meta $título #pos 🡺 #i T )

( meta $título #pos 🡺 F )

Llama a ^Choose3Index con un message handler vacío.

Hace :: 'DROPFALSE FLASHPTR ˆChoose3Index ;

073002 (ˆChoose3Save) ( meta $título #pos ::handler 🡺 ob T )

( meta $título #pos ::handler 🡺 F )

Guarda y restaura toda la pantalla alrededor de una llamada al comando Choose3. Hace:

::

FLASHPTR SaveHARDBUFF\_

FLASHPTR Choose3\_

FLASHPTR RestoreHARDBUFF\_

ClrDAsOK

;

06F002 (ˆChoose2Save) ( meta $título #pos 🡺 ob T )

( meta $título #pos 🡺 F )

Guarda y restaura toda la pantalla alrededor de una llamada al comando Choose2. Hace:

::

FLASHPTR SaveHARDBUFF\_

FLASHPTR Choose2\_

FLASHPTR RestoreHARDBUFF\_

ClrDAsOK

;

005002 (ˆsysCHOOSE) ( $título {} %sel 🡺 ob %1 )

( $título {} %sel 🡺 %0 )

Equivale al comando **CHOOSE** de User RPL.

088002 (ˆSaveHARDBUFF) ( 🡺 )

Guarda HARDBUFF y HARDBUFF2 en un lugar seguro.

089002 (ˆRestoreHARDBUFF) ( 🡺 )

Restaura HARDBUFF y HARDBUFF2 guardados con el comando SaveHARDBUFF.

077002 (ˆChoose3CANCL) ( 🡺 )

La acción CANCL ejecutada por Choose3 si es presionada la tecla CANCL u ON. Hace:

:: TakeOver FALSE 23PUTLAM\_ TRUE 1PUTLAM ;

**Direcc. Nombre Descripción**

076002 (ˆChoose3OK) ( 🡺 )

La acción OK ejecutada por Choose3 si es presionada la tecla ENTER u OK. Hace:

:: TakeOver

TRUE

1PUTLAM

' :: 2GETLAM 3GETLAM #+ TRUE ;

23PUTLAM\_

;

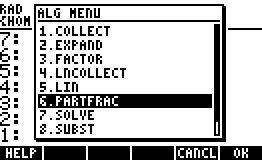
075002 (ˆChooseDefHandler) ( 🡺 ::handler )

Pone el message handler por defecto correspondiente a aplicaciones que muestran ayuda de comandos del CAS.

Cada uno de los ítems deberán ser listas de la forma {$ ob}.

Si ob es un comando del CAS (bibliotecas ya incorporadas 222 y 788), se mostrará su ayuda correspondiente.

No muestra ayuda para otros comandos.



004002 ˆRunChooseSimple ( $título {items} 🡺 ob T )

( $título {items} 🡺 F )

Llama al browser 49 con sólo dos argumentos.

El ítem inicial será el primero.

Guarda y restaura toda la pantalla alrededor de la llamada al browser.

IMPORTANTE: Cuando usas este comando, {items} debe ser una lista de esta forma:

{{ob1 ob1'} {ob2 ob2'} … {ob3 ob3'}}

Sólo el segundo elemento de cada lista es retornado en la pila. Por ejemplo, si el usuario sale con ENTER u OK, en la pila se devolverá ob1', ob2', u obn'.

0630B3 ˜ChooseSimple ( $title {items} 🡺 ob T )

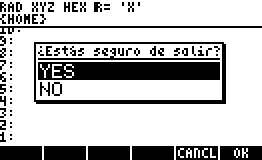
( $title {items} 🡺 F )

Hace lo mismo que ˆRunChooseSimple, pues sólo lo llama.

2F1A5 AskQuestion ( $ 🡺 flag )

Usa la cadena para preguntar al usuario con YES/NO en un browser.

Guarda y restaura toda la pantalla alrededor de la llamada al browser.



**Direcc. Nombre Descripción**

2F21C Lift ( grob #PLM #NLM #NLT #x #y #h 🡺 grob' )

Dibuja una línea de desplazamiento en el grob.

#PLM: Número de la primera línea mostrada.

#NLM: Número de líneas mostradas.

#NLT: Número de líneas totales.

#x, #y: ubicación de la línea de desplazamiento en el grob.

#h: Altura de la línea de desplazamiento en píxeles.

Se debe cumplir:

0 <= #PLM <= #NLT - #NLM

(comando tipo bang)

065002 FLASHPTR 002 064 ( 🡺 #hf+1 )

Retorna un bint que representa a la altura de la fuente que será mostrada más uno. El browser 49 usa este comando para hallar el número que será asignado al nombre local **FSize**

Si el flag 90 está activado, retorna 7.

Si el flag 90 está desactivado, retorna 7,8 ó 9.

065002 FLASHPTR 002 065 ( 🡺 #15-hf )

En la HP 50g retorna un bint que representa a 15 menos la altura de la fuente que será mostrada.

En la HP 49g retorna un bint que representa a 13 menos la altura de la fuente que será mostrada.

Su valor depende del estado del flag 90.

065002 FLASHPTR 002 066 ( 🡺 #16-hf )

En la HP 50g retorna un bint que representa a 16 menos la altura de la fuente que será mostrada.

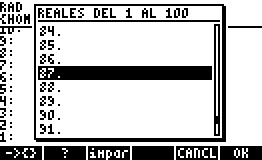
En la HP 49g retorna un bint que representa a 14 menos la altura de la fuente que será mostrada.

Su valor depende del estado del flag 90.

34.9 Ejemplos

Ejemplo 1 Browser 49

**Actualizando el menú.**

****Ejemplo de aplicación del browser 49.

Este programa muestra los números del 1 al 100 y retorna una lista de todos los valores seleccionados.

Presionando la tecla RAIZ muestra la raíz cuadrada del número actual en un mensaje en un rectángulo.

En el menú, presionando F1 agrega la versión descompilada del número seleccionado actualmente (como una cadena) a la lista que será retornada al terminar el browser.

Presionando F2 mostrará un texto de ayuda sobre el programa.

Hay otra tecla de menú, el botón F3, el cual no hace nada, pero muestra si el número seleccionado es par o impar. Debido a que debe cambiar lo mostrado en esta tecla de menú cada vez que cambiamos el ítem actual, necesitamos usar el mensaje número seis para forzar a una actualización del menú.

F5 y F6 hacen las acciones CANCL y OK por defecto.

\* Ejemplo Con el browser 49.

\* Su message handler es el NULLNAME MH\_B49\_EJEMPLO\_1

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B49Ejemplo1 ( -> {$} T // F )

::

CK0 ( ) ( Ningún argumento es requerido )

BINT101 ONE\_DO (DO)

INDEX@

UNCOERCE

LOOP

( %1 %2 ... %100 )

BINT100 ( %1 %2 ... %100 100 )

{}N ( {%} ) ( LISTA CON REALES DEL 1 AL 100 )

DUP ( {%} {%} )

NULL{} ( {%} {%} {} ) ( LISTA VACÍA QUE SE ARMARÁ CON F1 )

{ LAM MiLista LAM Resp } BIND ( {%} ) ( crea entorno temporal )

( {%} ) ( guarda copia de items en LAM MiLista )

INNERCOMP ( meta ) ( convierte la lista en meta )

"REALES DEL 1 AL 100" ( meta $título )

BINT0 ( meta $título #índice )

' MH\_B49\_EJEMPLO\_1 ( meta $título #índice MessHandler )

FLASHPTR Choose3\_ ( % T // F ) ( browser 49 )

ITE

:: DROP ( ) ( borra respuesta del browser 49 )

LAM Resp ( {$} ) ( retorna lista de cadenas )

FLASHPTR COMPRIMext ( {$}' ) ( suprime elementos repetidos )

TRUE ( {$}' T )

;

FALSE ( F )

( {$} T // F )

ABND ( {$} T // F ) ( Destruye el entorno temporal creado con ABND )

;

NULLNAME MH\_B49\_EJEMPLO\_1

:: BINT4 #=casedrop ( manejador de teclas )

:: DUP#1= 3PICK BINT23 #= AND case2drop ( SQRT key pressed )

:: ' :: LAM MiLista ( {%} )

2GETLAM 3GETLAM #+ #1+ ( {%} #i+1 )

NTHCOMPDROP ( % ) ( consigue valor )

%SQRT ( %' ) ( halla raiz )

DO>STR ( $ ) ( convierte a cadena )

FlashWarning ( ) ( Lo muestra )

;

TrueTrue ( Si, se asigna una acción a la tecla )

;

FALSE ( No se manejan otras teclas )

;

BINT5 #=casedrop ( proveer un menú )

:: ' :: NoExitAction ( No guardará el menú como LastMenu )

{

{ "->{}" ( Tecla de menú "Agregar a lista" )

:: TakeOver ( )

LAM Resp ( {$} ) ( Consigue lista actual )

2GETLAM 3GETLAM #+ ( {$} #índice )

GetElemBotVStack ( {$} $item ) ( Consigue cadena )

>TCOMP ( {$}' ) ( Agrega a la lista )

' LAM Resp STO ( ) ( Guarda en LAM Resp )

;

}

{ "?" ( Tecla de menú "Ayuda" )

:: TakeOver ( )

FLASHPTR SaveHARDBUFF\_ ( ) ( Guarda pantalla actual )

CLEARLCD ( ) ( limpia pantalla )

ZEROZERO ( 0 0 )

"->{}: AGREGA A LISTA\0A?: AYUDA\0ASQRT: MUESTRA RAIZ"

$>GROBCR ( 0 0 grob )

XYGROBDISP ( ) ( lo muestra )

WaitForKey ( #ct #p ) ( espera tecla )

2DROP ( )

FLASHPTR RestoreHARDBUFF\_ ( ) ( Restaura pantalla )

;

}

{ :: TakeOver ( )

LAM MiLista ( {} ) ( lista de 100 números )

2GETLAM 3GETLAM #+ #1+ ( {} #i+1 )

NTHCOMPDROP ( %i+1 ) ( elemento actual )

%2 %/ ( %[i+1]/2 )

%FP ( %ParteDecimal )

%0= ( flag ) ( Test if even )

ITE

"par"

"impar" ( $ )

; ( $ ) ( menu: par o impar )

NOP ( No realiza ninguna acción )

}

NullMenuKey ( 4º tecla de menú es vacía )

{ "CANCL" FLASHPTR Choose3CANCL\_ } ( acción CANCL estand )

{ "OK" FLASHPTR Choose3OK\_ } ( acción OK estandar )

}

;

TRUE ( Si, nosotros proveemos el menú )

;

BINT6 #=casedrop ( llamado al cambiar de ítem seleccionado )

:: FalseFalse 24PUTLAM\_ ; ( Fuerza a una actualización del menú )

DROPFALSE ( Otros mensajes no son manejados )

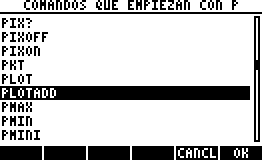
;

Ejemplo 2 Browser 49

**Usando la pantalla completa.**

**Sólo en HP 49g+ y HP 50g.**

**Minifuente y Título no vacío.**

En el siguiente ejemplo se muestra como debemos programar el message handler para que el browser 49 se muestre en pantalla completa.

Se usaron los mensajes 1, 2 y 3.

\* Browser 49 con PANTALLA COMPLETA que muestra los comandos

\* que comienzan con la letra P

\* Para que funcione esto:

\* a) El título debe ser una cadena no vacía

\* b) El flag 90 debe estar activado (CHOOSE:mini font)

\* El comando RunSafeFlags ejecuta el objeto después de el, guardando

\* y restaurando los flags alrededor de la ejecución.

\* El comando FLASHPTR 002 084 llama a un meta cuyos

\* elementos son cadenas con los nombre de comandos

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B49FullScr50g ( -> $ T // F )

::

CK0 ( ) ( no se requieren argumentos )

"P" ( "P" )

FLASHPTR 2 84 ( meta ) ( meta de cadenas )

"COMANDOS QUE EMPIEZAN CON P" ( meta $título )

BINT0 ( meta $título #índice )

' MH\_B49\_MINIFONT\_TITULO ( meta $título #índice MessHandler )

RunSafeFlags

:: BINT90

SetSysFlag ( meta $tít #índice MH ) ( activa flag 90 )

FLASHPTR Choose3\_ ( $ T // F )

;

;

NULLNAME MH\_B49\_MINIFONT\_TITULO

:: BINT1 #=casedrop

:: 11GETLAM ( #NbElm )

BINT9 ( #NbElm 9 )

#MIN ( #min[NbElm,9] )

7PUTLAM ( ) ( guarda en #DispLines )

TRUE ( T )

;

BINT2 #=casedrop

:: BINT9 ( #9 )

8PUTLAM ( ) ( guarda en #YTop )

9GETLAM ( #FSize )

7GETLAM ( #FSize #DispLines )

#\* ( #FSize·DispLines )

#1- ( #FSize·DispLines-1 )

4PUTLAM ( ) ( guarda en #YHeight )

80 131 ( 80 131 )

MAKEGROB ( grob )

BINT66 ( grob 66 )

BINT0 ( grob 66 0 )

2GETLAM ( grob 66 0 $ )

BINT128 ( grob 66 0 $ 128 )

CENTER$3x5 ( grob' ) ( Dibuja título )

BINT0 ( grob' 0 )

BINT6 ( grob' 0 6 )

BINT130 ( grob' 0 6 109 )

BINT6 ( grob' 0 6 109 6 )

LineB ( grob'' ) ( Dibuja línea debajo de título )

HARDBUFF ( grob'' GROB )

ZEROZERO ( grob'' GROB 0 0 )

GROB! ( ) ( MUESTRA grob'' EN LA PANTALLA )

TRUE ( T )

;

BINT3 #=casedrop

:: ' ERASE&LEFT$3x5

14PUTLAM ( Guarda Display )

BINT127 20PUTLAM ( Posición #x de la barra de desplazamiento )

BINT1 21PUTLAM ( Posición #x de los ítems )

BINT0 19PUTLAM ( Posición #x para resaltar ítem seleccionado )

BINT124 18PUTLAM ( Ancho #w para actualizar ítem si Lift?=T )

13GETLAM ( Lift? )

ITE

BINT124

BINT128

12PUTLAM ( ) ( Ancho #w para dibujar ítem )

TRUE ( T )

;

DROPFALSE

;

Ejemplo 3 Browser 49

**Usando la pantalla completa.**

**Cualquier calculadora HP 50g, HP 49g+, HP 49g o HP 48gII.**

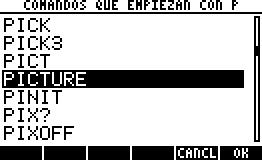
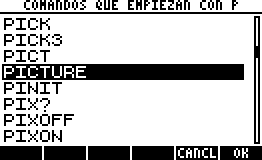
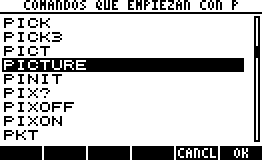
**Cualquier tamaño de fuente.**

**Cualquier título o también sin título funciona.**

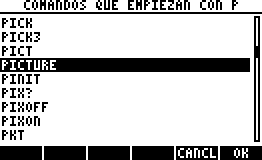
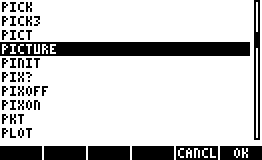
En el siguiente ejemplo se muestra como debemos programar el message handler para que el browser 49 se muestre en pantalla completa.

Se usaron los mensajes 1, 2 y 3.

131x80 FUENTE 8 131x80 FUENTE 7 131x80 FUENTE 6

131x80 MINIFUENTE 131x80 SIN TITULO 131x64 FUENTE 8

\* Browser 49 con PANTALLA COMPLETA que muestra los comandos

\* que comienzan con la letra P

\* Se puede usar en cualquier calculadora HP 50g, HP 49g+, HP 49g o HP 48gII

\* Con cualquier estado del flag 90 y cualquier fuente actual.

\* El comando FLASHPTR 002 084 llama a un meta cuyos

\* elementos son cadenas con los nombre de comandos

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B49FullScreen ( -> $ T // F )

::

CK0 ( ) ( no se requieren argumentos )

"P" ( "P" )

FLASHPTR 002 084 ( meta ) ( meta de cadenas )

"COMANDOS QUE EMPIEZAN CON P" ( meta $título )

BINT0 ( meta $título #índice )

' MH\_B49\_FULLSCREEN ( meta $título #índice MessHandler )

FLASHPTR Choose3\_ ( $ T // F )

;

NULLNAME MH\_B49\_FULLSCREEN

:: BINT1 #=casedrop

:: HARDHEIGHT ( #Altura ) ( 80/64 segun sea Hp50g/Hp49g )

#8-\_ ( #Altura ) ( 72/56 segun sea Hp50g/Hp49g )

3GETLAM ( #Altura flag ) ( T si el titulo es cadena no vacía )

IT #8-\_

( #Altura ) ( altura total disponible para mostrar items )

9GETLAM ( #Altura #FSize )

#/

SWAPDROP ( # ) ( cantidad de items que podrian caber en pantalla )

11GETLAM ( # #NbElm )

#MIN ( #min[#,#NbElm] )

7PUTLAM ( ) ( guarda en #DispLines )

TRUE ( T )

;

BINT2 #=casedrop

:: 9GETLAM ( #FSize )

7GETLAM ( #FSize #DispLines )

#\* ( #FSize·DispLines )

#1- ( #FSize·DispLines-1 )

4PUTLAM ( ) ( guarda en #YHeight )

BINT80 BINT131 ( 80 131 )

MAKEGROB ( grob131x80 )

3GETLAM ( grob131x80 flag ) ( T si el titulo es cadena no vacía )

ITE

::

BINT66 ( grob 66 )

BINT0 ( grob 66 0 )

2GETLAM ( grob 66 0 $ )

BINT128 ( grob 66 0 $ 128 )

CENTER$3x5 ( grob' ) ( Dibuja título )

BINT0 ( grob' 0 )

BINT6 ( grob' 0 6 )

BINT130 ( grob' 0 6 109 )

BINT6 ( grob' 0 6 109 6 )

LineB ( grob'' ) ( Dibuja línea debajo de título )

BINT9

;

BINT1

( grob # )

8PUTLAM ( grob ) ( guarda en #YTop )

HARDBUFF ( grob'' GROB )

ZEROZERO ( grob'' GROB 0 0 )

GROB! ( ) ( MUESTRA grob'' EN LA PANTALLA )

TRUE ( T )

;

BINT3 #=casedrop

::

BINT90

TestSysFlag

ITE

:: ' ERASE&LEFT$3x5 14PUTLAM ( Guarda Display )

BINT1 21PUTLAM ( Posición #x de los ítems )

BINT0 19PUTLAM ( Posición #x para resaltar ítem seleccionado )

BINT128 18PUTLAM ( Ancho #w para actualizar ítem si Lift?=T )

13GETLAM ( Lift? )

ITE

BINT124

BINT128

12PUTLAM ( Ancho #w para dibujar ítem )

;

:: ' ERASE&LEFT$5x7 14PUTLAM ( Guarda Display )

BINT2 21PUTLAM ( Posición #x de los ítems )

BINT1 19PUTLAM ( Posición #x para resaltar ítem seleccionado )

BINT126 18PUTLAM ( Ancho #w para actualizar ítem si Lift?=T )

13GETLAM ( Lift? )

ITE

BINT120

BINT126

12PUTLAM ( Ancho #w para dibujar ítem )

;

BINT127 20PUTLAM ( Posición #x de la barra de desplazamiento )

TRUE ( T )

;

DROPFALSE

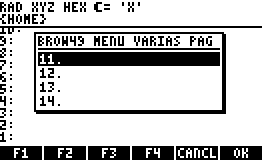
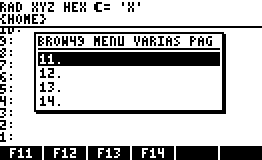
;

Ejemplo 4 Browser 49

**Mostrando un menú con varias páginas.**

En el siguiente ejemplo se muestra como debemos programar el message handler para que al presionar las teclas NEXT y PREV, se puedan mostrar las páginas de un menú grande.

Se usó el mensaje número 4 para manejar a a esas dos teclas y se fijó al lam 24 como FALSE cada vez que se presione las teclas para que se produzca una actualización del menú.



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B49MenuPaginas ( -> % T // F )

:: CK0 ( ) ( no se requieren argumentos )

%11 %12 %13 %14 BINT4 ( meta )

"BROW49 MENU VARIAS PAG" ( meta $título )

BINT0 ( meta $título #índice )

' MH\_B49\_VARIAS\_PAG ( meta $título #índice MessHandler )

FLASHPTR Choose3\_ ( $ T // F )

;

NULLNAME MH\_B49\_VARIAS\_PAG

:: BINT4 #=casedrop ( MANEJADOR DE TECLAS )

:: ( #ct #p )

OVER BINT13 #= OVER #1= AND ( #ct #p flag ) ( TECLA NEXT )

case2drop

:: ' :: DoNextRow ( ) ( Muestra siguiente página del menú )

FALSE 24PUTLAM\_ ( ) ( para redibujar el menú )

;

TrueTrue ( prog T T )

;

OVER BINT13 #= OVER #2= AND ( #ct #p flag ) ( tecla PREV )

case2drop

:: ' :: DoPrevRow ( ) ( Muestra anterior página del menú )

FALSE 24PUTLAM\_ ( ) ( para redibujar el menú )

;

TrueTrue ( prog T T )

;

( #ct #p )

FALSE ( #ct #p F )

;

BINT5 #=casedrop ( MENU DEL BROWSER 49 )

:: { { "F1" :: TakeOver "F1" FlashWarning ; }

{ "F2" :: TakeOver "F2" FlashWarning ; }

{ "F3" NOP }

{ "F4" NOP }

{ "CANCL" FLASHPTR Choose3CANCL\_ }

{ "OK" FLASHPTR Choose3OK\_ }

{ "F7" NOP }

{ "F8" NOP }

{ "F9" NOP }

{ "F10" NOP }

{ "CANCL" FLASHPTR Choose3CANCL\_ }

{ "OK" FLASHPTR Choose3OK\_ }

{ "F11" NOP }

{ "F12" NOP }

{ "F13" NOP }

{ "F14" NOP }

}

TRUE ( menu T )

;

DROPFALSE

;

Capítulo 35  
Browser 48

El browser 48 (el cual también está presente en la HP 50g) te permite hacer muchas cosas. El browser 48 te muestra una lista de ítems de los cuales podrás seleccionar uno de ellos o también marcar a varios de ellos para retornar varios ítems en la pila (no como el browser 49, el cual te permitía solamente seleccionar a un ítem). También puedes colocar un menú personalizado al browser 48.

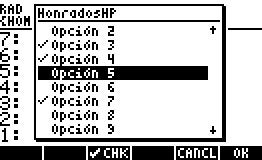
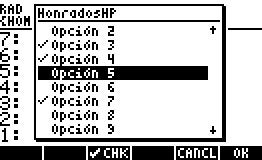
Con el browser 48 es muy sencillo mostrar los elementos en una pantalla completa, a diferencia del browser 49. El browser 48 tiene muchas características, y generalmente hay varias formas para hacer lo mismo.

El browser es llamado por el comando **˜Choose**.Este comando espera cinco parámetros en la pila. Al final retorna los resultados y TRUE, o sólo FALSE, dependiendo de la manera en que se terminó el browser. Aquí están los diagramas de pila:

( MessageHandler $Título ::Convert {}Items Init 🡺 resultado TRUE )

( MessageHandler $Título ::Convert {}Items Init 🡺 FALSE )

Aquí, resultado es o una lista de elementos o un sólo elemento, dependiendo de si es posible la marcación de objetos (selección multiple) o no.

**** ****

35.1 El Parámetro MessageHandler

Este es un programa que permite la configuración de varios aspectos del browser. Este funciona como otros message handlers hacen: este es llamado con un bint en la pila, que representa el código del mensaje. Si este mensaje es manejado, el programa debe retornar cualquier dato requerido por el mensaje y TRUE, de lo contrario este retorna sólo FALSE. Esto significa que **DROPFALSE** (el cual puede ser puesto en la pila con el comando **'DROPFALSE**) es un valor válido para este parámetro, el cual significa que ningún mensaje es manejado y las características por defecto del browser 48 son usadas todas las veces.

Aquí está la descripción de algunos de estos mensajes con sus diagramas de pila, en donde se ha omitido el bint correspondiente al código del mensaje.

**Código**

**(Decimal) Descripción y Pila**

57 Número de líneas que el browser mostrará en la pantalla.

El valor mínimo es 2 y el valor máximo es igual al número de elementos.

El valor por defecto depende del estado del flag 90, de la altura de la fuente, de si se mostrará el título o no y de si el browser es de pantalla completa o no.

Este mensaje es llamado una sóla vez, antes de empezar el browser 48.

( 🡺 # T )

( 🡺 F )

58 Altura de una línea en el browser.

El valor por defecto depende del estado del flag 90 y de la altura de la fuente.

Este valor por defecto es igual al tamaño de la fuente utilizada para mostrar los elementos, más uno (o sea 7, 8 ó 9).

Este mensaje es llamado una sóla vez, antes de empezar el browser 48.

( 🡺 # T )

( 🡺 F )

59 Ancho de una línea en el browser. Este ancho corresponde sólo al espacio donde se muestra a cada elemento (y al espacio para la marca check, si se permite la selección múltiple). Este ancho no incluye al espacio donde se muestran las flechas de arriba y abajo, cuando el número total de elementos es mayor al número de elementos mostrados (tamaño de la página).

El valor por defecto depende de si el browser es de pantalla completa o no, y de si el número total de elementos es mayor al número de elementos mostrados.

Este mensaje es llamado una sóla vez, antes de empezar el browser 48.

( 🡺 # T )

( 🡺 F )

60 Debe retornar TRUE si el browser será de **pantalla completa**, o FALSE si es una ventana. El valor por defecto es mostrar una ventana.

( 🡺 flag T )

( 🡺 F )

61 Debe retornar TRUE si las marcas de verificación son permitidas, es decir cuando se permita la **selección múltiple**, o FALSE si no son permitidas.

El valor por defecto es no permitir marcas de verificación.

( 🡺 flag T )

( 🡺 F )

62 Returna el número de elementos del browser. Si inicialmente quieres mostrar no todos los elementos puedes llamar a este mensaje.

La acción por defecto es hallar el número de elementos de la lista **{}ítems**.

Puedes usar este mensaje cuando el número de elementos cambia.

Normalmente este mensaje es llamado una sóla vez, antes de empezar el browser 48. Pero también será llamado si usas el comando **~BBReReadNElems\_**

( 🡺 # T )

( 🡺 F )

63 Con este mensaje podrás fijar las coordenadas x e y del primer ítem mostrado en la pantalla.

( 🡺 #x #y T )

( 🡺 F )

**Código**

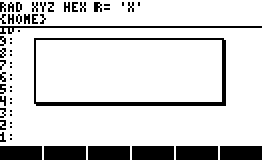
**(Decimal) Descripción y Pila**

64 Con este mensaje podrás fijar el offset inicial. El offset es el lugar que ocupa el elemento seleccionado respecto a los elementos que se están mostrando ahora en la pantalla.

Debes cerciorarte de que sea un número válido para que no suceda un crash.

( 🡺 #offset T )

( 🡺 F )

 65 Este mensaje debe dibujar el fondo de la pantalla. Puedes usar este mensaje para dibujar algo en el fondo.

Si está en modo pantalla completa, la acción por defecto es limpiar la pantalla.

Si está en modo ventana, la acción por defecto es dibujar un rectángulo que es el marco de toda la ventana del browser.

Este mensaje es llamado cuando se va a actualizar la pantalla.

Este es el primer mensaje en ser llamado luego de iniciado el POL del browser 48 y puede ser usado también para modificar algunos parámetros del POL como se ve en uno de los ejemplos al final de este capítulo.

( 🡺 T )

( 🡺 F )

66 Este mensaje debe dibujar el **título** en HARDBUFF. La acción por defecto es dibujar el título a partir del parámetro **$Título** (y una línea debajo del título cuando el browser está en modo ventana).

Este mensaje es llamado cuando se va a mostrar el título en la pantalla.

Si este mensaje es manejado, ya no se llaman a los mensajes 67, 68 y 69 cuando se va a mostrar el título en la pantalla.

( 🡺 T )

( 🡺 F )

67 Este mensaje debe retornar el **título** como un grob.

La acción por defecto es conseguir el grob a partir del parámetro **$Título**.

Este mensaje es llamado cuando se va a mostrar el título en la pantalla.

Si el mensaje 66 es manejado, este mensaje no será llamado.

Si este mensaje es manejado, ya no se llaman a los mensajes 67 y 68 cuando se va a mostrar el título en la pantalla.

( 🡺 grob T )

( 🡺 F )

68 Este mensaje debe retornar el **título** como un grob, cuando el browser está en **modo pantalla completa**.

La acción por defecto es conseguir el grob a partir del parámetro **$Título**.

Este mensaje es llamado cuando se va a mostrar el título en la pantalla.

Si el mensaje 66 o el 67 es manejado, este mensaje no será llamado.

( 🡺 grob T )

( 🡺 F )

**Código**

**(Decimal) Descripción y Pila**

69 Este mensaje debe retornar el **título** como un grob, cuando el browser está en **modo ventana**.

La acción por defecto es conseguir el grob a partir del parámetro **$Título**.

Este mensaje es llamado cuando se va a mostrar el título en la pantalla.

Si el mensaje 66 o el 67 es manejado, este mensaje no será llamado.

( 🡺 grob T )

( 🡺 F )

70 Este mensaje debería retornar la cadena que represente al **título**. Este mensaje puede modificar al parámetro **$Título** o también ignorarlo.

La acción por defecto es retornar el parámetro **$Título** si es una cadena o su mensaje de error correspondiente si es un bint.

Este mensaje es llamado cuando no se llama a ninguno de los mensajes 66, 67, 68 o 69.

Este mensaje es llamado cuando se va a mostrar el título en la pantalla y el parámetro **$Título** no es una cadena vacía.

( 🡺 $ T )

( 🡺 F )

74 Este mensaje debería mostrar los elementos de la lista **{}ítems** en la pantalla.

También debería mostrar la flecha arriba y/o la flecha abajo que se muestran a la derecha si es necesario.

Este mensaje es llamado cuando se van a mostrar los elementos en la pantalla.

Si este mensaje es manejado, ya no se llaman a los mensajes 79, 81, 82 y 80 cuando se van a mostrar los elementos en la pantalla. Pero si se va a mostrar un solo elemento con el comando **ˆLEDispItem**,este mensaje no es llamado y si es llamado el mensaje 79, 81, 82 y/o 80.

( 🡺 T )

( 🡺 F )

79 Este mensaje debería mostrar un elemento del browser. Si este es el elemento actualmente seleccionado, este mensaje debe mostrar este elemento en fondo invertido o mostrar de alguna otra forma que este es el elemento seleccionado.

Este mensaje es llamado cuando se va a mostrar un elemento en la pantalla.

Si este mensaje es manejado, entonces el parámetro **::Converter** no será tomado en cuenta para mostrar el objeto en la pantalla.

Si este mensaje es manejado, ya no se llaman a los mensajes 81, 82 y 80 cuando se va a mostrar un elemento en la pantalla.

( #índice #offset 🡺 T )

( #índice #offset 🡺 #índice #offset F )

**Código**

**(Decimal) Descripción y Pila**

80 Este mensaje debe retornar un objeto de acuerdo al índice proporcionado, que corresponde a un elemento. Este mensaje puede retornar el elemento correspondiente o cualquier otro objeto, pues **::Converter** se encargará después de convertirlo a cadena.

Si deseas cambiar dinámicamente los items mostrados en pantalla este mensaje te permite hacer eso, aunque el 82 es probablemente mejor para este fin.

Este mensaje es llamado cuando se va a mostrar un elemento en la pantalla, cuando se hace una búsqueda alfabética y cuando se presiona ENTER u OK.

Si el mensaje 79, 81 o el 82 es manejado, este mensaje no será llamado para mostrar objetos en la pantalla.

Sin embargo, este mensaje es siempre llamado para retornar el objeto que será colocado en la pila al finalizar el browser con la tecla ENTER u OK.

( # 🡺 ob T )

( # 🡺 # F )

81 Este mensaje debe convertir un elemento a grob. Este grob debe ser de dimensiones 7NULLLAMx8NULLLAM. Si está permitida la marcación de objetos, debes incorporar la marca de verificación en el grob si el elemento está marcado.

Si este mensaje es manejado, entonces el parámetro **::Converter** no será tomado en cuenta para mostrar el objeto en la pantalla.

Este mensaje es llamado cuando se va a mostrar un elemento en la pantalla.

Si el mensaje 79 es manejado, este mensaje no será llamado.

Si este mensaje es manejado, ya no se llaman a los mensajes 82 y 80 cuando se va a mostrar un elemento en la pantalla.

( # 🡺 grob T )

( # 🡺 # F )

82 Este mensaje debe convertir un elemento a cadena.

Este mensaje es llamado cuando se va a mostrar un elemento en la pantalla.

Si este mensaje es manejado, entonces el parámetro **::Converter** no será tomado en cuenta para mostrar el objeto en la pantalla.

Si el mensaje 79 o el 81 es manejado, este mensaje no será llamado para mostrar objetos en la pantalla.

Si este mensaje es manejado, ya no se llama al mensaje 80 cuando se va a mostrar un elemento en la pantalla.

( # 🡺 $ T )

( # 🡺 # F )

83 Debe proporcionar una lista que describa al menu. También puede ser un programa que retorne una lista. El formato de la lista es el mismo que todo menú y se explica en la sección 40.1.

( 🡺 {} T )

( 🡺 prog T )

( 🡺 F )

84 Este mensaje es llamado antes de que el browser se inicie y debe guardar los grobs correspondientes a HARDBUFF y al menú (en LAM5 y LAM4).

( 🡺 T )

( 🡺 F )

**Código**

**(Decimal) Descripción y Pila**

85 Este mensaje es llamado antes de que el browser se inicie (antes de iniciarse el POL), pero despúes de que todas las variables locales han sido fijadas.

En este mensaje, si se retorna TRUE o FALSE no hay ninguna diferencia.

( 🡺 T )

( 🡺 F )

86 Este es llamado cuando un elemento es marcado o desmarcado. Este mensaje debe guardar en LAM15 la lista de los elementos marcados. Este mensaje no dibuja nada en la pantalla.

( # 🡺 T )

( # 🡺 # F )

87 Este mensaje es llamado justo antes de que el browser finalice (pero después de finalizado el POL). La acción por defecto es devolver los grob de la pantalla y del menú a su estado anterior a la ejecución del browser (usando los grobs contenidos en LAM4 y LAM5).

En la pila se encuentran los objetos que retorna el browser al finalizar. Si deseas puedes cambiar estos objetos (ob es una lista de elementos o un solo elemento, esto depende de si estaba permitida la marcación de objetos).

( ob T 🡺 ob T T ) ( ob T 🡺 ??? T )

( ob T 🡺 ob T F ) ( ob T 🡺 ??? F )

( F 🡺 F T ) ( F 🡺 ??? T )

( F 🡺 F F ) ( F 🡺 ??? F )

91 Este es llamado cuando el usuario presiona la tecla ON o la tecla de menú CANCL. Si el flag es TRUE, el browser finaliza. Si es FALSE, el browser continua.

( 🡺 flag T )

( 🡺 F )

96 Este es llamado cuando el usuario presiona la tecla ENTER o la tecla de menú OK. Si el flag es TRUE, el browser finaliza. Si es FALSE, el browser continua.

( 🡺 flag T )

( 🡺 F )

35.2 El Parámetro $Título

Este parámetro especifica el título y es una cadena. También puede ser un bint y en este caso se muestra el mensaje de error correspondiente a ese bint en el título.

Hay mensajes que pueden hacer caso omiso de este parámetro: 66, 67, 68, 69 y 70.

35.3 El Parámetro ::Converter

Este es un programa o un bint. Su misión es convertir cualquiera que sea el elemento de la lista **{}items** a cadena para que esta sea mostrada después en la pantalla.

Este converter es llamado cuando se actualiza la pantalla (a menos que sea manejados los mensajes 79, 81 o el 82). El converter recibe el objeto retornado por el mensaje 80 (si este mensaje no es llamado, recibe al elemento de la lista **{}items**) y debe convertirlo en una cadena. Puede ser:

a) Un PROGRAMA. Si el parámetro es un programa, su diagrama de pila debe ser

( ob 🡺 $ )

b) Un BINT. Si el parámetro es un bint, puede ser cualquiera de estos:

• #1: No se hará ninguna descompilación al elemento.

Usar este bint cuando todos los elementos son cadenas.

Las cadenas se mostrarán sin comillas dobles.

• #2: Los números se mostrarán en el formato numérico actual.

Los ids se mostrarán sin comillas.

Las cadenas se mostrarán con comillas dobles.

• #4: Los números se mostrarán en el formato numérico estándar.

Los ids se mostrarán sin comillas.

Las cadenas se mostrarán con comillas dobles.

También puedes agregar (sumando al anterior) uno de estos bints:

• #16: Cuando el ítem es una lista y quieres mostrar el primer elemento de esta lista.

• #32: Cuando el ítem es una lista y quieres mostrar el segundo elemento de esta lista.

• #8: Se mostrará sólo el primer carácter.

Por ejemplo, si cada uno de los ítems son listas de esta forma {$ ob}, puedes fijar el parámetro **::Converter** como 17 (16+1), lo cual significa que se tomará el primer elemento de la lista y esta cadena se mostrará directamente.

El **::Converter** no es llamado en la actualización de la pantalla cuando son manejados los mensajes 79, 81 o el 82). Sin embargo, el **::Converter** siempre es llamado en la búsqueda alfabética (cuando el usuario presiona alpha seguida de una letra para ir a un ítem que empiece con esa letra). Por lo tanto, debes asegurarte que de alguna forma este parámetro siempre retorne una cadena. Un valor muy útil para este parámetro puede ser el comando **DO>STR**.

35.4 El Parámetro {}Items

Puedes especificar una lista de objetos aquí, o puedes también poner una lista vacía y usar los mensajes 74, 79, 81, 82 o el 80 para mostrar los elementos o proveerlos.

34.5 El Parámetro Init

Este parámetro puede ser un entero binario o una lista.

Si el bint es cero, el browser funcionará como visor y no se podrán seleccionar elementos. Si es otro bint, este corresponderá al elemento seleccionado inicialmente.

Si está permitida la marcación de objetos (selección múltiple), el parámetro **Init** también puede ser una lista de bints, donde el primer bint indica el elemento seleccionado inicialmente y los restantes bints indican los elementos marcados inicialmente. Por ejemplo si la lista es

{ BINT1 BINT1 BINT3 BINT4 }

entonces al inicio el elemento 1 será seleccionado y los elementos 1, 3 y 4 serán los marcados.

35.6 Uso Típico del Browser 48

Leyendo la descripción de los mensajes y de los parámetros arriba, probablemente habrás notado que hay varias formas de proveer los elementos del browser, y tal vez estés confundido sobre esto. Aquí se muestran dos maneras para proveer los elementos que formarán parte del browser.

1. Puedes proveer los elementos del browser usando el parámetro **{}Items**,y proveer un **::Converter** que convertirá cada uno de estos elementos en cadena. No necesitarás preocuparte de los mensajes 80, 81 ó 82.

Este método es bueno si los elementos no cambian mientras se ejecuta el programa.

2) Puedes dejar el parámetro **{}Items** como una lista vacía, y guardar la lista de elementos en algún otro lugar (por ejemplo en un lam).

Luego, usar los mensajes 80, 81 ó 82 para retornar los elementos.

a) Si usas los mensajes 81 o 82, retornarás los elementos como un grob o como una cadena, y **::Converter** no será llamado para mostrar los elementos en la pantalla. Sin embargo, **::Converter** siempre será llamado al hacer la búqueda alfabética y debes asegurarte que siempre retorne una cadena. También deberás llamar al mensaje 80 para retornar el elemento actual que será usado en la búsqueda alfabética y al finalizar el browser con ENTER u OK.

b) O también puedes usar el mensaje 80 (y no llamar al 81 o al 82) para retornar un elemento que será mostrado en la pantalla, el cual será convertido luego a cadena con el parámetro **::Converter**.

Este método es bueno si los elementos cambian mientras se ejecuta el programa. Si usas este método, deberás manejar el mensaje 62.

Cuando el número de elementos cambia, debes ejecutar este código para adaptar el browser a los cambios:

::

ROMPTR 0B3 03A ( ) ( RESTAURA LA PANTALLA A SU ESTADO ANTERIOR AL B48 )

ROMPTR BBReReadNElems\_ ( ) ( Relee el nº total de elementos )

ROMPTR BBReReadPageSize\_ ( ) ( Relee el nº de elementos mostrados )

ROMPTR BBReReadWidth\_ ( ) ( Relee ancho de elementos en píxeles )

ROMPTR BBReReadCoords\_ ( ) ( Relee coord. esquina superior izq. )

ROMPTR 0B3 039 ( ) ( GUARDA PANTALLA DEL ESTADO ANTERIOR AL B48 )

18GETLAM ( #Indice )

12GETLAM ( #Indice #NumElementos )

DUP#0=IT

DROPONE

( #Indice #NumElementos' )

#MIN ( min[#Indice,#NumElementos] )

18PUTLAM ( )

FALSE ( F )

ROMPTR BBRecalOff&Disp\_ ( ) ( Recalcula offset )

;

35.7 NULLLAMs Usados por el Browser 48

El Browser 48 usa 22 nombres locales sin nombre para guardar su información.

Aquí está la descripción de cada uno de ellos.

**Lam Descripción Tipo**

1 Contador usado por CACHE n/a

2 Condición de salida del POL. flag

Si es TRUE, el browser finalizará.

3 Estado inicial de la pantalla. Es una lista de la forma: {}

{ DA1IsStatFlag DA2bEditFlag DA1BadFlag

DA2aBadFlag DA2bBadFlag DA3BadFlag }

4 Menu anterior al browser 48 grob 131x8

5 Pantalla anterior al browser 48 grob 131x72

6 Offset #

7 Altura de cada item en píxeles #

8 Ancho de un elemento en píxeles sin contar barra #

de la derecha

9 coordenada y del primer ítem mostrado #

10 coordenada x del primer ítem mostrado #

11 Número de elementos mostrados #

12 Número total de elementos #

13 Menú {} ó programa

14 ¿Pantalla completa? flag

15 Lista de índices de los elementos marcados cuando se {#}

permite la marcación de objetos.

Si no hay elementos marcados es una lista vacía.

Si no se permite la marcación de objetos, sólo es MINUSONE.

16 ¿Es posible marcar elementos? flag

17 TRUE si es un visor, FALSE si es un seleccionador. flag

18 Indice del ítem seleccionado actualmente. #

19 **{}Items** {}

20 **::Converter** :: ó #

21 **$Título** $

22 **::MessageHandler** ::

35.8 Referencia

**Direcc. Nombre Descripción**

0000B3 ˜Choose ( ::Appl Título ::Convert {}items índice 🡺 {}' T )

( ::Appl Título ::Convert {}items índice 🡺 ob T )

( ::Appl Título ::Convert {}items índice 🡺 F )

**Título** es una cadena o un bint.

**Índice** es un bint o una lista de bints.

El valor retornado es una lista con varios elementos si se permite la marcación de elementos (selección múltiple). De lo contrario, se retorna un sólo elemento.

Sólo FALSE es retornado cuando el usuario presiona ON o la tecla de menú CANCL.

Puedes hacer que los objetos retornado sean otros con el mensaje 87.

Si los elementos del browser no se proporcionan a partir del parámetro **{}ítems**,deberás usar el mensaje 80 para retornar el ítem actual que será colocado en la pila al finalizar con ENTER u OK.

0050B3 ˜ChooseMenu0 ( 🡺 {} )

Menús con "OK".

0060B3 ˜ChooseMenu1 ( 🡺 {} )

Menús con "CANCL", "OK".

0070B3 ˜ChooseMenu2 ( 🡺 {} )

Menús con "CHK", "CANCL", "OK".

09F002 ˆDoCKeyCheck ( 🡺 )

Marca o desmarca el item seleccionado actualmente.

0A0002 ˆDoCKeyChAll ( 🡺 )

Marca todos los elementos.

0B0002 ˆDoCKeyUnChAll ( 🡺 )

Desmarca todos los elementos.

09E002 ˆDoCKeyCancel ( 🡺 )

Simula a la tecla CANCL.

09D002 ˆDoCKeyOK ( 🡺 )

Simula a la tecla OK.

0290B3 (˜BBRunCanclAction) ( 🡺 flag )

Si el mensaje 91 no es usado, retorna TRUE.

Si el mensaje 91 es usado, retorna TRUE o FALSE, que es el flag que devuelve ese mensaje 91 al ser llamado.

Este comando no finaliza el browser.

0280B3 (˜BBRunENTERAction) ( 🡺 flag )

Si el mensaje 96 no es usado, retorna TRUE.

Si el mensaje 96 es usado, retorna TRUE o FALSE, que es el flag que devuelve ese mensaje 96 al ser llamado.

Este comando no finaliza el browser.

04B0B3 (˜BBIsChecked?) ( #i 🡺 flag )

Retorna TRUE si el elemento está marcado.

05B0B3 (˜BBEmpty?) ( 🡺 flag )

Retorna TRUE si el browser no tiene elementos.

Hace :: 12GETLAM #0= ;

0560B3 ROMPTR 0B3 056 ( 🡺 flag )

Retorna TRUE si el título no es una cadena vacía.

**Direcc. Nombre Descripción**

03B0B3 (˜BBRereadChkEnbl) ( 🡺 )

Vuelve a llamar al mensaje 61 para decidir si se permite la marcación de objetos (selección múltiple) y fijarlo en LAM16.

03C0B3 (˜BBRereadFullScr) ( 🡺 )

Vuelve a llamar al mensaje 60 para decidir si se permite la pantalla completa y fijarlo en LAM14.

03D0B3 (˜BReReadMenus) ( 🡺 )

Vuelve a llamar mensaje 83 para guardar el menú en LAM13.

03E0B3 (˜BBReReadNElems) ( 🡺 )

Vuelve a llamar al mensaje 62 para fijar el número de elementos y guardarlo en LAM12.

0240B3 (˜BBReReadHeight) ( 🡺 )

Vuelve a llamar al mensaje 58 para fijar la altura de una línea del browser y guardarla en LAM7.

0230B3 (˜BBReReadPageSize) ( 🡺 )

Vuelve a llamar al mensaje 57 para fijar el número de líneas que el browser mostrará en la pantalla y guardarlo en LAM11.

0260B3 (˜BBReReadWidth) ( 🡺 )

Vuelve a llamar al mensaje 59 para fijar el ancho que tendrá cada línea del browser y guardarlo en LAM8.

0250B3 (˜BBReReadCoords) ( 🡺 )

Vuelve a llamar al mensaje 63 para fijar las coordenadas de la esquina superior izquierda del primer ítem mostrado en la pantalla y guardarlos en LAM10 y LAM9.

04A0B3 ROMPTR 0B3 04A ( 🡺 )

Si se permite la marcación de objetos, fija los elementos que están actualmente marcados (LAM15) y el elemento actualmente seleccionado (LAM18).

0270B3 ROMPTR 0B3 027 ( 🡺 )

Vuelve a llamar al mensaje 64 para fijar el offset y guardarlo en LAM6.

0220B3 (˜BBRunEntryProc) ( 🡺 )

Ejecuta el mensaje número 85. Es decir, ejecuta el programa inicial definido por el usuario.

0190B3 (˜BBRecalOff&Disp) ( flag 🡺 )

Recalcula el offset correspondiente al item seleccionado actualmente y lo guarda en LAM6.

Si el flag es TRUE, invierte los pixels de la línea actual.

0370B3 (˜BBGetNGrob) ( #n 🡺 grob )

Retorna el elemento de orden #i como un grob (mensaje 81).

0380B3 (˜BBGetNStr) ( #n 🡺 $ )

Retorna el elemento de orden #i como una cadena (mensaje 81 o mensaje 80 y converter).

03F0B3 (˜BBGetN) ( #i 🡺 ob )

Retorna el elemento de orden i.

Llama al mensaje 80. Si este no es manejado, retorna el elemento desde {}items.

0080B3 ROMPTR 0B3 008 ( #kc #pl 🡺 KeyDef T )

( #kc #pl 🡺 F )

Ejecuta el AppKeys por defecto correspondiente al browser 48, llamando a la definición de la tecla especificada.

Para tener a ese AppKeys en la pila, debes hacer:

:: ' ROMPTR 0B3 008 ROMPTR@ DROP ;

35.8.1 Cambiando el Ítem Actual

**Direcc. Nombre Descripción**

01A0B3 ROMPTR 0B3 01A ( 🡺 )

Mueve la selección un elemento hacia arriba.

05A0B3 (˜BBPgUp) ( 🡺 )

Fija valores para lams 6 y 18 (cuando es un visor) para el posterior uso de ROMPTR 0B3 01B.

01B0B3 ROMPTR 0B3 01B ( 🡺 )

Mueve la selección un elemento hacia arriba.

01C0B3 ROMPTR 0B3 01C ( 🡺 )

Mueve la selección un elemento hacia abajo.

0590B3 (˜BBPgDown) ( 🡺 )

Fija valores para lams 6 y 18 (cuando es un visor) para el posterior uso de ROMPTR 0B3 01D.

01D0B3 ROMPTR 0B3 01D ( 🡺 )

Mueve la selección un elemento hacia abajo.

01E0B3 ROMPTR 0B3 01E ( 🡺 )

Mueve la selección una página hacia arriba.

01F0B3 ROMPTR 0B3 01F ( 🡺 )

Mueve la selección una página hacia abajo.

0200B3 ROMPTR 0B3 020 ( 🡺 )

Mueve la selección al primer ítem.

0210B3 ROMPTR 0B3 021 ( 🡺 )

Mueve la selección al último ítem.

0150B3 (˜BBMoveTo) ( # 🡺 )

Mueve la selección al ítem indicado (además actualiza la pantalla cuando el primer ítem mostrado no cambia).

35.8.1 Dibujando la Pantalla del Browser 48

**Direcc. Nombre Descripción**

0010B3 ROMPTR 0B3 001 ( 🡺 )

Ejecuta el AppDisplay por defecto correspondiente al browser 48. Es decir, redibuja todo.

Para tener a ese AppDisplay en la pila, debes hacer:

:: ' ROMPTR 0B3 001 ROMPTR@ DROP ;

0020B3 ROMPTR 0B3 002 ( 🡺 )

Redibuja el fondo de la pantalla. Llama a ˜BBReDrawBackgr.

02F0B3 (˜BBReDrawBackgr) ( 🡺 )

Redibuja el fondo de la pantalla Llama al mensaje 65.

0030B3 ROMPTR 0B3 003 ( 🡺 )

Redibuja el título del browser. Llama a ˆLEDispPrompt.

0B3002 ˆLEDispPrompt ( 🡺 )

Redibuja el título.

0040B3 ROMPTR 0B3 004 ( 🡺 )

Redibuja las líneas del browser. Llama a ˆLEDispList.

0B2002 ˆLEDispList ( 🡺 )

Redibuja todas las líneas del browser (muestra los elementos).

Llama al mensaje 74. Si este no es manejado, dibuja los ítems con el comando ˆLEDispItem.

**Direcc. Nombre Descripción**

0B1002 ˆLEDispItem ( #índice #offset 🡺 )

Redibuja un elemento. Si #índice corresponde al elemento actualmente seleccionado, este elemento es dibujado en fondo oscuro.

05C0B3 (˜BBGetDefltHeight) ( 🡺 # )

Retorna la altura de las líneas basada en la fuente que el browser usa para mostrar los elementos (depende del flag 90 y de la altura de la fuente de sistema).

Este valor es la altura por defecto de las líneas.

Su valor es de 7, 8 ó 9 (1 más que la altura de la fuente usada).

Es equivalente al comando FLASHPTR 2 64.

0390B3 ROMPTR 0B3 039 ( 🡺 )

Guarda la pantalla y el grob del menú anteriores al browser en los lams 4 y 5. Estos se restaurarán cuando finalice el browser.

0390B3 ROMPTR 0B3 03A ( 🡺 )

Restaura la pantalla y el grob del menú anteriores al browser a partir de los grobs guardados en los lams 4 y 5.

0520B3 (˜BBUpArrow) ( 🡺 grob )

Retorna la flecha arriba como un grob.

0530B3 (˜BBDownArrow) ( 🡺 grob )

Retorna la flecha abajo como un grob.

0540B3 (˜BBSpace) ( 🡺 grob )

Retorna el carácter espacio, pero como un grob.

35.9 Ejemplos

Ejemplo 1 Browser 48

**Cuando los items son cadenas.**

En este ejemplo los items son cadenas.

El parámetro ::Converter es BINT1.

También se pudo colocar en el parámetro ::Converter a

'NOP

o también a

' :: ;

\* Browser 48 para seleccionar

\* En este ejemplo cada item es una cadena

\* El parámetro ::Converter es BINT1

\* Esto significa que:

\* No se hará ninguna descompilación al elemento.

\* Usar este bint cuando todos los elementos son cadenas.

\* Las cadenas se mostrarán sin comillas dobles.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B48Cadenas ( -> ob T // F )

::

CK0 ( )

'DROPFALSE ( MessageHandler )

"Título del browser 48" ( MH $título )

BINT1 ( MH $título converter )

{ "Item 1"

"Item 2"

"Item 3"

"Item 4"

"Item 5"

"Item 6"

"Item 7"

"Item 8"

"Item 9"

"Item 10" } ( MH $título converter {items} )

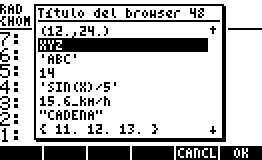
BINT2 ( MH $título converter {items} #índice )

ROMPTR Choose ( ob T // F )

;

Ejemplo 2 Browser 48

**Cuando los items son cualquier objeto.**

En este ejemplo los items son cualquier objeto.

El parámetro ::Converter usado en este ejemplo es BINT4.

\* Browser 48 para seleccionar

\* En este ejemplo cada item es cualquier objeto.

\* El parámetro ::Converter es BINT4

\* Esto significa que:

\* Los números se mostrarán en el formato numérico estándar.

\* Los ids se mostrarán sin comillas.

\* Las cadenas se mostrarán con comillas dobles.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B48CualquierOb ( -> ob T // F )

::

CK0 ( )

'DROPFALSE ( MessageHandler )

"Título del browser 48" ( MH $título )

BINT4 ( MH $título converter )

{ 11.

C% 12. 24.

ID XYZ

LAM ABC

ZINT 14

SYMBOL ID X xSIN Z5\_ x/ ;

UNIT % 15.6 CHR \6B "m" umP "h" um/ umEND ;

"CADENA"

{ 11. 12. 13. }

18. 19. 20. } ( MH $título converter {items} )

BINT2 ( MH $título converter {items} #índice )

ROMPTR Choose ( ob T // F )

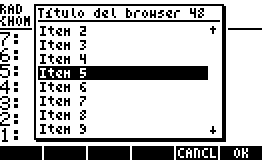
;

Ejemplo 3 Browser 48

**Cuando cada item es una lista que contiene dos elementos.**

**En pantalla se mostrará el primer elemento.**

**Al final se retorna el segundo elemento.**

En este ejemplo cada item es una lista con dos elementos.

El primer elemento es una cadena y el segundo elemento es cualquier objeto.

El primer elemento será mostrado en la pantalla en el browser 48.

El segundo elemento será puesto en la pila al confirmar con ENTER u OK.

El parámetro ::Converter usado en este ejemplo es BINT17

\* Browser 48 para seleccionar

\* En este ejemplo cada item es una lista con dos elementos.

\* El primer elemento es una cadena y el segundo elemento es un objeto.

\* El parámetro ::Converter es BINT17 (17=16+1)

\* Esto significa que:

\* 16: Se mostrará el primer elemento de la lista

\* 1: No se hará ninguna descompilación al elemento y

\* Las cadenas se mostrarán sin comillas dobles.

\* Al final, si se confirma con ENTER u OK, se retorna el 2º elemento

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B48Lista2 ( -> {} T // F )

::

CK0 ( )

'DROPFALSE ( MessageHandler )

"Título del browser 48" ( MH $título )

BINT17 ( MH $título converter )

{ { "Item 1" 11. }

{ "Item 2" 12. }

{ "Item 3" 13. }

{ "Item 4" 14. }

{ "Item 5" 15. }

{ "Item 6" 16. }

{ "Item 7" 17. }

{ "Item 8" 18. }

{ "Item 9" 19. }

{ "Item 10" 20. } } ( MH $título converter {items} )

BINT2 ( MH $título converter {items} #índice )

ROMPTR Choose ( ob T // F )

ITE

:: TWONTHCOMPDROP\_

TRUE

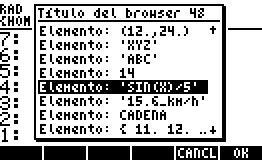
;

FALSE

;

Ejemplo 4 Browser 48

**Cuando el ::Converter es un objeto programa que realiza una acción sobre cada item.**

****

En este ejemplo el parámetro ::Converter es un objeto programa que realiza una acción sobre cada item, lo cual luiego será mostrado en la pantalla.. En este caso le agrega la palabra “elemento” a la izquierda a cada item.

\* Browser 48 para seleccionar

\* En este ejemplo

\* El parámetro ::Converter es un programa

\* que realiza una acción sobre cada item

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B48ConverterP ( -> ob T // F )

::

CK0 ( )

'DROPFALSE ( MessageHandler )

"Título del browser 48" ( MH $título )

' :: "Elemento: "

SWAP

DO>STR

&$

; ( MH $título converter )

{ 11.

C% 12. 24.

ID XYZ

LAM ABC

ZINT 14

SYMBOL ID X xSIN Z5\_ x/ ;

UNIT % 15.6 CHR \6B "m" umP "h" um/ umEND ;

"CADENA"

{ 11. 12. 13. }

18.3 19. 20. } ( MH $título converter {items} )

BINT2 ( MH $título converter {items} #índice )

ROMPTR Choose ( ob T // F )

;

Ejemplo 5 Browser 48

**Usando la pantalla completa del browser 48**

En este ejemplo se podrá usar la pantalla completa del browser 48.

Para esto se usa el message handler número 60.

\* Browser 48 con PANTALLA COMPLETA

\* En este caso se usa el

\* Message Handler 60

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B48PantallaComp ( -> ob T // F )

::

CK0 ( )

' MH\_B48\_PantallaComp ( MessageHandler )

"Browser 48 pant completa" ( MH $título )

BINT4 ( MH $título converter )

{ 11.

12.

13.

14.

15.

16.

17.

18.

19.

20. } ( MH $título converter {items} )

BINT2 ( MH $título converter {items} #índice )

ROMPTR Choose ( ob T // F )

;

NULLNAME MH\_B48\_PantallaComp

:: BINT60 #=casedrop ( Pantalla completa )

TrueTrue

DROPFALSE

;

Ejemplo 6 Browser 48

**Usando la selección múltiple en el browser 48**

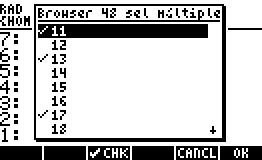
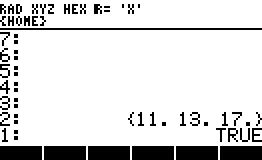
En este ejemplo se podrá usar la selección múltiple en el browser 48.

Para esto se usa el message handler número 61.

En el parámetro #indice se colocó la lista:

{ BINT1 BINT1 BINT3 BINT7 }

Lo cual significa que inicialmente estará seleccionado el elemento número 1 e inicialmente estarán marcados los elementos 1, 3 y 7.



\* Browser 48 con SELECCIÓN MÚLTIPLE

\* En este caso se usa el Message Handler 61

\* En el parámetro #indice se colocó la lista:

\* { BINT1 BINT1 BINT3 BINT7 }

\* Lo cual significa que inicialmente estará seleccionado el elemento 1

\* e inicialmente estarán marcados los elementos 1, 3 y 7.

\* Al retirarse con OK o ENTER, se retorna una lista con los ítems marcados

\* Si ningún ítem está marcado, entonces se retorna una lista con

\* un único objeto que es el ítem seleccionado.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B48SeleccionMult ( -> {#} T // F )

::

CK0 ( )

' MH\_B48\_SeleccionMul ( MessageHandler )

"Browser 48 sel múltiple" ( MH $título )

BINT4 ( MH $título converter )

{ 11.

12.

13.

14.

15.

16.

17.

18.

19.

20. } ( MH $título converter {items} )

{ BINT1 BINT1 BINT3 BINT7 } ( MH $título converter {items} #índice )

ROMPTR Choose ( ob T // F )

;

NULLNAME MH\_B48\_SeleccionMul

:: BINT61 #=casedrop ( Selección múltiple )

TrueTrue

DROPFALSE

;

Ejemplo 7 Browser 48

**Usando el browser 48 como visor.**

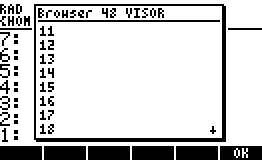
En este ejemplo se podrá el browser 48 como visor.

Para esto se coloca BINT0 como el valor del parámetro #indice.

Al salir del browser 48 VISOR, en la pila siempre será retornado FALSE.

Observemos que no hay ningún ítem seleccionado.





\* Browser 48 como VISOR.

\* En este caso se usa en el parámetro #indice el BINT0

\* Al salir del browser 48 VISOR, en la pila siempre será retornado FALSE

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B48Visor ( -> F )

::

CK0 ( )

'DROPFALSE ( MessageHandler )

"Browser 48 VISOR" ( MH $título )

BINT4 ( MH $título converter )

{ 11.

12.

13.

14.

15.

16.

17.

18.

19.

20. } ( MH $título converter {items} )

BINT0 ( MH $título converter {items} #índice )

ROMPTR Choose ( F )

;

Ejemplo 8 Browser 48

**Usando un Título en grob inverso y en pantalla completa.**

En este ejemplo se podrá ver el browser 48 de pantalla completa y con un título en grob inverso.

Para esto se usan los message handler 60 y 68.

\* Browser 48 con TITULO GROB INVERSO EN PANTALLA COMPLETA

\* En este caso se usan

\* Message Handler 60: Pantalla completa

\* Message Handler 68: $Titulo a grob en pantalla completa

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B48PCTitGrobInv ( -> {#} T // F )

::

CK0 ( )

' MH\_B48\_PC&TGrobInv ( MessageHandler )

"Browser 48" ( MH $título )

BINT4 ( MH $título converter )

{ 11.

12.

13.

14.

15.

16.

17.

18.

19.

20. } ( MH $título converter {items} )

BINT1 ( MH $título converter {items} #índice )

ROMPTR Choose ( ob T // F )

;

NULLNAME MH\_B48\_PC&TGrobInv

:: BINT60 #=casedrop ( Pantalla completa )

TrueTrue

BINT68 #=casedrop ( Convierte $Titulo a grob en pantalla completa )

:: 21GETLAM

TITULO->GROB131x7\_INVERSA

TRUE

;

DROPFALSE

;

\*\*\* Dada una cadena o bint la pone en un grob de

\*\*\* tamaño 131 x 7 con fondo oscuro en la pila

NULLNAME TITULO->GROB131x7\_INVERSA ( $/# -> grob131x7 )

:: ( $/# )

DUPTYPEBINT?

IT JstGETTHEMSG

( $ )

BINT1 BINT32 SUB$ ( $ ) ( corta la cadena si es mayor a 32 caract )

BINT7 BINT131 ( $ 7 131 )

MAKEGROB ( $ grob131x7\_blanco )

BINT33 ( $ grob131x7\_blanco 33 )

3PICK LEN$ ( $ grob131x7\_blanco 33 #w )

#-#2/ ( $ grob131x7\_blanco #[33-w]/2 )

Blank$ ( $ grob131x7\_blanco $' )

ROT ( grob131x7\_blanco $' $ )

&$ ( grob131x7\_blanco $'' )

$>grob ( grob131x7\_blanco grob' )

ONEONE ( grob131x7\_blanco grob' #1 #1 )

Repl ( grob131x7 )

INVGROB ( grob131x7\_inversa )

\* Lo siguiente es sólo para redondear las esquinas del grob

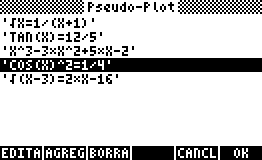
ZEROZERO PixonW ( grob131x7\_inversa )

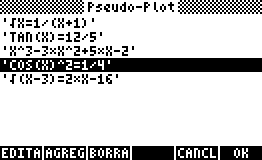
BINT130 BINT0 PixonW ( grob131x7\_inversa )

BINT0 BINT6 PixonW ( grob131x7\_inversa )

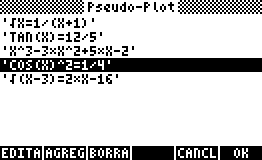
BINT130 BINT6 PixonW ( grob131x7\_inversa )

;

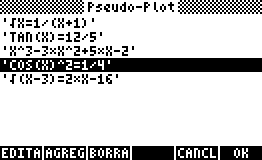
Ejemplo 9 Browser 48

Este ejemplo usa el browser 48 para permitir al usuario ingresar una lista de ecuaciones (inspirado en la ventana **Y=**, pero es considerablemente diferente). Inicialmente la lista está vacía. El usuario puede agregar ecuaciones a la lista. Las ecuaciones pueden ser también editadas, borradas o cambiadas de posición.

Este programa maneja entre otros los mensajes 62, 80 y 82.

• Mensaje 62: Debe retornar el número de ítems del browser. Es llamado al inicio del browser y también cada vez que se usa el comando **ROMPTR BBReReadNElems\_** para actualizar el número de ítems. En este programa obtiene el número de ítems a partir de los elementos de la lista de ecuaciones guardada en el lam EQS.

• Mensaje 80: Debe retornar el ítem actual. En este programa es llamado cuando se hace una búsqueda alfabética y cuando se finaliza el browser con ENTER u OK. No es llamado para mostrar los ítems en la pantalla porque se maneja el mensaje 82.

• Mensaje 82: Debe retornar el ítem actual como una cadena. En este programa es llamado para mostrar los ítems en la pantalla.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B48Ejemplo ( -> {ecs} T // F )

:: CK0 ( )

AppMode? case xKILL

( )

NULL{} ( {} )

' LAM EQS ( {} lam )

BINT1 ( {} lam #1 )

DOBIND ( ) ( CREA ENTORNO TEMPORAL )

' MH\_B48\_Ejemplo ( MessageHandler )

"Pseudo-Plot" ( MH $título )

' DO>STR ( MH $título converter )

NULL{} ( MH $título converter {} ) ( no ítems )

BINT1 ( MH $título converter {} #índice )

ROMPTR Choose ( ob T // F )

ITE

:: ( ob )

DROP ( )

LAM EQS ( {ecs} )

TRUE ( {ecs} T )

;

FALSE ( F )

( {ecs} T // F )

ABND ( {ecs} T // F ) ( destruye entorno temporal )

;

NULLNAME MH\_B48\_Ejemplo

:: BINT60 #=casedrop ( Pantalla completa )

TrueTrue

BINT62 #=casedrop ( número de elementos )

:: LAM EQS LENCOMP ( #n )

DUP#0=IT #1+ ( #n' ) ( si la lista es vacía, retorna 1 )

TRUE ( #n' T )

;

BINT80 #=casedrop ( retorna el elemento actual )

:: ( #i )

LAM EQS SWAP ( #i {ecs} )

NTHELCOMP ( ecu T // F )

NOT\_IT

"No hay ecuaciones, "

( ob )

TRUE ( ob T )

;

BINT82 #=casedrop ( retorna el elemento actual como una cadena )

:: ( #i )

LAM EQS SWAP ( #i {ecs} )

NTHELCOMP ( ecu T // F )

ITE

:: setStdWid ( ecu )

FLASHPTR FSTR7 ( $ )

;

"No hay ecuaciones, " ( $ )

TRUE ( $ T )

;

BINT83 #=casedrop ( el menú del browser 48 )

:: '

:: NoExitAction

{ { "EDITA" ( EDITA UNA ECUACIÓN )

:: LAM EQS ( {ecs} )

18GETLAM ( {ecs} #i )

NTHELCOMP ( ecuac T // F )

NOTcase ( saldrá si lista está vacía )

:: DoBadKey ( hace un sonido )

SetDAsNoCh ( no se cambiará nada en la pantalla )

;

( ecuac )

BINT1 ( ecuac #1 )

PushVStack&KeepDROP ( ecuac ) ( mueve pilaRPN sin meta a pv )

124. InitMenu% ( ecuac ) ( fija menú ALGEBRA )

FLASHPTR EQW3Edit ( ecuac' T // F ) ( edita en EQW )

BINT83 ( ... 83 )

22GETLAM ( ... 83 MessageHandler )

EVAL ( ... menu T )

DROPONE ( ... menu #1 )

StartMenu ( ecuac' T // F )

NOTcase ( sale si se canceló la edición )

PopVStack ( mueve pila virtual a pila RPN )

( ecuac' )

OBJ>R\_ ( )

PopVStack ( ) ( mueve pila virtual a pila RPN )

R>OBJ\_ ( ecuac' )

18GETLAM ( ecuac' #i )

LAM EQS ( ecuac' #i {ecs} )

PUTLIST ( {ecs}' )

' LAM EQS STO ( )

;

}

{ "AGREGA" ( CREA NUEVA ECUACIÓN )

:: PushVStack&Clear ( ) ( mueve pila RPN a pila virtual )

124. InitMenu% ( ) ( fija menú ALGEBRA )

FLASHPTR EQW3 ( ob T // F ) ( abre EQW para crear ec. )

BINT83 ( ... 83 )

22GETLAM ( ... 83 MessageHandler )

EVAL ( ... menu T )

DROPONE ( ... menu #1 )

StartMenu ( ecuac' T // F )

NOTcase ( sale si se canceló la edición )

PopVStack ( mueve pila virtual a pila RPN )

( ob )

OBJ>R\_ ( )

PopVStack ( ) ( mueve pila virtual a pila RPN )

R>OBJ\_ ( ob )

LAM EQS SWAP ( {ecs} ob )

OVER LENCOMP ( {ecs} ob #n )

18GETLAM ( {ecs} ob #n #i )

#MIN ( {ecs} ob #min[n;i] ) ( 0 si es NULL{} )

#1+ ( {ecs} ob #1+min[n;i] ) ( 1 si es NULL{} )

DUP ( {ecs} ob #1+min[n;i] #1+min[n;i] )

18PUTLAM ( {ecs} ob #1+min[n;i] )

FLASHPTR INSERT{}N ( {ecs}' ) ( fija nuevo índice )

' LAM EQS STO ( ) ( guarda nueva lista de ecuaciones )

ROMPTR BBReReadNElems\_ ( ) ( Fija nuevo nº de elementos )

ROMPTR BBReReadWidth\_ ( ) ( fija nuevo ancho de ítems )

6GETLAM #1+ 11GETLAM #MIN ( #min[offset+1;NElemMostr] )

12GETLAM #MIN ( #min[offset+1;NElemMostr;NElem] )

6PUTLAM ( ) ( fija nuevo offset )

;

}

{ "BORRA" ( BORRA ECUACIÓN )

:: LAM EQS ( {ecs} )

INNERDUP ( ob1...obn #n #n )

#0= ( ob1...obn #n flag )

casedrop ( saldrá si lista está vacía )

:: DoBadKey ( hace un sonido )

SetDAsNoCh ( no se cambiará nada en la pantalla )

;

( ob1...obn #n )

DUP

18GETLAM ( ob1...obn #n #n #i )

#- #2+ ( ob1...obn #n #n-i+2 )

ROLLDROP ( ... #n )

#1- ( ... #n-1 )

{}N ( {ecs}' )

' LAM EQS STO ( ) ( guarda nueva lista de ecuaciones )

ROMPTR BBReReadNElems\_ ( ) ( Fija nuevo nº de elementos )

18GETLAM ( #i )

12GETLAM ( #i #NElems )

#MIN ( #min[i;NElems] )

18PUTLAM ( ) ( fija nuevo índice )

ROMPTR BBReReadWidth\_ ( ) ( fija nuevo ancho de ítems )

FALSE ( F )

ROMPTR BBRecalOff&Disp\_ ( ) ( fija nuevo offset )

( sin alterar pantalla )

11GETLAM 12GETLAM ( #NElemMostrados #NElems )

#> ( flag )

ITE

:: 9GETLAM ( #Yitem1 )

12GETLAM 7GETLAM #\* #+ ( #Yitem1+NElems·HItem )

DUP ( #Y1 #Y1 )

11GETLAM 12GETLAM #- ( #Y1 #Y1 #NEMostr-NElems )

7GETLAM #\* #+ ( #Y1 #Y2 )

BLANKIT ( ) ( limpia lineas )

FLASHPTR LEDispList ( ) ( dibuja ítems )

SetDAsNoCh ( ) ( no se cambiará pantalla )

;

SetDA12a3NCh ( ) ( solo cambiará ítems en pantalla )

;

}

NullMenuKey

{ "CANCL" FLASHPTR DoCKeyCancel }

{ "OK" FLASHPTR DoCKeyOK }

{ "" ( FLECHA ARRIBA )

:: LAM EQS ( {ecs} )

18GETLAM ( {ecs} #i )

#1= ( {ecs} flag ) ( TRUE si el índice es 1 )

casedrop

:: DoBadKey ( hace un sonido )

SetDAsNoCh ( no se cambiará nada en la pantalla )

;

INNERCOMP ( ob1...obn #n )

18GETLAM ( ob1...obn #n #i )

ARRIBA\_SUBE\_UNO ( {ecs}' )

' LAM EQS STO ( )

18GETLAM #1- ( #i-1 )

18PUTLAM ( ) ( fija nuevo índice )

6GETLAM #1- ( #offset-1 )

BINT1 #MAX ( #max[offset-1;1] )

6PUTLAM ( ) ( fija nuevo #offset )

SetDA12a3NCh ( ) ( solo cambiará ítems en pantalla )

;

}

{ "" ( FLECHA ABAJO )

:: LAM EQS ( {ecs} )

18GETLAM ( {ecs} #i )

12GETLAM ( {ecs} #i #NElems )

#= ( {ecs} flag )

casedrop

:: DoBadKey ( hace un sonido )

SetDAsNoCh ( no se cambiará nada en la pantalla )

;

( {ecs} )

INNERCOMP ( ob1...obn #n )

18GETLAM ( ob1...obn #n #i )

ARRIBA\_BAJA\_UNO ( {ecs}' )

' LAM EQS STO ( )

18GETLAM #1+ ( #i-1 )

18PUTLAM ( ) ( fija nuevo índice )

6GETLAM #1+ ( #offset+1 )

11GETLAM #MIN ( #min[offset-1;NElemMostrados] )

6PUTLAM ( ) ( fija nuevo #offset )

SetDA12a3NCh ( ) ( solo cambiará ítems en pantalla )

;

}

{ "LIMPIA" ( QUITA TODAS LAS ECUACIONES )

:: LAM EQS

NULLCOMP?

case

:: DoBadKey ( hace un sonido )

SetDAsNoCh ( no se cambiará nada en la pantalla )

;

"Seguro de quitar todas las ecuac?"

AskQuestion

NOTcase

SetDAsNoCh ( no se cambiará nada en la pantalla )

NULL{}

' LAM EQS STO ( {ecs} )

ROMPTR BBReReadNElems\_ ( ) ( Fija nuevo nº de elementos )

ROMPTR BBReReadWidth\_ ( ) ( fija nuevo ancho de ítems )

ONEONE ( 1 1 )

18PUTLAM ( 1 ) ( Fija nuevo índice )

6PUTLAM ( 1 ) ( Fija nuevo offset )

BlankDA12 ( ) ( limpia áreas 1 y 2 )

FLASHPTR LEDispPrompt ( ) ( muestra título )

FLASHPTR LEDispList ( ) ( muestra ítem "no hay ecuac" )

SetDAsNoCh ( ) ( no se cambiará nada en la pantalla )

;

}

NullMenuKey

{ "CANCL" FLASHPTR DoCKeyCancel }

{ "OK" FLASHPTR DoCKeyOK }

}

;

TRUE

;

DROPFALSE

;

\* Mueve el elemento i del meta una posición hacia la izquierda

\* #i es contado desde el primer elemento del meta hacia el último

\* Finalmente convierte el meta en lista

NULLNAME ARRIBA\_SUBE\_UNO ( meta #i -> {} ) ( #i = 1,2,3...n ) ( no 1 )

:: ( meta #i )

2DUP #- #3+ ( .... #n #i #n-i+2 )

ROLL ( .... #n #i ob )

3PICK ROT#- ( .... #n ob #n-i )

#3+ ( .... #n ob #n-i+1 )

UNROLL ( meta #n )

{}N ( {} )

;

\* Mueve el elemento i del meta una posición hacia la derecha

\* #i es contado desde el primer elemento del meta hacia el último

\* Finalmente convierte el meta en lista

NULLNAME ARRIBA\_BAJA\_UNO ( meta #i -> {} ) ( #i = 1,2,3...n ) ( no n )

:: ( meta #i )

2DUP #- #3+ ( meta #n #i #n-i+2 )

ROLL ( meta #n #i ob )

3PICK ROT#- ( meta #n ob #n-i )

#1+

UNROLL ( meta #n )

{}N ( {} )

;

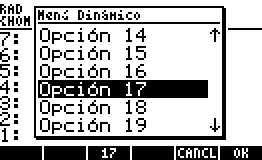
Ejemplo 10 Browser 48

**Mostrar un menú dinámico al usar el browser 48**

En el siguiente ejemplo se muestra un menú dinámico con el browser 48. Este menú muestra diferentes etiquetas de acuerdo al elemento seleccionado.

Para esto, se debe de cambiar la asignación de las teclas flecha arriba y flecha abajo en el parámetro AppKeys del POL del browser 48.

Estas asignaciones de teclas son cambiadas cuando el POL se inicia (con el mensaje 65), y después ya no son modificadas.



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B48MenúChange ( -> ob T // F )

:: CK0 ( Ningún argumento es requerido )

FALSE 1LAMBIND ( )

' MH\_B48\_MenuDinamico ( MH ) ( ES EL MESSAGE HANDLER )

"Menú Dinámico" ( MH $tit )

BINT1 ( MH $tit Converter )

{ "OpciÓn 1" "OpciÓn 2" "OpciÓn 3" "OpciÓn 4" "OpciÓn 5"

"OpciÓn 6" "OpciÓn 7" "OpciÓn 8" "OpciÓn 9" "OpciÓn 10"

"OpciÓn 11" "OpciÓn 12" "OpciÓn 13" "OpciÓn 14" "OpciÓn 15"

"OpciÓn 16" "OpciÓn 17" "OpciÓn 18" "OpciÓn 19" "OpciÓn 20"

} ( MH $tit Converter {ítems} )

BINT1 ( MH $tit Converter {ítems} #índice )

ROMPTR Choose ( ob T // F )

ABND ( )

;

\* El message handler 65 es para dibujar el marco de la ventana.

\* No es manejado aquí, pero se usa para cambiar el parámetro

\* AppKeys del POL, pues el mensaje 65 es el primero en ser llamado una vez \* que se inició el POL.

\* En cambio el mensaje 85 no puede ser utilizado para cambiar el parámetro

\* AppKeys, pues sería llamado antes de iniciarse el POL.

\* El message handler 83 es para definir el menú.

NULLNAME MH\_B48\_MenuDinamico

:: BINT65 #=casedrop ( este mensaje es para el marco )

:: 24GETLAM\_

caseFALSE

( )

TRUE 24PUTLAM\_ ( )

' :: ( #ct\_B #p\_B )

OVER ( #ct\_B #p\_B #ct\_B )

BINT10 #= ( #ct\_B #p\_B flag )

3PICK ( #ct\_B #p\_B flag #ct\_B )

BINT15 #= ( #ct\_B #p\_B flag flag' )

ORcase

:: ( #ct\_B #p\_B )

ROMPTR 0B3 008 ( prog T )

DROP ( prog )

' ROMPTR BReReadMenus\_ ( prog romptr )

' DispMenu.1 ( prog romptr DispMenu.1 )

BINT3 ( prog romptr DispMenu.1 #3 )

::N ( prog' )

TRUE ( prog' T )

;

( #ct #p ) ( para AppKeys )

;

' ROMPTR 0B3 008 ( prog romptr )

ROMPTR@ ( prog prog' T )

DROP ( prog prog' )

&COMP ( prog'' )

AppKeys! ( )

FALSE ( F )

;

BINT83 #=casedrop ( es el menú )

:: ' :: NoExitAction

{ NullMenuKey

NullMenuKey

{ :: TakeOver 18GETLAM #>$ ;

:: TakeOver

"El menú tambien puede cambiar en el browser 48"

FlashWarning

;

}

NullMenuKey

{ "CANCL" FLASHPTR DoCKeyCancel }

{ "OK" FLASHPTR DoCKeyOK }

}

;

TRUE

;

DROPFALSE

;

Ejemplo 11 Browser 48

**Cuando los elementos del browser 48 cambian**

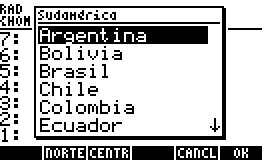
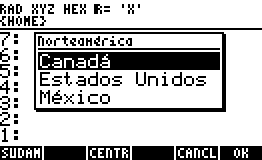
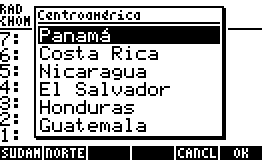
En este ejemplo los elementos del browser cambian cuando se presionan alguna teclas de menú. Los elementos actuales del browser se guardan en el lam ITEMS, el cual puede contener los elementos guardados en los lams L1, L2 o L3. El NULLNAME REFRES\_B48 se encarga de actualizar el browser 48 cada vez que se decide cambiar los elementos.

Este programa maneja los mensajes 62, 80 y 83.

• Mensaje 62: Debe retornar el número de ítems del browser. Es llamado al inicio del browser y también cada vez que se usa el comando **ROMPTR BBReReadNElems\_** para actualizar el número de ítems. En este programa obtiene el número de ítems a partir de los elementos de la lista guardada en el lam ITEMS.

• Mensaje 80: Debe retornar el ítem actual. En este programa es llamado cuando se hace una búsqueda alfabética y cuando se finaliza el browser con ENTER u OK. También es llamado para mostrar los ítems en la pantalla porque no se manejan los mensaje 81 ni 82.

• Mensaje 82: Debe retornar el ítem actual como una cadena. En este programa es llamado para mostrar los ítems en la pantalla. En este programa obtiene el ítem actual a partir de los elementos de la lista guardada en el lam ITEMS.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B48Grupos ( -> ob T // F )

::

CK0 ( ) ( Ningún argumento es requerido )

{ "Argentina" "Bolivia" "Brasil" "Chile" "Colombia"

"Ecuador" "Paraguay" "Perú" "Uruguay" "Venezuela" }

{ "Canadá" "Estados Unidos" "México" }

{ "Panamá" "Costa Rica" "Nicaragua" "El Salvador" "Honduras" "Guatemala" }

{ "Sudamérica" "Norteamérica" "Centroamérica" }

BINT1 ( ... {$} #1 )

5PICK ( ... {$} #1 {$} )

{ LAM L1 LAM L2 LAM L3 LAM LTITULOS

LAM #G LAM ITEMS }

BIND ( )

' MH\_B48\_Ejemplo11 ( MH )

LAM LTITULOS LAM #G NTHCOMPDROP ( MH $título )

'NOP ( MH $título converter )

NULL{} ( MH $título converter items )

BINT1 ( MH $título converter items #índice )

ROMPTR Choose ( ob T // F )

ABND ( )

;

NULLNAME MH\_B48\_Ejemplo11

:: BINT62 #=casedrop ( proporciona el número de elementos )

:: ( )

LAM ITEMS ( {} )

LENCOMP ( #n )

TRUE ( # T )

;

BINT80 #=casedrop ( retorna el elemento correspondiente al índice )

:: ( # )

LAM ITEMS ( # lam )

SWAP ( {} # )

NTHCOMPDROP ( $ )

TRUE ( $ T )

;

BINT83 #=casedrop ( retorna el menú )

:: { { :: TakeOver LAM #G #1= case NULL$ "SUDAM" ;

:: TakeOver LAM #G #1= case DoBadKey BINT1 ' LAM #G STO REFRES\_B48 ;

}

{ :: TakeOver LAM #G #2= case NULL$ "NORTE" ;

:: TakeOver LAM #G #2= case DoBadKey BINT2 ' LAM #G STO REFRES\_B48 ;

}

{ :: TakeOver LAM #G #3= case NULL$ "CENTRO" ;

:: TakeOver LAM #G #3= case DoBadKey BINT3 ' LAM #G STO REFRES\_B48 ;

}

NullMenuKey

{ "CANCL" FLASHPTR DoCKeyCancel }

{ "OK" FLASHPTR DoCKeyOK }

}

TRUE

;

DROPFALSE

;

NULLNAME REFRES\_B48 ( -> )

::

{ LAM L1 LAM L2 LAM L3 } ( {lams} )

LAM #G ( {lams} #g )

NTHCOMPDROP ( lam )

@LAM ( {} T )

DROP ( {} )

' LAM ITEMS STO ( ) ( fija lam de ITEMS )

LAM LTITULOS ( {$} )

LAM #G ( {$} # )

NTHCOMPDROP ( $título )

21PUTLAM ( ) ( fija nuevo título )

ROMPTR 0B3 03A ( ) ( RESTAURA LA PANTALLA A SU ESTADO ANTERIOR AL B48 )

ROMPTR BBReReadNElems\_ ( ) ( Relee el nº total de elementos )

ROMPTR BBReReadPageSize\_ ( ) ( Relee el nº de elementos mostrados )

ROMPTR BBReReadWidth\_ ( ) ( Relee ancho de elementos en píxeles )

ROMPTR BBReReadCoords\_ ( ) ( Relee coord. esquina superior izq. )

ROMPTR 0B3 039 ( ) ( GUARDA PANTALLA DEL ESTADO ANTERIOR AL B48 )

18GETLAM ( #Indice )

12GETLAM ( #Indice #NumElementos )

DUP#0=IT

DROPONE

( #Indice #NumElementos' )

#MIN ( min[#Indice,#NumElementos] )

18PUTLAM ( )

FALSE ( F )

ROMPTR BBRecalOff&Disp\_ ( ) ( Recalcula offset )

;

Capítulo 36  
Browser 224

El browser 224 es un browser que siempre se ejecuta a pantalla completa y con la fuente de sistema. La altura de cada ítem siempre será de 8 píxeles. No es posible la selección múltiple ni usarlo como visor.

Este browser es usado por la calculadora para mostrar los resultados en el entorno MSOLVR (Multiple Solver Equation) y también en la biblioteca de ecuaciones.

El comando que llama al browser 224 es **ROMPTR Brbrowse**. Este comando requiere de ocho argumentos:

Menú

$Título

{ AcciónENTER AcciónON }

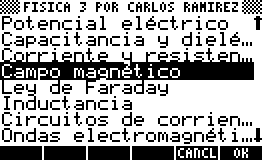
#IndiceSuperior

#IndiceInicial

Items

::Converter

{#letras}

El comando **ROMPTR Brbrowse** puede retornar cualquier objeto en la pila (o nada) de acuerdo a la acción que le asignemos a las teclas ENTER, OK o a las teclas de menú.

El comando **ROMPTR Brbrowse** llama internamente a un POL (como los otros browsers), pero no guarda ni restaura el entorno anterior al POL. Por eso debemos llamar al browser 224 de la siguiente manera:

::

POLSaveUI ( Guarda la interfaz actual)

ERRSET

::

Menú

$Título

{ AcciónENTER AcciónON }

#IndiceSuperior

#IndiceInicial

Items

::Converter

{#}

ROMPTR BRbrowse

;

ERRTRAP

POLResUI&Err ( si un error ocurre, restaura la interfaz guardada )

( por POLSaveUI y luego llama al error que ocurrió )

POLRestoreUI ( restaura la interfaz guardada por POLSaveUI )

;

36.1 El Parámetro Menú

Este es una lista o programa que retorna una lista.

Este menú puede tener varias páginas pero no se pueden asignar acciones a las combinaciones shift izquierdo/derecho + tecla de menú.

36.2 El Parámetro $Título

Este es el título que se mostrará en la parte superior del browser y debe ser una cadena. También puedes colocar una cadena vacía y de esta manera será visible un ítem adicional en la pantalla.

36.3 El Parámetro AccionesENTER&ON

Este debe ser una lista de dos elementos. El primer elemento será la asignación para la tecla ENTER y el segundo debe ser la asignación para la tecla ON.

Si deseas que la tecla ENTER retorne el elemento actual y TRUE y finalice el browser, y que la tecla ON retorne FALSE y finalice el browser (como en los otros browsers) puedes fijar este parámetro de la siguiente manera (cuando el parámetro **Items** es una lista):

{ :: ROMPTR BRRclC1 ( {ITEMS} ) ( llama a lista de ítems )

ROMPTR BRRclCurRow ( {ITEMS} #i ) ( llama al índice actual )

NTHCOMPDROP ( ITEMi ) ( llama al ítem actual )

FLASHPTR BRdone ( ITEMi ) ( el browser finalizará )

TRUE ( ITEMi T )

;

:: FLASHPTR BRdone ( ) ( el browser finalizará )

FALSE ( F )

;

}

Cuando el parámetro **Items** es un arreglo:

{ :: ROMPTR BRRclCurRow ( #i ) ( llama al índice actual )

ROMPTR BRRclC1 ( #i [] ) ( llama a arreglo de ítems )

GETATELN ( ítem T )

FLASHPTR BRdone ( ítem T ) ( el browser finalizará )

;

:: FLASHPTR BRdone ( ) ( el browser finalizará )

FALSE ( F )

;

}

36.4 El Parámetro #IndiceSuperior

Este es un bint y representa el índice del ítem que se mostrará en la parte superior de la pantalla (debajo del título) cuando se inicie el browser.

36.5 El Parámetro #IndiceInicial

Este es un bint y representa el índice del ítem inicialmente seleccionado. Debe ser mayor o igual que **#IndiceSuperior** y debe ser un bint válido.

36.6 El Parámetro Items

Este es un objeto único que contiene a los ítems del browser.

El valor más común para este parámetro es una lista, aunque también puede ser un objeto de otro tipo.

36.7 El Parámetro ::Converter

Este debe ser un programa que convierta un ítem a cadena para que pueda ser mostrado en la pantalla.

El diagrama de pila de este programa debe ser:

Items #índice 🡺 $item

Items #0 🡺 #NElementos

Observa que si este programa es llamado con el bint cero en la pila, el converter debe devolver el número de ítems del browser.

|  |  |
| --- | --- |
| Parámetro **Items** | Parámetro **::Converter** |
| Lista de cadenas de la forma:  { $item1  $item2  …  $itemn  } | :: DUP ( {$ITEMS} #i/#0 #i/#0 )  #0=case  :: ( {$ITEMS} #0 )  DROP ( {$ITEMS} )  LENCOMP ( #n )  ;  ( {$ITEMS} #i )  NTHCOMPDROP ( $itemi )  ; |
| Lista de listas de la forma:  { {$item1 ob1}  {$item2 ob2}  ...  {$itemn obn}  } | :: DUP ( {ITEMS} #i/#0 #i/#0 )  #0=case  :: ( {ITEMS} #0 )  DROP ( {ITEMS} )  LENCOMP ( #n )  ;  ( {ITEMS} #i )  NTHCOMPDROP ( {$ ob} )  CARCOMP ( $ )  ; |
| Arreglo de cadenas de la forma:  [ $item1  $item2  ...  $itemn  ] | :: DUP ( [$] #i/#0 #i/#0 )  #0=case  :: ( [$] #0 )  DROP ( [$] )  ARSIZE\_ ( #n )  ;  ( [$] #i )  SWAP ( #i [$] )  GETATELN ( $ítemi T )  DROP ( $itemi )  ; |

36.8 El Parámetro {#letras}

Este parámetro es una lista con 26 bints o una lista vacía. Actualmente no está muy bien comprendido. Si usas una lista vacía el browser funcionará correctamente.

36.9 Los LAMs del Browser 224

El browser 224 usa internamente 16 lams, de tal manera que si tu quieres llamar a otros lams que tu hayas creado anteriormente, es preferible que los hayas creado como lams con nombre. Los LAMs del browser 224 son los siguientes:

**LAM Nombre Contenido**

1LAM **'BR26** Un bint. Es el número de elementos mostrados.

2LAM **'BR25** Un carácter. Representa a la última letra de la búsqueda alfabética.

Si todavía no se ha hecho una búsqueda alfabético, es el bint cero.

3LAM **'BR24** Un bint o un carácter. Si la última acción realizada fue una búsqueda

alfabética y ha tenido éxito, es el carácter correspondiente.

4LAM **'BR20** Un flag. Puedes fijar su valor como TRUE para que finalice el POL.

5LAM **'BR16** Un bint. Es la coordenada Y del primer ítem mostrado en la pantalla.

6LAM **'BR12** Un bint. Es la coordenada Y de la parte inferior del ítem actual.

7LAM **'BR14** Un bint. Es la coordenada Y de la parte superior del ítem actual.

8LAM **'BR6** Un bint. Es el número de ítems del browser.

9LAM **'BR2** Un bint. Es el índice del último ítem mostrado actualmente en la pantalla.

10LAM **'BR28** Un bint. Es el número de elementos del menú.

11LAM **'BR27** Una lista. Es el parámetro **{#letras}**.

12LAM **'BR22** Un programa. Es el parámetro **::Converter**.

13LAM **'BR5** Es el parámetro **Items**.

14LAM **'BR3** Un bint. Es el índice del ítem actualmente seleccionado.

15LAM **'BR1** Un bint. Es el índice del primer ítem mostrado actualmente en la pantalla.

16LAM **'BR21** Una lista con dos elementos. Es el parámetro **AccionesENTER&ON**

36.10 Atajos de Tecla del Browser 224

Algunos atajos de tecla predefinidos del browser 224 son:

NEXT Muestra la siguiente página del menú.

Shift izquierdo + NEXT Muestra la anterior página del menú.

Shift derecho + NEXT Muestra la primera página del menú.

Shift izquierdo + ENTER Muestra el ítem actual si este es muy largo. Equivale a usar el comando **^BRViewItem**.

Shift derecho + ENTER Muestra el ítem actual si este es muy largo. Equivale a usar el comando **^BRViewItem**.

Shift derecho + ON Apaga la calculadora.

ALPHA Activa búsqueda alfabética.

ALPHA + ALPHA Desactiva búsqueda alfabética si estaba activada.

36.11 Referencia

**Direcc. Nombre Descripción**

0100E0 ˜BRbrowse ( menu $ {}ENT&ON #sup #i ítems ::Convert {} 🡺 ? )

Llama al browser 224.

$ es el **Título**.

0130E0 ˜BRoutput ( 🡺 #superior #índice )

Retorna el índice del primer ítem mostrado actualmente y el índice del ítem seleccionado actualmente.

0180E0 ˜BRRclCurRow ( 🡺 #índice )

Llama al índice actual.

0190E0 ˜BRRclC1 ( 🡺 ítems )

Llama al parámetro **ítems**.

0030E0 ˜BRStoC1 ( ítems 🡺 )

Guarda un nuevo valor para el parámetro **ítems**.

0A4003 ^BRdone ( 🡺 )

Fija el lam **‘BR20** como TRUE para que el browser termine.

0AB003 ^BRGetItem ( #índice 🡺 $ )

( #0 🡺 #n )

Retorna el ítem actual como una cadena, evaluando el parámetro **::Converter**.

Si el bint cero está en la pila, retorna el número de ítems.

0A6003 ^BRinverse ( 🡺 )

Invierte los píxeles del ítem actual.

0A5003 ^BRDispItems ( 🡺 )

Muestra los ítems en la pantalla.

0A7003 ^BRViewItem ( 🡺 )

Si el item actual es muy largo, este es mostrado en la pantalla en varias líneas.

Para regresar a la pantalla normal del browser, debes presiona ENTER u ON.

03A0E0 ROMPTR 0E0 03A ( 🡺 ? )

Ejecuta la acción de la tecla ON fijada en el parámetro **AccionesENTER&ON**.

0390E0 ROMPTR 0E0 039 ( 🡺 ? )

Ejecuta la acción de la tecla ENTER fijada en el parámetro **AccionesENTER&ON**.

36.12 Ejemplos Browser224

Ejemplo 1 Browser 224

**Cuando los ítems están en un arreglo de cadenas**

En este ejemplo los elementos del browser están en un arreglo de cadenas.

La acción de la tecla ENTER será retornar el ítem actual en la pila seguido de TRUE y finalizar el browser.

La acción de la tecla ON será finalizar el browser dejando FALSE en la pila.

Las teclas de menú F5 y F6 realizarán las mismas acciones que ON y ENTER respectivamente.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B224Ej1 ( -> $ T // F )

:: CK0 ( ) ( ningún argumento es requerido )

POLSaveUI

ERRSET

::

{ NullMenuKey

NullMenuKey

NullMenuKey

NullMenuKey

{ "CANCL" ROMPTR 0E0 03A }

{ "OK" ROMPTR 0E0 039 }

} ( menú )

"TITULO" ( menú $título )

{ :: ROMPTR BRRclCurRow ( #i ) ( llama al índice actual )

ROMPTR BRRclC1 ( #i [$] ) ( llama a arreglo de cadenas )

GETATELN ( $ítem T ) ( obtiene elemento del arreglo )

FLASHPTR BRdone ( $ítem T ) ( el browser finalizará )

;

:: FLASHPTR BRdone ( ) ( el browser finalizará )

FALSE ( F )

;

}

BINT3 ( ... #sup )

BINT7 ( ... #inicial )

ARRY [ "Panamá" "Costa Rica" "Nicaragua" "El Salvador" "Honduras"

"Guatemala" "Cuba" "Haití" "República Dominicana" "Puerto Rico"

"Jamaica" "Bahamas" ]

' :: ( [$] #i/#0 )

DUP#0=csedrp

ARSIZE\_ ( #n ) ( sale con #n )

( [$] #i )

SWAP ( #i [$] )

GETATELN ( $ítemi T )

DROP ( $itemi )

;

{ } ( ... ::Converter {} )

ROMPTR BRbrowse ( ob T // F )

;

ERRTRAP

POLResUI&Err

POLRestoreUI

;

Ejemplo 2 Browser 224

**Cuando los ítems están en una lista de cadenas**

En este ejemplo los elementos del browser están en una lista de cadenas.

La acción de la tecla ENTER será retornar el ítem actual en la pila seguido de TRUE y finalizar el browser.

La acción de la tecla ON será finalizar el browser dejando FALSE en la pila.

Las teclas de menú F5 y F6 realizarán las mismas acciones que ON y ENTER respectivamente.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B224Ej2 ( -> $ T // F )

::

CK0 ( ) ( ningún argumento es requerido )

POLSaveUI

ERRSET

::

{ NullMenuKey

NullMenuKey

NullMenuKey

NullMenuKey

{ "CANCL" ROMPTR 0E0 03A }

{ "OK" ROMPTR 0E0 039 }

} ( menú )

"TITULO" ( menú $título )

{ :: ROMPTR BRRclC1 ( {$} ) ( llama a lista de cadenas )

ROMPTR BRRclCurRow ( {$} #i ) ( llama al índice actual )

NTHELCOMP ( $ítem T ) ( obtiene elemento de la lista )

FLASHPTR BRdone ( $ítem T ) ( el browser finalizará )

;

:: FLASHPTR BRdone ( ) ( el browser finalizará )

FALSE ( F )

;

}

BINT3 ( ... #sup )

BINT7 ( ... #inicial )

{ "Panamá" "Costa Rica" "Nicaragua" "El Salvador" "Honduras" "Guatemala"

"Cuba" "Haití" "República Dominicana" "Puerto Rico" "Jamaica" "Bahamas" }

' :: ( [$] #i/#0 )

DUP#0=csedrp

LENCOMP ( #n ) ( sale con #n )

( [$] #i )

NTHCOMPDROP ( $ítemi )

;

{ } ( ... ::Converter {} )

ROMPTR BRbrowse ( ob T // F )

;

ERRTRAP

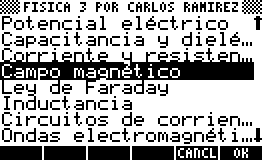
POLResUI&Err

POLRestoreUI

;

Ejemplo 3 Browser 224

**Cuando los ítems están en una lista de listas de la forma {$ ob}**

Si el parámetro Items ges una lista de listas de la forma:

{ {$item1 ob1} {$item2 ob2}... {$itemn obn} }

La acción de la tecla ENTER será retornar el ítem actual en la pila seguido de TRUE y finalizar el browser.

La acción de la tecla ON será finalizar el browser dejando FALSE en la pila.

Las teclas de menú F5 y F6 realizarán las mismas acciones que ON y ENTER respectivamente.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME B224Ej3 ( -> {$ #} T // F )

:: CK0 ( ) ( ningún argumento es requerido )

POLSaveUI

ERRSET

::

{ NullMenuKey

NullMenuKey

NullMenuKey

NullMenuKey

{ "CANCL" ROMPTR 0E0 03A }

{ "OK" ROMPTR 0E0 039 } } ( menú )

"FÍSICA 3 POR CARLOS RAMÍREZ" ( menú $título )

{ :: ROMPTR BRRclC1 ( {$} ) ( llama a lista de cadenas )

ROMPTR BRRclCurRow ( {$} #i ) ( llama al índice actual )

NTHELCOMP ( $ítem T ) ( obtiene elemento de la lista )

FLASHPTR BRdone ( $ítem T ) ( el browser finalizará )

;

:: FLASHPTR BRdone ( ) ( el browser finalizará )

FALSE ( F )

;

}

BINT2 ( ... #sup )

BINT7 ( ... #inicial )

{ { "Campo eléctrico" BINT1 }

{ "Potencial eléctrico" BINT2 }

{ "Capacitancia y dieléctricos" BINT3 }

{ "Corriente y resistencia" BINT4 }

{ "Campo magnético" BINT5 }

{ "Ley de Faraday" BINT6 }

{ "Inductancia" BINT7 }

{ "Circuitos de corriente alterna" BINT8 }

{ "Ondas electromagnéticas" BINT9 }

{ "Naturaleza de la luz" BINT10 } }

' :: ( [$] #i/#0 )

DUP#0=csedrp

LENCOMP ( #n ) ( sale con #n )

( [$] #i )

NTHCOMPDROP ( $ítemi )

CARCOMP

;

{ } ( ... ::Converter {} )

ROMPTR BRbrowse ( ob T // F )

;

ERRTRAP

POLResUI&Err

POLRestoreUI

;

Capítulo 37  
Formularios de Entrada con IfMain

Los formularios de entrada proveen una interfaz gráfica para ingresar datos que sean requeridos por un programa. Los datos son ingresados por medio de varios campos, los cuales son “espacios” que el usuario puede llenar con los datos apropiados. Los formularios de entrada son usados en muchos lugares en las calculadoras HP. Puedes ver uno de estos presionando la tecla MODE. Es posible crear formularios de entrada en User RPL con el comando **I**NFO**RM**, pero esta no es una de las tareas más fáciles. En System RPL es todavía más dificil. Pero hay varias ventajas: en User RPL, sólo puedes tener campos de texto, en System RPL puedes también tener cuadros de selección y casillas de verificación. También puedes restringir las entradas válidas, y hacer aparecer o desaparecer los campos durante la ejecución. Finalmente, en System RPL, los formularios de entrada son considerablemente más rápidos.

Los formularios de entrada son creados con el comando **ˆIfMain**, el cual es un flashpointer. Este necesita muchos argumentos. Estos son divididos en tres categorías:

* Definición de etiquetas
* Definición de campos
* Información general

Cada definición de etiqueta y cada definición de campo está compuesta de varios argumentos.

El comando **ˆIfMain** es el generador de formularios de entrada presente a partir de la HP49. La tabla de abajo muestra la estructura general de los argumentos a ingresar para el comando **ˆIfMain**:

**Parámetro Descripción**

**etiqueta\_1**

. . . Definiciones de las etiquetas

**et**ique**ta\_n**

**campo\_1**

. . . Definiciones de los campos

**campo\_n**

**#etiquetas** Número de etiquetas

**#campos** Número de campos

**MessageHandlerIfMain**  Ver sección 37.4 abajo

**Titulo**  Título a ser mostrado en la parte alta de la pantalla.

37.1 Definiciones de Etiquetas

Cada definición de una etiqueta consiste de tres argumentos:

**Parametro Descripción**

**cuerpo\_etiqueta** Cadena o grob a ser mostrado.

**#x\_posición** Coordenada X

**#y\_posición** Coordenada Y

**cuerpo\_etiqueta** es una cadena o un grob. Si pones una cadena, esta sera convertida a grob usando la minifuente. También puede ser un bint, en este caso se mostrará el mensaje de error (ver Apéndice E) correspondiente a ese bint.

La etiqueta sera mostrada en las coordenadas especificadas. Estas coordenadas son dos bints que representan las posiciones x e y de la etiqueta en la pantalla. La esquina superior izquierda tiene coordenadas (0, 0) y las coordenadas x e y se incrementan hacia la derecha y hacia abajo respectivamente.

37.2 Definiciones de Campos

Cada definición de un campo consiste de trece argumentos:

**Parametro Descripción**

**MessageHandlerCampo** Ver sección 37.4 abajo.

**#x\_posición** Coordenada X

**#y\_posición** Coordenada Y (normalmente coord Y de etiqueta menos 1)

**#w\_ancho** Ancho del campo.

**#h\_altura** Altura del campo (usualmente 8 ó 6).

**#TipoDeCampo** Tipo de campo, ver abajo los valores válidos.

**TiposPermitidos** Lista con los tipos de objetos válidos para este campo.

**Decompile** Ver abajo.

**"Ayuda"** Cadena de ayuda para este campo.

**ChooseData** Ver abajo.

**ChooseDecompile** Ver abajo.

**ValorReset** Valor reset de este campo.

**ValorInicial** Valor inicial de este campo.

Los **Message Handler de los campos** serán descritos abajo.

Las **posiciones #x** e **#y** especifican donde aparecerán los campos. Funcionan de manera similar a las posiciones X e Y en las definiciones de etiquetas.

Las dimensiones **#w** y **#h** son también dos bints, los cuales especifican el tamaño del campo. Por ejemplo, campos que muestran objetos con la minifuente pueden tener una altura de 6. Campos que muestran objetos con fuente de tamaño 8 pueden tener altura 8. En campos **CHECKBOX** tanto **#w** como **#h** deben tener como su valor a MINUSONE.

El parámetro **#TipoDeCampo** es un bint que define el tipo del campo.

**Valor decimal Tipo de campo**

BINT1 **TEXTO**: el usuario puede ingresar cualquier cosa.

BINT12 **CHOOSE**: el usuario debe seleccionar de una lista de valores válidos.

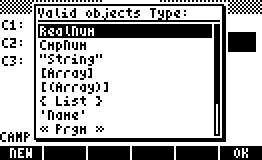
BINT2 **COMBOCHOOSE**: el usuario puede seleccionar de una lista de valores o ingresar otro.

BINT32 **CHECKBOX**: es una casilla de verificación que el usuario puede marcar o desmarcar.

El parámetro **TiposPermitidos** es usado en los campos **TEXTO** y **COMBOCHOOSE**. Es una lista de bints, que representan los tipos permitidos de los objetos que pueden ser ingresados en ese campo. Sólo son permitidos objetos de los tipos mostrados en la siguiente tabla (son los mismos tipos que acepta el comando **xINFORM** de User RPL, pero en lugar de números reales estos tipos se indican con bints).

|  |  |
| --- | --- |
| **BINT** | **TIPOS DE OBJETOS** |
| BINT0 | Reales |
| BINT1 | Complejos |
| BINT2 | Cadenas |
| BINT3 | Arreglos reales y matrices simbólicas |
| BINT4 | Arreglos no reales |
| BINT5 | Listas |
| BINT6 | Nombres globales |
| BINT8 | Programas |
| BINT9 | Simbólicos |
| BINT10 | Cadenas hexadecimales |
| BINT13 | Unidades |
| # FF | Enteros |

Para los campos **CHOOSE** y **CHECKBOX** el parámetro **TiposPermitidos** debe ser MINUSONE. También puedes especificar MINUSONE para campos **TEXTO** y **COMBOCHOOSE**, dando como resultado que serán aceptados los objetos de todos los tipos mostrados en la tabla anterior.



El parámetro **#Decompile** es un bint que especifica como serán mostrados en la pantalla los objetos ingresados en el campo. Es un bint situado entre BINT0 y BINT47.

**Valor**

**decimal Descripción**

BINT0 No hace descompilación. Usar BINT0, sólo cuando el campo tiene cadenas o bints.

Las cadenas se mostrarán sin comillas dobles y con fuente de sistema.

BINT1 Muestra el objeto usando la minifuente.

BINT2 Muestra los números usando el modo actual (STD, FIX, SCI, ENG).

Las cadenas se mostrarán con comillas dobles y los ids con comillas.

Los objetos unidad serán mostrados sin comillas.

BINT4 Muestra los números usando siempre el modo estándar (STD).

Las cadenas se mostrarán con comillas dobles y los ids con comillas.

Los objetos unidad serán mostrados con comillas.

BINT8 Muestra solamente el primer carácter del objeto.

Puedes ingresar otro bint que sea la suma de algunos de los indicados.

Por ejemplo, si ingresas BINT5 (4+1), el objeto se mostrará usando el modo estándar y con minifuente.

Si el objeto contenido en el campo es una lista o cualquier compuesto, puedes agregar:

**Valor**

**decimal Descripción**

BINT16 Muestra sólo el primer objeto del compuesto.

BINT32 Muestra sólo el segundo objeto del compuesto.

Esta opción puede ser útil cuando uses un campo **CHOOSE**.

Por ejemplo, si ingresas BINT41 (32+8+1), se mostrará el segundo objeto del compuesto (sólo puede ser cadena o bint porque no tiene a 2 o a 4 como uno de sus sumandos), solamente su primer carácter y con fuente pequeña.

Otro ejemplo, si ingresas BINT18 (16+2), se mostrará el primer objeto del compuesto, usando el modo numérico actual y con fuente de tamaño normal (pues no está el 1).

OBSERVACIONES:

También puedes especificar el parámetro **#Decompile** como BINT0. En este caso, ninguna descompilación es hecha y sólo puedes usar cadenas y bints (se mostrará la cadena correspondiente al mensaje de error de dicho bint). En ambos casos, la cadena se mostrará sin las comillas dobles y con fuente de tamaño normal.

El parámetro **#Decompile** debe contener necesariamente como uno de sus sumandos a BINT2 o a BINT4, excepto cuando el objeto que se mostrará es una cadena o un bint.

El valor del parámetro **#Decompile** no tiene ningún significado en un campo **CHECKBOX**.

El parámetro **"Ayuda"** especifica la cadena que sera mostrada en la parte inferior de la pantalla cuando ese campo tenga el enfoque. Puedes escribir cualquier cadena.

El parámetro **ChooseData** es usado solamente en campos **CHOOSE** y **COMBOCHOOSE**. Para otros campos debes poner MINUSONE como el valor de este parámetro. Este parámetro es la lista de valores que serán presentados al usuario para que este seleccione.

Cuando usas un **#Decompile** que contiene al valor 0 o 1 (fuente normal o fuente pequeña), puedes usar un ChooseData como este:

{ "cadena1" "cadena2" ... }

En cambio, si usas un **#Decompile** que contiene al valor 16 o 17 (fuente normal o fuente pequeña), puedes usar un **ChooseData** como este:

{ { "cadena1" <cuerpo1> } { "cadena2" <cuerpo2> } { ... } ... }

De esta manera, solo el primer objeto de cada lista sera mostrado, pero la lista completa sera retornada (como al usar el comando **CHOOSE** de User RPL).

Hasta el momento, el parámetro **ChooseDecompile** es ignorado al usar el comando **ˆIfMain**. Puedes usar el mismo valor que en **#Decompile**, o cualquier otro. Se puede usar el espacio de los ChooseDecompile de cada campo en IfMain para guardar objetos ahí de la siguiente manera (posiciones 5, 13, 21, 29, 37, 45, etc):

Para poner un objeto en ChooseDecompile del campo 0

BINT5 PutElemTopVStack ( ob --> )

Para conseguir el objeto que está en ChooseDecompile del campo 0

BINT5 GetElemTopVStack ( --> ob )

Para poner un objeto en ChooseDecompile del campo 1

BINT13 PutElemTopVStack ( ob --> )

Para conseguir el objeto que está en ChooseDecompile del campo 1

BINT13 GetElemTopVStack ( --> ob )

Para poner un objeto en ChooseDecompile del campo 2

BINT21 PutElemTopVStack ( ob --> )

Para conseguir el objeto que está en ChooseDecompile del campo 2

BINT21 GetElemTopVStack ( --> ob )

Los parámetros **valorReset** y **ValorInicial** se refieren al valor del campo cuando es mostrado inicialmente y cuando es reseteado. Ambos deben ser objetos de algún tipo permitido para este campo. Para un campo **CHOOSE** deben ser alguno de los elementos de la lista **ChooseData**. Para un campo **CHECKBOX**, usa TRUE o FALSE. Puedes dejar un campo **TEXTO** o **COMBOCHOOSE** en blanco, especificando MINUSONE como uno o ambos de estos parámetros.

37.3 Número de Etiquetas y de Campos

Estos son dos bints que representan el número de etiquetas y de campos del formulario de entrada. Observa que estos dos valores pueden ser diferentes, de manera que puedes tener etiquetas que sólo muestren algún tipo de información al usuario, o campos sin ninguna etiqueta asociada. El número de etiquetas puede ser cero, pero el número de campos debe ser mayor que cero.

37.4 Message Handler

Como otras aplicaciones de la calculadora HP, los formularios de entrada también usan message handlers para permitir al programador un mayor control. Hay un message handler para cada campo, y también un message handler para el formulario de entrada. Los message handler son llamados cuando algo “interesante” ocurre en el campo o en el formulario, y durante el inicio del formulario de entrada. Como sucede con otros message handlers, el programa proporcionado es llamado con un número (un bint) en el nivel uno, y a veces otros parámetros.

Si el programa message handler maneja al mensaje correspondiente al bint proporcionado, entonces debería retornar lo que sea que pida ese mensaje (a veces nada).

Si el programa message handler no maneja al mensaje que corresponde al bint proporcionado, entonces el programa debe borrar ese bint de la pila y poner FALSE, dejando los argumentos restantes (si los hubiera) en su lugar.

Por lo tanto un programa message handler que no maneja ningún mensaje es simplemente **DROPFALSE**, el cual, como sabemos, puede ser puesto en la pila con el comando **'DROPFALSE**.

En el cuerpo de un message handler, los comandos listados en la sección de referencia de abajo pueden ser usados para conseguir información desde el formulario de entrada o para modificar este. La sección 37.8.2 describirá cada uno de los message handler disponibles en ˆIfMain.

Esta es una plantilla para un programa message handler con tres mensajes:

' :: BINT16 #=casedrop ( IfMsgOK ) ( puede ser cualquier otro mensaje )

::

\* Código para el mensaje. No olvidar retornar TRUE.

;

BINT18 #=casedrop ( IfMsgType ) ( puede ser cualquier otro )

::

\* Código para el mensaje.

;

BINT22 #=casedrop ( IfMsgCommandLineValid ) ( puede ser otro )

::

\* Código para el mensaje.

;

\* Y posiblemente más mensajes.

DROPFALSE ( indica que otros mensajes no serán llamados )

;

Los mensajes en **^Ifmain** van desde el 0 hasta el 26.

37.5 El Título del Formulario de Entrada

El título del formulario de entrada sera mostrado en la parte superior de la pantalla. Puede ser una cadena, un grob o un bint.

• Si es una cadena esta será mostrada con minifuente. Esta cadena debe tener como máximo 32 caracteres (el ancho de la pantalla). Si tiene más caracteres, no se podrá ver el título.

• Si es un grob, este debe tener el tamaño exacto de 131x7 pixeles.

• Si es un bint, será mostrado el mensaje de error correspondiente a ese bint.

37.6 Resultados del Formulario de Entrada

La pila de salida, si el usuario salió del formulario de entrada con la tecla ENTER es:

N+1: Valor\_Campo\_1 (valor externo)

N: Valor\_Campo\_2 (valor externo)

. . . . .

2: Valor\_Campo\_n (valor externo)

1: TRUE

Si CANCEL fue usado para salir del formulario de entrada, entonces sólo FALSE es retornado en la pila.

Si un campo está vacío, entonces xNOVAL es retornado como el valor de este campo.

37.7 Los LAMs del IfMain

El generador de formularios de entrada **^I**fMa**in** usa internamente 8 lams, de tal manera que si tu quieres llamar a otros lams que tu hayas creado anteriormente, es preferible que los hayas creado como lams con nombre. Los LAMs del **^IfMain** son los siguientes:

**LAM Nombre Contenido**

1LAM **'CmdLine** Un flag. Su valor es TRUE si existe una línea de comandos activa.

2LAM **'IfMenu** Es el menú del formulario de entrada.

3LAM **'Depth** Un bint. Es el número de objetos presentes en la pila antes del

formulario de entrada (sin contar sus argumentos).

4LAM **'CurrentField** Un bint. Representa el número del campo actual.

Para el primer campo, este valor es cero.

Cuidado: Antes de iniciarse el POL del formulario de entrada, algunos mensajes (4,5,6,7 y 26) son llamados. En ese momento, este lam todavía no indica el valor del campo, pues su valor es MINUSONE. Debes tener en cuenta esto si llamas a este lam cuando programas en esos mensajes.

5LAM **'Quit** Un flag. Puedes fijar su valor como TRUE para que finalice el POL.

6LAM **'FieldHandler** Message handler del campo actual.

7LAM **'IfHandler** Message handler del formulario.

8LAM **'StringData** Una cadena.

37.8 Referencia

**Direcc. Nombre Descripción**

020004 ˆIfMain ( l1..ln f1..fm #n #m msg $ 🡺 ob1..obn T )

( l1..ln f1..fm #n #m msg $ 🡺 F )

l = $ #x #y

f = MsgH #x #y #w #h #TipoCampo TipoObjetos #dec

$ayuda ChData #ChDec ValorReset ValorInicial

Inicia un formulario de entrada.

021004 ˆIfSetFieldVisible ( # T/F(camp/etiq) T/F(val) 🡺 )

( # T/F(camp/etiq) #0 🡺 T/F(val) )

Vuelve a un campo o una etiqueta, visible o invisible. El segundo argumento especifica si # es número de campo o de etiqueta. El tercer argumento es para volverlo visible o invisible. Si el tercer argumento es ZERO, entonces el comando devuelve en la pila un flag que indica si el elemento especificado es visible o invisible ahora.

022004 ˆIfSetSelected ( # T/F(camp/etiq) T/F(val) 🡺 )

( # T/F(camp/etiq) #0 🡺 T/F(val) )

Vuelve a un campo o una etiqueta, seleccionada o no seleccionada (si es seleccionada aparecerá en video inverso en la pantalla).

023004 ˆIfSetGrob ( # T/F(camp/etiq) grb 🡺 )

Establece el grob a mostrarse para el campo o la etiqueta (modifica los datos guardados en data string).

03E004 ˆIfSetGrob3 ( #$dat # T/F(camp/etiq) grb/$/# 🡺 )

Establece el grob a mostrarse para el campo o la etiqueta (modifica los datos guardados en data string).

Si en el nivel 1 hay una cadena, esta es convertida en grob usando la minifuente.

030004 ˆIfGetFieldInternalValue ( # 🡺 val )

Retorna el valor interno de un campo (el valor interno está guardado en la pila virtual).

026004 ˆIfGetFieldValue ( # 🡺 val )

Retorna el valor externo de un campo (el mensaje número 4 convierte un valor interno en externo). Si este es MINUSONE, retorna xNOVAL.

027004 ˆIfGetCurrentFieldValue ( 🡺 )

Retorna el valor externo del campo actual.

Equivale a usar:

LAM 'CurrentField FLASHPTR IfGetFieldValue

024004 ˆIfSetFieldValue ( val # 🡺 )

Fija un nuevo valor para un campo.

En el nivel 2 de la pila debe estar el valor externo. El mensaje número 5 lo convierte a valor interno. Después es guardado el valor interno en la pila virtual.

Luego dibuja el valor externo del campo en la pantalla llamando al mensaje 4 y al 6.

Finalmente, llama al mensaje 26 (del campo actual, no del que está siendo cambiado) y 26 del formulario.

**Direcc. Nombre Descripción**

025004 ˆIfSetCurrentFieldValue ( val 🡺 )

Cambia el valor del campo actual.

Equivale a usar:

LAM 'CurrentField FLASHPTR IfSetFieldValue

028004 ˆIfGetFieldMessageHandler ( # 🡺 prg )

( # 🡺 ROMPTR )

Devuelve el **Message Handler de un campo**.

045004 ˆIfGetFieldPos ( # T/F(camp/etiq) 🡺 #x #y #b #h )

Devuelve el tamaño y posición de un objeto (a partir de StringData).

044004 ˆIfSetFieldPos ( # T/F(camp/etiq) #x #y #b #h 🡺 )

Cambia el tamaño y la posición de un objeto.

Nota: Una etiqueta sólo puede cambiar su posición.

Un campo CHECK sólo puede cambiar su posición Y.

029004 ˆIfGetFieldType ( # 🡺 #tipo\_campo )

Devuelve el **tipo de campo** (1, 12, 2 ó 32).

02A004 ˆIfGetFieldObjectsType ( # 🡺 {} )

( # 🡺 MINUSONE )

Devuelve una lista con los **tipos de objetos permitidos** para un campo.

02B004 ˆIfGetFieldDecompObject ( # 🡺 #Decomp )

Devuelve el valor “**decompile**” de un campo.

02C004 ˆIfGetFieldChooseData ( # 🡺 {} )

( # 🡺 MINUSONE )

Devuelve el valor “**Choose Data**” de un campo.

02D004 ˆIfGetFieldChooseDecomp ( # 🡺 ChooseDecompile )

Devuelve el valor “**Choose Decompile**” de un campo.

02E004 ˆIfGetFieldResetValue ( # 🡺 val )

Devuelve el “**valor reset**” de un campo.

02F004 ˆIfSetFieldResetValue ( val # 🡺 )

Cambia el “**valor reset**” de un campo.

032004 ˆIfGetNbFields ( 🡺 #n )

Devuelve el número de campos del formulario.

(a partir de StringData).

043004 ˆIfPutFieldsOnStack ( 🡺 ob1...obn )

Devuelve en la pila los valores externos de cada campo

031004 ˆIfDisplayFromData ( 🡺 )

Actualiza la pantalla.

Este comando toma en cuenta el tamaño de la línea de comandos.

046004 ˆIfDisplayFromData2 ( #campo #AlturaPíxelesLComand $dat 🡺 )

Actualiza la pantalla.

Muestra la **ayuda** correspondiente a #campo.

033004 ˆIfCheckSetValue ( # val 🡺 )

Este comando sólo es para campos **CHECKBOX**.

Muestra en la pantalla el campo **CHECKBOX** marcado o desmarcado.

No cambia el valor del campo **CHECKBOX**.

**Direcc. Nombre Descripción**

034004 ˆIfCheckFieldtype ( ob 🡺 ob flag )

( Z 🡺 Z/% flag )

Retorna TRUE si el objeto es de un tipo permitido en el campo actual.

Si el objeto es un entero y en el campo actual no se permiten enteros, entonces el entero es convertido a real primero.

Si existe línea de edición, no retorna el flag.

036004 ˆIfSetField ( #campo 🡺 )

Hace que un nuevo campo reciba el enfoque.

Llama a los mensajes 1,2 y 3.

038004 ˆIfKeyEdit ( 🡺 (cmd line) )

Edita el valor del campo actual. No podrás editar un campo **CHOOSE**.

Llama al mensaje 25, 25 del formulario y 23.

Equivale a presionar la tecla de menú EDIT.

037004 ˆIfKeyChoose ( 🡺 )

Si el campo actual es un campo **CHOOSE** o **COMBOCHOOSE**, muestra las opciones y permite al usuario escoger.

Llama al mensaje 17 y 17 del formulario.

Equivale a presionar la tecla de menú CHOOS.

03B004 ˆIfKeyInvertCheck ( 🡺 )

Si el campo actual es un campo **CHECKBOX**, invierte su valor.

Equivale a presionar la tecla de menú CHK.

035004 ˆIfReset ( 🡺 )

Permite escoger entre 2 opciones: Reset value y Reset all. Luego resetea el valor del campo actual o de todos los campos según la opción escogida. Para esto, fija el valor del campo como el valor del parámetro reset del campo.

Equivale a presionar la tecla de menú RESET.

03A004 ˆIfKeyCalc ( 🡺 val )

( 🡺 )

Interrumpe temporalmente el formulario de entrada e inicia un entorno con el valor del campo actual en el nivel uno de la pila. Permite al usuario computar un nuevo valor. Si el usuario se retira con CANCL u ON no se actualizará el campo actual.

Equivale a presionar la tecla de menú CALC.

039004 ˆIfKeyTypes ( 🡺 (cmd line) )

( 🡺 )

Muestra un browser con todos los posibles tipos para el objeto del campo actual.

Si el usuario, presiona ENTER o la tecla de menú NEW, una línea de comandos es abierta.

Llama al mensaje 18.

Equivale a presionar la tecla de menú TYPES.

03C004 ˆIfONKeyPress ( 🡺 )

Llama a los mensajes 13 y 14.

La acción por defecto es finalizar el formulario dejando FALSE en la pila (fijando el lam ‘Quit como TRUE).

Equivale a presionar la tecla CANCL o la tecla ON.

**Direcc. Nombre Descripción**

03D004 ˆIfEnterKeyPress ( 🡺 )

Llama a los mensajes 15 y 16.

La acción por defecto es finalizar el formulario dejando los valores externos de los campos y TRUE en la pila (fijando el lam ‘Quit como TRUE).

Equivale a presionar la tecla OK o la tecla ENTER.

042004 ˆIfMain2 ( $dat MessageHandler {} 🡺 F )

( $dat MessageHandler {} 🡺 ob1...obn T )

Programa interno principal para crear formularios de entrada.

La lista contiene 8•nc elementos, donde nc es el número de campos.

Para cada campo, debe haber 8 parámetros:

MsgH #TipoCampo TipoObjetos #dec ChData #ChDec

ValorReset ValorInicial

04A004 ˆIfInitDepth ( 🡺 )

Inicializa el contador interno de la pila.

Equivale a hacer: DEPTH ' LAM 'Depth STO

Este es usado luego de una acción que modifica la pila.

040004 ˆIfSetTitle ( $dat grb/$/# 🡺 $dat' )

Con este comando puedes modificar el **título**.

Por ejemplo, el siguiente código fija un nuevo **título**.

:: LAM 'StringData "NUEVO TITULO"

FLASHPTR IfSetTitle DROP

;

041004 ˆIfSetTitle2 ( grb/$/# 🡺 )

Con este comando también puedes modificar el **título**.

03F004 ˆIfSetHelpString ( $dat #campo $/# 🡺 $dat' )

Con este comando puedes modificar la **ayuda** de cualquier campo.

Por ejemplo, el siguiente código fija una nueva **ayuda** para el campo 0.

:: LAM 'StringData BINT0 "Nueva ayuda"

FLASHPTR IfSetHelpString DROP

;

Nota: La nueva cadena de ayuda debe de tener la misma cantidad de caracteres que la antigua cadena de ayuda.

048004 ˆIfSetAllHelpStrings ( $dat #MsjeInicial #NMsjes 🡺 $dat )

Fija el texto de **ayuda** de los campos.

047004 ˆIfSetAllLabelsMessages ( $dat #MsjeInicial #NMsjes 🡺 $dat )

Fija el texto de las etiquetas.

04D004 ˆIsUncompressDataString ( $datcompr 🡺 $dat )

Descomprime un data string comprimido.

37.9 Mensajes para Campos y para Formulario

Primero mostramos un cuadro resumen y luego se explica en detalle a cada uno.

**#MH Lugar Descripción**

0 Campo/Form Es llamado cuando se presiona una tecla.

1 Campo Es llamado cuando un campo está a punto de perder el enfoque.

2 Form Es llamado justo cuando un campo ha recibido el enfoque.

3 Campo Es llamado justo cuando un campo ha recibido el enfoque.

4 Campo Es llamado para conseguir el valor externo de un campo.

5 Campo Es llamado para conseguir el valor interno de un campo.

6 Campo Es llamado para dibujar un campo.

7 Form Es llamado al inicio para fijar el campo que primero tendrá el enfoque.

11 Form Provee el menú.

12 Form Cambia las tres últimas teclas de menú de la primera fila.

13 Form Es llamado al presionar la tecla CANCL u ON.

14 Form Es llamado al presionar la tecla CANCL u ON.

15 Form Es llamado al presionar la tecla ENTER u OK.

16 Form Es llamado al presionar la tecla ENTER u OK.

17 Campo/Form Es llamado al presionar la tecla CHOOS en un campo CHOOSE o

en un campo COMBOCHOOSE.

18 Form Es llamado al presionar la tecla TYPES en campos TEXTO,

COMBOCHOOSE o CHECK.

20 Campo/Form Es llamado cuando una nueva línea de edición es creada.

21 Campo/Form Es llamado cuando finaliza una línea de edición.

22 Campo Es llamado para validar una entrada de la línea de edición.

23 Campo Es llamado para descompilar un objeto para que se pueda editar

(al presionar la tecla EDIT)

24 Campo/Form Es llamado al presionar la tecla +/- en un campo CHOOSE o

en un campo CHECK.

25 Campo/Form Es llamado al presionar la tecla EDIT en un campo TEXTO,

COMBOCHOOSE o CHECK.

26 Campo/Form Es llamado cuando el valor de algún campo cambia.

37.9.1 Mensaje 0 (campo y formulario)

Este mensaje es llamado cuando se presiona alguna tecla. Este mensaje es enviado primero al message handler del campo actual y luego al message handler del formulario.

Por lo tanto, si el mensaje es llamado por ambos, solamente tendrá efecto el message handler del campo.

Si uno escoge la salida con efecto, entonces luego se ejecutará el programa colocado en la pila.

Si uno escoge la salida sin efecto, entonces luego se ejecutará el código normal correspondiente a la tecla presionada.

**Entrada** 2: #CódigoDeTecla

1: #PlanoDeTecla

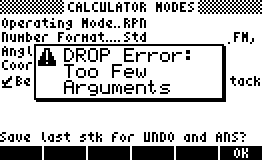
**Salida (con efecto)** 2: ::Programa

1: TRUE

**Salida (sin efecto)** 3: #CódigoDeTecla

2: #PlanoDeTecla

1: FALSE

a) Cuando se usa un formulario de entrada y no hay activa una línea de edición, algunas teclas realizan operaciones con la pila que a veces no son deseadas. Por ejemplo, la tecla STO, la tecla DROP, la tecla +/- realizan acciones con los objetos de la pila. Para quitar estas asignaciones de teclas puedes usar el siguiente message handler en el formulario.

' :: BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

LAM 'CurrentField ( #ct #p #campo )

FLASHPTR IfGetFieldType ( #ct #p #TipoCampo )

BINT2 ( #ct #p #TipoCampo #2 )

#> ( #ct #p flag )

3PICK ( #ct #p flag #ct )

BINT28 ( #ct #p flag #ct 28 )

#= ( #ct #p flag flag' )

AND ( #ct #p flag'' )

EditLExists? ( #ct #p flag'' flag''' )

ORcase

FALSE ( SALE CON ESTA PILA: #ct #p F )

( #ct #p )

2DUP ( #ct #p #ct #p )

TWO{}N ( #ct #p {#ct #p} )

\* Puedes agregar o quitar elementos de esta lista

{ { BINT12 BINT1 } ( NOTA: tecla STO )

{ BINT19 BINT2 } ( NOTA: tecla MTRW )

{ BINT19 BINT3 } ( NOTA: tecla EQW )

{ BINT21 BINT1 } ( NOTA: tecla DROP )

{ BINT21 BINT4 } ( NOTA: tecla ALPHA+DROP )

{ BINT28 BINT1 } ( NOTA: tecla +/- )

} ( #ct #p {#ct #p} {} )

FLASHPTR ListPos ( #ct #p #i/#0 )

#0<> ( #ct #p flag )

case2drop 'DoBadKeyT ( SALE CON ESTA PILA: Dobadkey T )

( #ct #p )

FALSE ( #ct #p F )

;

DROPFALSE

;

b) Para que funcione correctamente un campo **COMBOCHOOSE** puedes usar el siguiente message handler **en el formulario**, el cual cambia las acciones correspondientes al parámetro AppKeys del POL que se ejecutarán después de llamado el message handler.

' :: BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

FALSE ( #ct #p F )

LAM 'CurrentField ( #ct #p F #campo )

FLASHPTR IfGetFieldType ( #ct #p F #TipoCampo )

#2= ( #ct #p F flag ) ( TRUE si es combochoos )

NOT?SEMI

( #ct #p F )

R> ( #ct #p F prog )

FLASHPTR 002 0A5 ( #ct #p F $ ) ( Equivale al comando ->H )

"52133" ( #ct #p F $ "52133" ) ( dirección de BINT3 )

"B1133" ( #ct #p F $ "52133" "B1133" ) ( dirección de BINT2 )

FLASHPTR 00F 01A ( #ct #p F $' %3 ) ( Equivale al comando xSREPL )

DROP ( #ct #p F $' )

FLASHPTR 002 0A4 ( #ct #p F prog' ) ( Equivale al comando H-> )

>R ( #ct #p F )

;

DROPFALSE

;

También puedes usar en su lugar el siguiente message handler **en cada campo COMBOCHOOSE** que esté presente.

' :: BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

R> ( #ct #p prog )

FLASHPTR 002 0A5 ( #ct #p $ ) ( Equivale al comando ->H )

"52133" ( #ct #p $ "52133" ) ( dirección de BINT3 )

"B1133" ( #ct #p $ "52133" "B1133" ) ( dirección de BINT2 )

FLASHPTR 00F 01A ( #ct #p $' %3 ) ( Equivale al comando xSREPL )

DROP ( #ct #p $' )

FLASHPTR 002 0A4 ( #ct #p prog' ) ( Equivale al comando H-> )

>R ( #ct #p )

FALSE ( #ct #p F )

;

DROPFALSE

;

c) Para quitar las asignaciones de algunas teclas (como STO, DROP y +/-) y para que funcionen correctamente los campos combochoose (combinación de los casos a y b) puedes usar el message handler de la parte ‘a’ **en el formulario**, y el message handler de la parte ‘b’ (del campo) **en cada campo COMBOCHOOSE** que haya en tu aplicación.

O también puedes usar solamente el siguiente message handler en el formulario.

' :: BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

LAM 'CurrentField ( #ct #p #campo )

FLASHPTR IfGetFieldType ( #ct #p #TipoCampo )

BINT2 ( #ct #p #TipoCampo #2 )

#> ( #ct #p flag )

3PICK ( #ct #p flag #ct )

BINT28 ( #ct #p flag #ct 28 )

#= ( #ct #p flag flag' )

AND ( #ct #p flag'' )

EditLExists? ( #ct #p flag'' flag''' )

ORcase

FALSE ( SALE CON ESTA PILA: #ct #p F )

( #ct #p )

2DUP ( #ct #p #ct #p )

TWO{}N ( #ct #p {#ct #p} )

\* Puedes agregar o quitar elementos de esta lista

{ { BINT12 BINT1 } ( NOTA: tecla STO )

{ BINT19 BINT2 } ( NOTA: tecla MTRW )

{ BINT19 BINT3 } ( NOTA: tecla EQW )

{ BINT21 BINT1 } ( NOTA: tecla DROP )

{ BINT21 BINT4 } ( NOTA: tecla ALPHA+DROP )

{ BINT28 BINT1 } ( NOTA: tecla +/- )

} ( #ct #p {#ct #p} {} )

FLASHPTR ListPos ( #ct #p #i/#0 )

#0<> ( #ct #p flag )

case2drop 'DoBadKeyT ( SALE CON ESTA PILA: Dobadkey TRUE )

( #ct #p )

FALSE ( #ct #p F )

LAM 'CurrentField ( #ct #p F #campo )

FLASHPTR IfGetFieldType ( #ct #p F #TipoCampo )

#2= ( #ct #p F flag )

NOT?SEMI

( #ct #p F )

R> ( #ct #p F prog )

FLASHPTR 002 0A5 ( #ct #p F $ ) ( Equivale al comando ->H )

"52133" ( #ct #p F $ "52133" ) ( dirección de BINT3 )

"B1133" ( #ct #p F $ "52133" "B1133" ) ( dirección de BINT2 )

FLASHPTR 00F 01A ( #ct #p F $' %3 ) ( Equivale al comando xSREPL )

DROP ( #ct #p F $' )

FLASHPTR 002 0A4 ( #ct #p F prog' ) ( Equivale al comando H-> )

>R ( #ct #p F )

;

DROPFALSE

;

37.9.2 Mensaje 1 (campo)

Este mensaje es llamado cuando el campo está a punto de perder el enfoque. Puedes hacer cualquier cosa aquí, incluso hacer que el campo actual no pierda el enfoque. Si haces eso, entonces no serán llamados los mensajes 2 y 3.

**Entrada** 1: #campo (número del campo que recibirá el enfoque)

**Salida** 2: #campo' (número del campo que recibirá el enfoque)

1: TRUE o FALSE

37.9.3 Mensaje 2 (formulario)

Este mensaje es llamado justo cuando un campo ha recibido el enfoque.

**Entrada** Nada

**Salida** 1: TRUE o FALSE

37.9.4 Mensaje 3 (campo)

Este mensaje es llamado justo cuando un campo ha recibido el enfoque.

Este mensaje es llamado después del mensaje número 2.

**Entrada** Nada

**Salida** 1: TRUE o FALSE

**Valor Interno y valor externo de un campo.**

Cuando se usa el comando ^IfGetFieldValue es llamado el mensaje 4, de manera que se obtiene el valor externo a partir del valor interno (el cual está guardado en la pila virtual).

Cuando se usa el comando ^IfSetFieldValue es llamado el mensaje 5, de manera que se obtiene el valor interno (para guardarlo en la pila virtual) a partir del valor externo (el cual nosotros lo proporcionamos al comando o es el objeto ingresado desde la línea de edición o de la lista de opciones de un campo Choose).

Cuando se finaliza el formulario con ENTER u OK, son retornados los valores externos de los campos y TRUE.

Lo más usual es usar sólo el mensaje número 5 (llamado al inicio y cuando cambia el valor del campo) sin cambiar el valor interno o externo, para que, según las condiciones existentes, los otros campos o etiquetas se hagan visibles o invisibles o se cambien sus valores.

37.9.5 Mensaje 4 (campo)

Este mensaje es llamado cada vez que se quiere conseguir el valor externo de un campo (el cual será mostrado posteriormente en la pantalla).

**Entrada** 1: Valor Interno

**Salida** 2: Valor Externo

1: TRUE o FALSE

37.9.6 Mensaje 5 (campo)

Este mensaje es llamado cuando se inicia el formulario de entrada.

También es llamado cuando el valor del campo cambia, ya sea por un valor ingresado o escogido por el usuario, o por la asignación de un valor al campo con el comando ^IfSetFieldValue o con el comando ^IfSetCurrentFieldValue.

Su misión es convertir el valor externo en valor interno (el cual será guardado en la pila virtual).

Si este mensaje no es incluido en el message handler del campo, entonces el Valor externo siempre se convertirá en el Valor Interno del campo.

**Entrada** 1: Valor Externo

**Salida** 2: Valor Interno

1: TRUE o FALSE

37.9.7 Mensaje 6 (campo)

Es llamado al inicio para cada campo y cuando cambia el valor de un campo (no es llamado para un campo **CHECKBOX**).

Si te decides por la salida con efecto, deberás mostrar el grob correspondiente a este campo en la pantalla (usando el comando ˆIfSetGrob).

Si te decides por la salida sin efecto, entonces luego será llamado el código estándar para mostrar el valor de ese campo.

**Entrada** 2: #Campo

1: Valor Externo

**Salida (con efecto)** 2: TRUE o FALSE

1: TRUE

**Salida (sin efecto)** 3: #Campo

2: Valor Externo

1: FALSE

Aquí un ejemplo de la salida con efecto para este mensaje:

' :: BINT6 #=casedrop ( #c val -> flag T // #c val F )

:: ( #campo valor )

TRUE ( #campo valor TRUE )

SWAP ( #campo TRUE valor )

DO>STR ( #campo TRUE $ )

$>grob ( #campo TRUE grob )

FLASHPTR IfSetGrob ( ) ( muestra el campo en la pantalla )

TrueTrue ( TRUE TRUE )

;

DROPFALSE

;

37.9.8 Mensaje 7 (formulario)

Este mensaje es enviado durante el inicio del formulario de entrada para fijar el número del campo que primero tendrá el enfoque. Sólo debes borrar el bint de la pila con DROP y luego escribir el bint correspondiente al campo que desees, seguido por TRUE o FALSE.

**Entrada** 1: #0

**Salida** 2: #Campo

1: TRUE o FALSE

37.9.9 Mensaje 11 (formulario)

Este mensaje es llamado antes del inicio del formulario de entrada, y puede ser usado para proveer un menú. El menu debe estar en el formato descrito en la sección 39.1.

El menú original es una lista con 13 elementos. Los últimos 12 corresponden a las teclas.

Por ejemplo: EDIT es el segundo elemento, CHOOS es el tercero, RESET es el séptimo.

**Entrada** 1: { } (menú original)

**Salida (con efecto)** 3: { } (menú original)

2: { }' (menú nuevo)

1: TRUE

**Salida (sin efecto)** 2: { } (menú original)

1: FALSE

Si el message handler del formulario de entrada es el siguiente, entonces será disponible una tecla de menú para ver un texto de ayuda.

' :: BINT11 #=casedrop ( F ) ( {} -> {} {}' T // {} F )

:: ( {}menú )

{ "HELP" :: TakeOver FALSE "Texto de ayuda" ViewStrObject DROP ; }

( {}menú {} )

BINT4 ( {}menú {} #4 ) ( 4 y no 3 para la tecla F3 )

3PICK ( {}menú {} #4 {}menú )

PUTLIST ( {}menú {}menú' )

TRUE ( {}menú {}menú' T )

;

DROPFALSE

;

 ****

37.9.10 Mensaje 12 (formulario)

Este mensaje es llamado antes del inicio del formulario de entrada, y puede ser usado para cambiar las 3 últimas teclas de la primera fila del menú. Si este mensaje es manejado, se debe de retornar una lista con tres sublistas, cada una de estas será la definición de la tecla.

Si los mensajes 11 y 12 son llamados para decidir sobre la definición de una de estas tres teclas, la definición del mensaje 11 será la definitiva.

Entrada Nada

Salida (con efecto) 2: { } (lista con tres elementos)

1: TRUE

Salida (sin efecto) 1: FALSE

A continuación un message handler que imita a la acción por defecto.

' :: BINT12 #=casedrop ( F ) ( -> {F4 F5 F6} T // F )

:: NullMenuKey ( {} )

{ "CANCL" FLASHPTR IfONKeyPress } ( {} {} )

{ "OK" FLASHPTR IfEnterKeyPress } ( {} {} {} )

BINT3 ( {} {} {} #3 )

{}N ( {} )

TRUE ( {} T )

;

DROPFALSE

;

37.9.11 Mensaje 13 (formulario)

Permite al usuario reemplazar la accion por defecto para cancelar el formulario de entrada. Este mensaje es llamado cuando es presionada la tecla CANCL o la tecla ON.

Si este mensaje es llamado, entonces ya no se ejecutará el código por defecto.

La acción por defecto es fijar el valor del LAM **'Quit** cómo TRUE para finalizar el POL y dejar FALSE en la pila.

Si este mensaje es llamado y se hace una salida con efecto, entonces, el mensaje 14 ya no será llamado después.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación un message handler que imita a la acción por defecto.

' :: BINT13 #=casedrop

:: ( )

BINT14 ( 14 )

LAM 'IfHandler ( 14 MHF )

EVAL ( T/F T // F )

NOT ( T/F F // T )

IT

TRUE

( T/F // T )

ITE

:: FALSE ( F )

TRUE ' LAM 'Quit STO ( F )

;

ERRBEEP ( )

( F // )

TRUE ( F T // T )

;

DROPFALSE

;

37.9.12 Mensaje 14 (formulario)

Este mensaje es llamado cuando es presionada la tecla CANCL o la tecla ON.

El programador puede evitar que el formulario sea finalizado si hay una entrada inválida, por ejemplo.

**Entrada** Nada

**Salida (con efecto)** 2: TRUE o FALSE (TRUE finalizará el POL, FALSE no)

1: TRUE

**Salida (sin efecto)** 1: FALSE

Por lo tanto:

* Para que se finalice el formulario de entrada debes poner en la pila: TRUE TRUE o solo FALSE.
* Para que no finalice el formulario de entrada debes poner en la pila: FALSE TRUE.

Con el siguiente message handler se consigue que si el usuario presiona CANCL entonces se le pregunta si está seguro de salir.

' :: BINT14 #=casedrop ( F ) ( -> flag T // F )

:: ( )

"Estás seguro de salir?" ( $ )

AskQuestion ( T/F )

TRUE ( T/F T )

;

DROPFALSE

;

37.9.13 Mensaje 15 (formulario)

Permite al usuario reemplazar la accion por defecto para aceptar el formulario de entrada. Este mensaje es llamado cuando es presionada la tecla ENTER o la tecla OK.

Si este mensaje es llamado, entonces ya no se ejecutará el código por defecto.

La acción por defecto es fijar el valor del LAM **'Quit** cómo TRUE para finalizar el POL y dejar en la pila los valores externos de los campos seguidos por TRUE.

Si este mensaje es llamado y se hace una salida con efecto, entonces , el mensaje 16 ya no será llamado después.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación, un message handler que imita a la acción por defecto.

' :: BINT15 #=casedrop

:: ( )

BINT16 ( 16 )

LAM 'IfHandler ( 16 MHF )

EVAL ( T/F T // F )

NOT ( T/F F // T )

IT

TRUE

( T/F // T )

ITE

:: FLASHPTR IfPutFieldsOnStack

TRUE ( ob1...obn T )

TRUE ' LAM 'Quit STO

;

ERRBEEP ( )

( ob1...obn T // )

TRUE ( ob1...obn T T // T )

;

DROPFALSE

;

37.9.14 Mensaje 16 (formulario)

Este mensaje es llamado cuando es presionada la tecla ENTER o la tecla OK.

El programador puede evitar que el formulario sea finalizado si hay una entrada inválida, por ejemplo.

**Entrada** Nada

**Salida (con efecto)** 2: TRUE o FALSE (TRUE finalizará el POL, FALSE no)

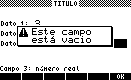
1: TRUE

**Salida (sin efecto)** 1: FALSE

Por lo tanto:

* Para que se finalice el formulario de entrada debes poner en la pila: TRUE TRUE o solo FALSE.
* Para que no finalice el formulario de entrada debes poner en la pila: FALSE TRUE.

Con el siguiente message handler se impide que el usuario confirme el formulario de entrada cuando hay campos vacíos. Además, el enfoque se mueve hacia el campo que está vacío.



' :: BINT16 #=casedrop ( F ) ( -> flag T // F )

:: ( )

FLASHPTR IfPutFieldsOnStack ( ob1 ob2...obn )

FLASHPTR IfGetNbFields ( ob1 ob2...obn #n )

{}N ( { ob1 ob2...obn } )

' xNOVAL ( { ob1 ob2...obn } xNOVAL )

SWAP ( xNOVAL { ob1 ob2...obn } )

FLASHPTR ListPos ( #pos/#0 )

DUP ( #pos/#0 #pos/#0 )

#0=ITE

:: ( #0 )

DROP ( )

TrueTrue ( T T )

;

:: ( #pos )

#1- ( #campo )

FLASHPTR IfSetField ( )

"Este campo está vacio" ( $ )

FlashWarning ( )

FalseTrue ( F T )

;

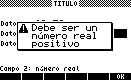
( flag T )

;

DROPFALSE

;

Con el siguiente message handler se exige que todos los campos deben contener a números reales positivos. Si eso no se cumple, se impide que el usuario confirme el formulario de entrada con OK o ENTER. Además, el enfoque se mueve hacia el campo que se debe de corregir.



' :: BINT16 #=casedrop ( F ) ( -> flag T // F )

:: ( )

TRUE ( T )

1LAMBIND ( )

FLASHPTR IfGetNbFields ( #n )

ZERO\_DO (DO)

INDEX@ ( #i )

FLASHPTR IfGetFieldValue ( campoi )

:: DUPTYPEREAL? ( campoi flag )

NOTcase

DROPFALSE ( F )

( %i )

%0> ( flag )

;

( flag )

NOT\_IT

:: "Debe ser un número real positivo" ( $ )

FlashWarning ( )

INDEX@ ( #i )

FLASHPTR IfSetField ( )

FALSE ( F )

1PUTLAM ( )

ExitAtLOOP ( )

;

LOOP

( )

1GETABND ( flag )

TRUE ( flag T )

;

DROPFALSE

;

37.9.15 Mensaje 17 (campo y formulario)

Este mensaje es llamado cuando se presiona la tecla CHOOS en un campo CHOOSE o en un campo COMBOCHOOSE. Primero es llamado el mensaje del campo, luego el del formulario.

Si se maneja el mensaje del campo, ya no es llamado el mensaje del formulario ni tampoco el código estándar.

Si se maneja el mensaje del formulario, ya no se llama al código estándar.

Para manejar a este mensaje debes dejar TRUE en la pila y tu mismo deberás mostrar un browser que permita escoger a un elemento.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

37.9.16 Mensaje 18 (formulario)

Este mensaje es llamado cuando se presiona la tecla TYPES en un campo TEXTO, COMBOCHOOSE o CHECK.

Si este mensaje es manejado, ya no se llamará al código estándar y tu mismo deberás mostrar un browser con los tipos permitidos para el valor del campo actual.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

37.9.17 Mensaje 20 (campo y formulario)

Este mensaje es llamado cuando una nueva línea de edición es creada. Si se maneja el mensaje para el campo, ya no se llama al mensaje del formulario.

**Campo:**

**Entrada** Nada

**Salida (con efecto)** 1: TRUE (ya no se llama al mensaje del formulario)

**Salida (sin efecto)** 1: FALSE

**Formulario**:

**Entrada** Nada

**Salida** 1: TRUE o FALSE (no hay diferencia)

37.9.18 Mensaje 21 (campo y formulario)

Este mensaje es llamado cuando finaliza una línea de edición.

Si se maneja el mensaje para el campo, ya no se llama al mensaje del formulario.

**Campo:**

Entrada Nada

Salida (con efecto) 1: TRUE (ya no se llama al mensaje del formulario)

Salida (sin efecto) 1: FALSE

**Formulario**:

**Entrada** Nada

**Salida** 1: TRUE o FALSE (no hay diferencia)

37.9.19 Mensaje 22 (campo)

Este mensaje es para validar una entrada de la línea de edición.

Este mensaje es llamado cuando hay una línea de edición activa y es presionada la tecla ENTER o la tecla OK.

Cuando este mensaje es llamado, los objetos de la pila RPN han sido movidos temporalmente a la pila virtual. De esta manera, en la pila no habrá ningún objeto.

Si este mensaje es manejado, ya no se hará la acción por defecto (ya no se hará nada más) y la línea de edición continuará activa a menos que nosotros mismos escribamos código para desactivarla.

Puedes usar la salida con efecto cuando ha sido ingresado un objeto de un tipo válido pero con un valor no deseado.

Sin embargo, también puedes usar el mensaje número 5 para validar objetos (ver ejemplos).

**Entrada** Nada. Hay una línea de comandos activa. No hay nada en la pila.

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

NOTA: El comando ^IfMain guarda 8 parámetros (excepto la ayuda y los cuatro bints que indican la posición y el tamaño) de cada uno de los campos en el nivel más alto de la pila virtual. De esta forma, comandos como ˆIfGetFieldResetValue, ^IfGetFieldValue o ^IfSetFieldValue que obtienen valores del nivel más alto de la Pila Virtual o la modifican, no funcionarán adecuadamente al ser ejecutados mientras se está llamando al mensaje 22.

Para poder usar estos comandos aquí, puedes usar el NULLNAME SWAP\_VSTACK que se encuentra al final del capítulo 24 para intercambiar los dos niveles más altos de la Pila Virtual. Por ejemplo así:

SWAP\_VSTACK

FLASHPTR IfSetCurrentFieldValue

SWAP\_VSTACK

37.9.20 Mensaje 23 (campo)

Este mensaje es llamado para descompilar un objeto para que se pueda editar. Si este mensaje no es llamado, la descompilación es hecha con el comando DecompEdit.

Este mensaje es llamado cuando se presiona la tecla EDIT. Este mensaje no sera llamado cuando se manejen los mensajes 25 del formulario o 25 del campo.

**Entrada** 1: Valor externo

**Salida (con efecto)** 2: Cadena

1: TRUE

**Salida (sin efecto)** 2: Valor externo

1: FALSE

37.9.21 Mensaje 24 (campo y formulario)

Este mensaje es llamado cuando se presiona la tecla +/- en un campo CHOOSE o en un campo CHECK.

Primero es llamado el mensaje del campo, luego el del formulario.

Si se maneja el mensaje del campo, ya no es llamado el mensaje del formulario ni tampoco el código estándar.

Si se maneja el mensaje del formulario, ya no se llama al código estándar.

Para manejar a este mensaje debes dejar TRUE en la pila.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

37.9.22 Mensaje 25 (campo y formulario)

Este mensaje es llamado cuando se presiona la tecla EDIT en un campo TEXTO, COMBOCHOOSE o CHECK.

Primero es llamado el mensaje del campo, luego el del formulario.

Si se maneja el mensaje del campo, ya no es llamado el mensaje del formulario ni tampoco el código estándar.

Si se maneja el mensaje del formulario, ya no se llama al código estándar.

Para manejar a este mensaje debes dejar TRUE.

La entrada es el valor externo del campo actual.

Puedes iniciar la línea de edición con ese valor. O también puedes editar el valor del campo actual con otro editor y luego modificar el campo actual. O también puedes editar el valor del campo actual con otro editor y dejar el valor modificado en la pila seguido por TRUE (ese valor será puesto en el campo actual de forma automática).

Si este mensaje es manejado, ya no se llamará al mensaje número 23.

**Entrada** 1: Valor externo

**Salida (con efecto)** 1: TRUE

**Salida (con efecto)** 2: Valor externo

1: TRUE

**Salida (sin efecto)** 2: Valor externo

1: FALSE

a) Si deseas editar en la línea de edición (objetos de cualquier tipo), puedes usar el siguiente message handler:

' :: BINT25 #=casedrop ( C/F ) ( Vext -> T // Vext T // Vext F )

:: ( ob )

DecompEdit ( $ ) ( descompila objeto )

CMD\_PLUS ( ) ( abre el editor con cursor al final de $ )

TRUE ( T )

;

DROPFALSE

;

b) Si deseas editar en el escritor de ecuaciones a objetos algebraicos, nombres globales, reales, complejos y enteros, puedes usar el siguiente message handler.

Válido cuando el parámetro TiposPermitidos del campo es { 0 1 6 9 # FF }

\* Llama al NULLNAME EditaEnEQW

' :: BINT25 #=casedrop ( C/F ) ( Vext -> T // Vext T // Vext F )

:: ( ob )( ob es symb/id/Z/%/C% )

EditaEnEQW ( ob' T // F ) ( ob' es symb/id/Z/%/C% )

DROPTRUE ( ob' T // T )

;

DROPFALSE

;

\* Si en la pila se encuentra xNOVAL, abre el EQW.

\* Si hay otro objeto, lo edita en EQW (de ser posible)

NULLNAME EditaEnEQW ( ob -> ob' T // F )

:: DUP ( ob ob )

' xNOVAL ( ob ob xNOVAL )

EQUAL ( ob flag )

ITE

:: DROP ( )

FLASHPTR EQW3 ( ob' T // F ) ( abre EQW )

;

FLASHPTR EQW3Edit ( ob' T // F ) ( edita ob en EQW )

;

c) Si deseas editar en el escritor de ecuaciones a objetos algebraicos o variables globales, de tal forma que sólo se permitan expresiones que contengan a por lo menos una variable global, puedes usar el siguiente message handler.

Válido cuando el parámetro TiposPermitidos del campo es { BINT6 BINT9 }

\* Llama al NULLNAME EditaEnEQW

\* Llama al NULLNAME VariablesAqui?

' :: BINT25 #=casedrop ( C/F ) ( Vext -> T // Vext T // Vext F )

:: ( ob )

BEGIN

:: EditaEnEQW ( ob' T // F ) ( ob' es symb/id/Z/%/C% )

NOTcase

TrueTrue ( sale con: T T )

( ob' )

DUP ( ob' ob' )

VariablesAqui? ( ob' flag ) ( test realizado a ob' )

NOTcase

:: "Escribe objeto con variables" ( ob' $ )

FlashWarning ( ob' )

FALSE ( ob' F )

;

( ob' )

TrueTrue ( ob' T T )

;

UNTIL

( ob' T // T )

;

DROPFALSE

;

\* Retorna TRUE si el objeto contiene a por lo menos una variable

\* Retorna FALSE si el objeto no contiene variables

NULLNAME VariablesAqui? ( ob -> flag )

:: FLASHPTR LIDNText ( {} ) ( retorna lista de variables de la expresión )

NULLCOMP? ( flag )

NOT ( flag' )

;

d) Si deseas editar en el escritor de ecuaciones a objetos algebraicos, de tal forma que sólo se permitan expresiones que contengan a por lo menos una variable global, puedes usar el siguiente message handler.

Válido cuando el parámetro TiposPermitidos del campo es { BINT9 }

\* Llama al NULLNAME EditaEnEQW

\* Llama al NULLNAME SimbolicoConVars?

' :: BINT25 #=casedrop ( C/F ) ( Vext -> T // Vext T // Vext F )

:: ( ob )

BEGIN

:: EditaEnEQW ( ob' T // F ) ( ob' es symb/id/Z/%/C% )

NOTcase

TrueTrue ( sale con: T T )

( ob' )

DUP ( ob' ob' )

SimbolicoConVars? ( ob' flag ) ( test realizado a ob' )

NOTcase

:: "Escribe objeto simbólico con variables" ( ob' $ )

FlashWarning ( ob' )

FALSE ( ob' F )

;

( symb )

TrueTrue ( symb T T )

;

UNTIL

( symb T // T )

;

DROPFALSE

;

\* Retorna TRUE si el objeto es simbólico y contiene por lo

\* menos una variable

\* Retorna FALSE si el objeto no es simbólico o no contiene variables

NULLNAME SimbolicoConVars? ( ob -> T // F )

:: ( ob )

DUPTYPESYMB? ( ob flag )

NOTcase

DROPFALSE ( symb )

FLASHPTR LIDNText ( {} ) ( retorna lista de variables de la expresión )

NULLCOMP? ( flag )

NOT ( flag' )

;

e) Si deseas editar en el escritor de matrices a arreglos reales, arreglos complejos o matrices simbólicas, puedes usar el siguiente message handler.

Válido cuando el parámetro TiposPermitidos del campo es { BINT3 BINT4 }

\* Llama al NULLNAME EditaFormacionEnMTRW

' :: BINT25 #=casedrop ( C/F ) ( Vext -> T // Vext T // Vext F )

:: ( ob )

EditaFormacionEnMTRW ( RealArry/CArry/MATRIX T // F )

DROPTRUE ( RealArry/CArry/MATRIX T // T )

;

DROPFALSE

;

\* Si en la pila se encuentra xNOVAL, abre el MTRW.

\* Si hay otro objeto, lo edita en MTRW (de ser posible)

NULLNAME EditaFormacionEnMTRW ( ob -> RealArry/CArry/MATRIX T // F )

::

RunSafeFlags

:: BEGIN

BINT91 ClrSysFlag

DUP ( ob ob )

' xNOVAL ( ob ob xNOVAL )

EQUAL ( ob flag )

ITE

:: DROP ( )

FLASHPTR RunDoNewMatrix ( ob' T // F ) ( abre MTRW )

;

FLASHPTR RunDoOldMatrix ( ob' T // F ) ( edita ob en MTRW )

ITE

:: DUPTYPELIST? ( ob' flag )

ITE

FALSE ( ob' F )

TrueTrue ( ob' T T )

;

FalseTrue

UNTIL

( RealArry/CArry/MATRIX T // F )

;

;

f) Si deseas editar en el escritor de matrices a arreglos reales, puedes usar el siguiente message handler.

Válido cuando el parámetro TiposPermitidos del campo es { BINT3 }

\* Llama al NULLNAME EditaArregloRealEnMTRW

' :: BINT25 #=casedrop ( C/F ) ( Vext -> T // Vext T // Vext F )

:: ( ob )

EditaArregloRealEnMTRW ( RealArry T // F )

DROPTRUE ( RealArry T // T )

;

DROPFALSE

;

\* Si en la pila se encuentra xNOVAL, abre el MTRW.

\* Si hay otro objeto, lo edita en MTRW (de ser posible)

\* Retorna un arreglo real y TRUE o sólo FALSE

NULLNAME EditaArregloRealEnMTRW ( ob -> RealArry T // F )

::

RunSafeFlags

:: BEGIN

BINT91 ClrSysFlag

DUP ( ob ob )

' xNOVAL ( ob ob xNOVAL )

EQUAL ( ob flag )

ITE

:: DROP ( )

FLASHPTR DoNewMatrixReal ( ob' T // F ) ( abre MTRW )

;

FLASHPTR DoOldMatrixReal ( ob' T // F ) ( edita ob en MTRW )

( ob' T // F )

ITE

:: DUPTYPELIST? ( ob' flag )

ITE

:: INNERDUP ( ob1...obn #f #f )

ZERO\_DO

ROLL ( ...obi )

INNERCOMP ( ... ob1'...obm' #c )

TYPEMATRIX\_ ( ... ob1'...obm' #c #2686 )

COMPN\_ ( ... 1DMATRIX )

ISTOP@ ( ... 1DMATRIX #f )

LOOP

TYPEMATRIX\_ ( 1DMATRIX1...1DMATRIXn #f #2686 )

COMPN\_ ( 2DMATRIX1 )

FALSE ( 2DMATRIX1 F )

;

TrueTrue ( ob' T T )

;

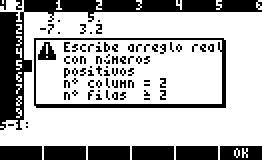
FalseTrue

UNTIL

( RealArry T // F )

;

;

g) Si deseas editar en el escritor de matrices a arreglos reales, de forma que el resultado sea un arreglo real de números positivos 2 dos columnas y con 2 o más filas, entonces puedes usar el siguiente message handler.

Válido cuando el parámetro TiposPermitidos del campo es { BINT3 }

\* Llama al NULLNAME EditaArregloRealEnMTRW

\* Llama al NULLNAME TestRealArry

' :: BINT25 #=casedrop ( C/F ) ( Vext -> T // Vext T // Vext F )

:: ( ob )

BEGIN

:: EditaArregloRealEnMTRW ( RealArry T // F )

NOTcase

TrueTrue ( sale con: T T )

( ob' )

DUP ( ob' ob' )

TestRealArry ( ob' flag ) ( test al objeto de la pila )

NOTcase

:: "Escribe arreglo real con números positivos\0Anº column = 2\0Anº filas Š 2"

FlashWarning ( ob' )

FALSE ( ob' F )

;

( [[%]] )

TrueTrue ( [[%]] T T )

;

UNTIL

( [[%]] T // T )

;

DROPFALSE

;

\* Realiza un test al objeto del nivel uno de la pila

\* Retorna TRUE si en la pila hay un arreglo real con números positivos, de

\* 2 dimensiones con 2 columnas y con un número de filas mayor o igual a 2.

NULLNAME TestRealArry ( ob -> flag )

::

DUP TYPERARRY? ( ob flag )

NOTcase

DROPFALSE

( RealArry )

DUP ( RealArry RealArry )

FLASHPTR MDIMS ( [[%]] #filas #cols T // [%] #elem F )

NOTcase

2DROPFALSE

( [[%]] #filas #cols )

#2= ( [[%]] #filas flag )

SWAP ( [[%]] flag #filas )

BINT1 #> ( [[%]] flag flag' )

AND ( [[%]] flag'' )

NOTcase

DROPFALSE

( [[%]] ) ( arreglo real de 2 dimensiones en la pila )

FLASHPTR XEQARRY> ( %1...%n {%f %c} )

INCOMPDROP ( %1...%n %f %c )

%\* ( %1...%n %f·c )

COERCE ( %1...%n #f·c )

ONE\_DO %MIN LOOP

( %' )

%0> ( flag )

;

37.9.23 Mensaje 26 (campo y formulario)

El mensaje del campo actual es llamado cuando el valor de algún campo cambia, sea de forma manual o con el comando ^IfSetFieldValue.

El mensaje del formulario es llamado al inicio para cada campo y cuando el valor de cualquier campo cambia, sea de forma manual o con el comando ^IfSetFieldValue.

Puedes hacer cualquier cosa aquí.

Primero es llamado el mensaje del campo, luego el del formulario.

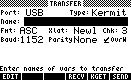
**Entrada** Nada

**Salida** 1: TRUE o FALSE

37.10 Ejemplos

Ejemplo 1 IfMain

**Imitando la apariencia del formulario TRANSFER.**

Este ejemplo imita la apariencia del formulario de entrada TRANSFER, pero es muy diferente al original en su funcionamiento.

El código define todas las etiquetas y los campos que se mostrarán. El formulario tiene un message handler muy simple que maneja a dos mensajes. El mensaje 7 para fijar el campo que tendrá el enfoque inicialmente; y el mensaje 12 para fijar las tres últimas teclas de la primera página del menú (sin embargo, las teclas en nuestro ejemplo sólo hacen beep al ser presionadas).

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMaEmuTransfer ( -> ob1...obn T // F )

:: CK0 ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Port:" 1 10 ( #xetiq ) ( #yetiq=#ycampo+1 )

"Type:" 70 10 ( #xetiq ) ( #yetiq=#ycampo+1 )

"Name:" 1 19 ( #xetiq ) ( #yetiq=#ycampo+1 )

"Fmt:" 1 28 ( #xetiq ) ( #yetiq=#ycampo+1 )

"Xlat:" 49 28 ( #xetiq ) ( #yetiq=#ycampo+1 )

"Chk:" 103 28 ( #xetiq ) ( #yetiq=#ycampo+1 )

"Baud:" 1 37 ( #xetiq ) ( #yetiq=#ycampo+1 )

"Parity:" 49 37 ( #xetiq ) ( #yetiq=#ycampo+1 )

"OvrW" 111 37 ( #xetiq=#xcampo+7 en CHK ) ( #yetiq=#ycampo+1 )

\* Campo 0: Port

'DROPFALSE ( MH del campo )

BINT26 ( #x )

BINT9 ( #y: 9 para fila 1/6 )

BINT24 ( #w )

BINT8 ( #h: 8 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT0 ( #Decompile: 0=NoDesc No1=SysF ) ( $ #[ME] ) ( $ )

"Choose transfer port" ( Ayuda )

{ "IrDA" "USB" "Serial" } ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

"USB" ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* Campo 1: Type

'DROPFALSE ( MH del campo )

BINT92 ( #x )

BINT9 ( #y: 9 para fila 1/6 )

BINT36 ( #w )

BINT8 ( #h: 8 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT0 ( #Decompile: 0=NoDesc No1=SysF ) ( $ #[ME] ) ( $ )

"Choose type of transfer" ( Ayuda )

{ "Kermit" "XModem" } ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

"Kermit" ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* Campo 2: Name

'DROPFALSE ( MH del campo )

BINT25 ( #x )

BINT18 ( #y: 18 para fila 2/6 )

BINT103 ( #w )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT5 BINT6 } ( TiposPermitidos: Listas e ids )

BINT2 ( #Decompile: 2=Actual No1=SysF ) ( ) ( "$" 'id' u )

"Enter names of vars to transfer" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

MINUSONE ( ValorReset: campo TEXTO en blanco )

MINUSONE ( ValorInicial: campo TEXTO en blanco )

\* Campo 3: Fmt

'DROPFALSE ( MH del campo )

BINT20 ( #x )

BINT27 ( #y: 27 para fila 3/6 )

BINT18 ( #w )

BINT8 ( #h: 8 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT0 ( #Decompile: 0=NoDesc No1=SysF ) ( $ #[ME] ) ( $ )

"Choose transfer format" ( Ayuda )

{ "ASCII" "Binary" } ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

"ASCII" ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* Campo 4: Xlat

'DROPFALSE ( MH del campo )

BINT74 ( #x )

BINT27 ( #y: 27 para fila 3/6 )

BINT24 ( #w )

BINT8 ( #h: 8 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT0 ( #Decompile: 0=NoDesc No1=SysF ) ( $ #[ME] ) ( $ )

"Choose character translations" ( Ayuda )

{ "None" "Newline (Ch 10)" "Chr 128-159" "Chr 128-255" } ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

"Newline (Ch 10)" ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* Campo 5: Chk

'DROPFALSE

BINT122 ( #x )

BINT27 ( #y: 27 para fila 3/6 )

BINT7 ( #w )

BINT8 ( #h: 8 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT0 ( #Decompile: 0=NoDesc No1=SysF ) ( $ #[ME] ) ( $ )

"Choose checksum type" ( Ayuda )

{ "1" "2" "3" } ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

"3" ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* Campo 6: Baud

'DROPFALSE ( MH del campo )

BINT20 ( #x )

BINT36 ( #y: 36 para fila 4/6 )

BINT24 ( #w )

BINT8 ( #h: 8 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT0 ( #Decompile: 0=NoDesc No1=SysF ) ( $ #[ME] ) ( $ )

"Choose baud rate" ( Ayuda )

{ "2400" "4800" "9600" "14400" "19200" "38400" "57600" "115200" } ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

"115200" ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* Campo 7: Parity

'DROPFALSE ( MH del campo )

BINT74 ( #x )

BINT36 ( #y: 36 para fila 4/6 )

BINT24 ( #w )

BINT8 ( #h: 8 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT0 ( #Decompile: 0=NoDesc No1=SysF ) ( $ #[ME] ) ( $ )

"Choose parity" ( Ayuda )

{ "None" "Odd" "Even" "Mark" "Space" } ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

"None" ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* Campo 8: OvrW

'DROPFALSE ( MH del campo )

BINT104 ( #x: mayor a 0 en CHECKBOX )

BINT36 ( #y: 36 para fila 4/6 )

MINUSONE ( #w: MINUSONE siempre en campo CHECKBOX )

MINUSONE ( #h: MINUSONE siempre en campo CHECKBOX )

BINT32 ( #TipoDeCampo: CHECKBOX )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHECKBOX )

MINUSONE ( #Decompile: MINUSONE siempre en campo CHECKBOX )

"Overwrite existing variables?" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo CHECKBOX )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

TRUE ( ValorReset : campo CHECKBOX activado )

TRUE ( ValorInicial: campo CHECKBOX activado )

BINT9 ( #Netiq )

BINT9 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es enviado durante el inicio del IfMain para

\* fijar el número del campo que primero tendrá el enfoque

BINT7 #=casedrop ( F ) ( #0 -> #c flag )

:: ( #0 )

DROP ( )

TWO ( #2 )

TRUE ( #2 T )

;

\* Este mensaje es llamado antes del inicio del IfMain para

\* cambiar las 3 últimas teclas de la primera fila del menú.

BINT12 #=casedrop ( F ) ( -> {F4 F5 F6} T // F )

:: ( )

{ { "RECV" DoBadKey }

{ "KGET" DoBadKey }

{ "SEND" DoBadKey }

} ( {} )

TRUE ( {} T )

;

\* Fin del Message handler del formulario:

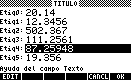
DROPFALSE

;

"TRANSFER" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 ... ob9 T // F )

;

Ejemplo 2 IfMain

**Un formulario de entrada con 6 campos**

En este ejemplo se muestra un formulario de entrada con 6 campos **TEXTO** en una columna.

Las etiquetas tienen sus posiciones #x en la ubicación 0.

Los campos tienen sus posiciones #y en las ubicaciones 9, 18, 27, 36, 45 y 54.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMain1COL ( -> ob1 ob2 ob3 ob4 ob5 ob6 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq0:" 0 10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Etiq1:" 0 19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Etiq2:" 0 28 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Etiq3:" 0 37 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Etiq4:" 0 46 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Etiq5:" 0 55 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT105 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT105 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

12.3456 ( ValorInicial )

\* CAMPO 2

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT27 ( #y: 27 para fila 3/6 )

BINT105 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

502.367 ( ValorInicial )

\* CAMPO 3

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT36 ( #y: 36 para fila 4/6 )

BINT105 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

111.2561 ( ValorInicial )

\* CAMPO 4

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT45 ( #y: 45 para fila 5/6 )

BINT105 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

87.25948 ( ValorInicial )

\* CAMPO 5

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT54 ( #y: 54 para fila 6/6 )

BINT105 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

19.356 ( ValorInicial )

BINT6 ( #Netiq )

BINT6 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 ob4 ob5 ob6 T // F )

;

Ejemplo 3 IfMain

**Un formulario de entrada con 12 campos en 2 columnas**

En este ejemplo se muestra un formulario de entrada con 12 campos **TEXTO** en 2 columnas.

Las etiquetas tienen sus posiciones #x en las ubicaciones 0 y 67.

Los campos tienen sus posiciones #y en las ubicaciones 9, 18, 27, 36, 45 y 54.

Se ha colocado una línea vertical (grob de ancho 1) como etiqueta en la posición x=65, y=8

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMain2COL ( -> ob1 ob2 ob3 ... ob12 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq0:" 0 10 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 )

"Etiq1:" 67 10 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 )

"Etiq2:" 0 19 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 )

"Etiq3:" 67 19 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 )

"Etiq4:" 0 28 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 )

"Etiq5:" 67 28 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 )

"Etiq6:" 0 37 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 )

"Etiq7:" 67 37 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 )

"Etiq8:" 0 46 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 )

"Etiq9:" 67 46 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 )

"Eti10:" 0 55 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 )

"Eti11:" 67 55 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 )

BINT54 BINT1 MAKEGROB INVGROB 65 8 ( #xetiq=65 ) ( #yetiq=8 )

\* CAMPO 0

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

12.345 ( ValorInicial )

\* CAMPO 2

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

502.36 ( ValorInicial )

\* CAMPO 3

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

111.25 ( ValorInicial )

\* CAMPO 4

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT27 ( #y: 27 para fila 3/6 )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

87.259 ( ValorInicial )

\* CAMPO 5

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT27 ( #y: 27 para fila 3/6 )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

19.356 ( ValorInicial )

\* CAMPO 6

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT36 ( #y: 36 para fila 4/6 )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

15.23 ( ValorInicial )

\* CAMPO 7

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT36 ( #y: 36 para fila 4/6 )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

78.259 ( ValorInicial )

\* CAMPO 8

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT45 ( #y: 45 para fila 5/6 )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

47.251 ( ValorInicial )

\* CAMPO 9

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT45 ( #y: 45 para fila 5/6 )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

36.982 ( ValorInicial )

\* CAMPO 10

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT54 ( #y: 54 para fila 6/6 )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

44.512 ( ValorInicial )

\* CAMPO 11

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT54 ( #y: 54 para fila 6/6 )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

77.98 ( ValorInicial )

BINT13 ( #Netiq )

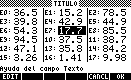
BINT12 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 ... ob12 T // F )

;

Ejemplo 4 IfMain

**Un formulario de entrada con 18 campos en 3 columnas**

En este ejemplo se muestra un formulario de entrada con 18 campos **TEXTO** en 3 columnas.

Las etiquetas tienen sus posiciones #x en las ubicaciones 0 y 67.

Los campos tienen sus posiciones #y en las ubicaciones 9, 18, 27, 36, 45 y 54.

Se han colocado dos líneas verticales (grobs de ancho 1) como etiquetas en las posiciones (43,8) y (87,8)

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMain3COL ( -> ob1 ob2 ob3 ... ob18 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"E0:" 0 10 ( #xetiq=0 COL1/3 ) ( #yetiq=#ycampo+1 )

"E1:" 45 10 ( #xetiq=45 COL2/3 ) ( #yetiq=#ycampo+1 )

"E2:" 89 10 ( #xetiq=89 COL3/3 ) ( #yetiq=#ycampo+1 )

"E3:" 0 19 ( #xetiq=0 COL1/3 ) ( #yetiq=#ycampo+1 )

"E4:" 45 19 ( #xetiq=45 COL2/3 ) ( #yetiq=#ycampo+1 )

"E5:" 89 19 ( #xetiq=89 COL3/3 ) ( #yetiq=#ycampo+1 )

"E6:" 0 28 ( #xetiq=0 COL1/3 ) ( #yetiq=#ycampo+1 )

"E7:" 45 28 ( #xetiq=45 COL2/3 ) ( #yetiq=#ycampo+1 )

"E8:" 89 28 ( #xetiq=89 COL3/3 ) ( #yetiq=#ycampo+1 )

"E9:" 0 37 ( #xetiq=0 COL1/3 ) ( #yetiq=#ycampo+1 )

"10:" 45 37 ( #xetiq=45 COL2/3 ) ( #yetiq=#ycampo+1 )

"11:" 89 37 ( #xetiq=89 COL3/3 ) ( #yetiq=#ycampo+1 )

"12:" 0 46 ( #xetiq=0 COL1/3 ) ( #yetiq=#ycampo+1 )

"13:" 45 46 ( #xetiq=45 COL2/3 ) ( #yetiq=#ycampo+1 )

"14:" 89 46 ( #xetiq=89 COL3/3 ) ( #yetiq=#ycampo+1 )

"15:" 0 55 ( #xetiq=0 COL1/3 ) ( #yetiq=#ycampo+1 )

"16:" 45 55 ( #xetiq=45 COL2/3 ) ( #yetiq=#ycampo+1 )

"17:" 89 55 ( #xetiq=89 COL3/3 ) ( #yetiq=#ycampo+1 )

BINT54 BINT1 MAKEGROB INVGROB 43 8 ( #xetiq=65 ) ( #yetiq=8 )

BINT54 BINT1 MAKEGROB INVGROB 87 8 ( #xetiq=65 ) ( #yetiq=8 )

\* CAMPO 0

'DROPFALSE ( MH del campo )

BINT14 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT28 ( #w: 42-#x para columna 1/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

36.5 ( ValorInicial )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT59 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT27 ( #w: 86-#x para columna 2/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

15.2 ( ValorInicial )

\* CAMPO 2

'DROPFALSE ( MH del campo )

BINT103 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT28 ( #w: 131-#x para columna 3/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

78.5 ( ValorInicial )

\* CAMPO 3

'DROPFALSE ( MH del campo )

BINT14 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT28 ( #w: 42-#x para columna 1/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

39.8 ( ValorInicial )

\* CAMPO 4

'DROPFALSE ( MH del campo )

BINT59 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT27 ( #w: 86-#x para columna 2/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

42.9 ( ValorInicial )

\* CAMPO 5

'DROPFALSE ( MH del campo )

BINT103 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT28 ( #w: 131-#x para columna 3/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

44.9 ( ValorInicial )

\* CAMPO 6

'DROPFALSE ( MH del campo )

BINT14 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT27 ( #y: 27 para fila 3/6 )

BINT28 ( #w: 42-#x para columna 1/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

54.3 ( ValorInicial )

\* CAMPO 7

'DROPFALSE ( MH del campo )

BINT59 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT27 ( #y: 27 para fila 3/6 )

BINT27 ( #w: 86-#x para columna 2/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

17.7 ( ValorInicial )

\* CAMPO 8

'DROPFALSE ( MH del campo )

BINT103 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT27 ( #y: 27 para fila 3/6 )

BINT28 ( #w: 131-#x para columna 3/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

85.5 ( ValorInicial )

\* CAMPO 9

'DROPFALSE ( MH del campo )

BINT14 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT36 ( #y: 36 para fila 4/6 )

BINT28 ( #w: 42-#x para columna 1/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

94.5 ( ValorInicial )

\* CAMPO 10

'DROPFALSE ( MH del campo )

BINT59 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT36 ( #y: 36 para fila 4/6 )

BINT27 ( #w: 86-#x para columna 2/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

37.5 ( ValorInicial )

\* CAMPO 11

'DROPFALSE ( MH del campo )

BINT103 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT36 ( #y: 36 para fila 4/6 )

BINT28 ( #w: 131-#x para columna 3/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

38.7 ( ValorInicial )

\* CAMPO 12

'DROPFALSE ( MH del campo )

BINT14 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT45 ( #y: 45 para fila 5/6 )

BINT28 ( #w: 42-#x para columna 1/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

47.1 ( ValorInicial )

\* CAMPO 13

'DROPFALSE ( MH del campo )

BINT59 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT45 ( #y: 45 para fila 5/6 )

BINT27 ( #w: 86-#x para columna 2/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

35.8 ( ValorInicial )

\* CAMPO 14

'DROPFALSE ( MH del campo )

BINT103 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT45 ( #y: 45 para fila 5/6 )

BINT28 ( #w: 131-#x para columna 3/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

84.2 ( ValorInicial )

\* CAMPO 15

'DROPFALSE ( MH del campo )

BINT14 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT54 ( #y: 45 para fila 6/6 )

BINT28 ( #w: 42-#x para columna 1/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

3.26 ( ValorInicial )

\* CAMPO 16

'DROPFALSE ( MH del campo )

BINT59 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT54 ( #y: 45 para fila 6/6 )

BINT27 ( #w: 86-#x para columna 2/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

1.41 ( ValorInicial )

\* CAMPO 17

'DROPFALSE ( MH del campo )

BINT103 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT54 ( #y: 45 para fila 6/6 )

BINT28 ( #w: 131-#x para columna 3/3 )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

9.98 ( ValorInicial )

BINT20 ( #Netiq )

BINT18 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 ... ob18 T // F )

;

Ejemplo 5 Ifmain

**Título inverso en IfMain**

En este ejemplo el parámetro **Titulo** del formulario de entrada es un grob inverso (debe ser del tamaño exacto de 131x7).

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME TituloInverso ( -> ob T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* ETIQUETA

"Etiqueta:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO TEXTO

'DROPFALSE ( MH del campo )

BINT38 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT93 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

20.14 ( ValorInicial )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" TITULO->GROB131x7\_INVERSA ( Titulo del IfMain )

FLASHPTR IfMain ( ob T // F )

;

\*\*\* Dada una cadena o bint la pone en un grob de

\*\*\* tamaño 131 x 7 con fondo oscuro en la pila

NULLNAME TITULO->GROB131x7\_INVERSA ( $/# -> grob131x7 )

:: ( $/# )

DUPTYPEBINT? ( $/# flag )

IT JstGETTHEMSG

( $ )

BINT1 BINT32 SUB$ ( $ ) ( corta la cadena si es mayor a 32 caract )

BINT7 BINT131 ( $ 7 131 )

MAKEGROB ( $ grob131x7\_blanco )

BINT33 ( $ grob131x7\_blanco 33 )

3PICK ( $ grob131x7\_blanco 33 $ )

LEN$ ( $ grob131x7\_blanco 33 #w )

#-#2/ ( $ grob131x7\_blanco #[33-w]/2 )

Blank$ ( $ grob131x7\_blanco $' )

ROT ( grob131x7\_blanco $' $ )

&$ ( grob131x7\_blanco $'' )

$>grob ( grob131x7\_blanco grob' )

ONEONE ( grob131x7\_blanco grob' #1 #1 )

Repl ( grob131x7 ) ( Copia grb2 en grb1 en modo REPLACE )

INVGROB ( grob131x7\_inversa )

\* Lo siguiente es sólo para redondear las esquinas del grob

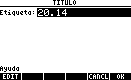
ZEROZERO PixonW ( grob131x7\_inversa )

BINT130 BINT0 PixonW ( grob131x7\_inversa )

BINT0 BINT6 PixonW ( grob131x7\_inversa )

BINT130 BINT6 PixonW ( grob131x7\_inversa )

;

Ejemplo 6 Ifmain

**Título subrayado en IfMain**

En este ejemplo el parámetro **Titulo** del formulario de entrada es un grob con una línea debajo (debe ser del tamaño exacto de 131x7).

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME TituloSubrayado ( -> ob T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* ETIQUETA

"Etiqueta:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO TEXTO

'DROPFALSE ( MH del campo )

BINT38 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT93 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

20.14 ( ValorInicial )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" TITULO->GROB131x7\_SUBRAYADO ( Titulo del IfMain )

FLASHPTR IfMain ( ob T // F )

;

\*\*\* Dada una cadena o bint la pone en un grob de

\*\*\* tamaño 131 x 7 con una línea horizontal debajo

NULLNAME TITULO->GROB131x7\_SUBRAYADO ( $/# -> grob131x7 )

:: ( $/# )

DUPTYPEBINT? ( $/# flag )

IT JstGETTHEMSG

( $ )

BINT1 BINT32 SUB$ ( $ ) ( corta la cadena si es mayor a 32 caract )

BINT7 BINT131 ( $ 7 131 )

MAKEGROB ( $ grob131x7\_blanco )

BINT33 ( $ grob131x7\_blanco 33 )

3PICK ( $ grob131x7\_blanco 33 $ )

LEN$ ( $ grob131x7\_blanco 33 #w )

#-#2/ ( $ grob131x7\_blanco #[33-w]/2 )

Blank$ ( $ grob131x7\_blanco $' )

ROT ( grob131x7\_blanco $' $ )

&$ ( grob131x7\_blanco $'' )

$>grob ( grob131x7\_blanco grob' )

BINT1

BINT0 ( grob131x7\_blanco grob' #1 #0 )

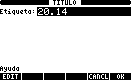
Repl ( grob131x7 ) ( Copia grb2 en grb1 en modo REPLACE )

BINT0 BINT6

BINT130 BINT6

LineB ( grob131x7 ) ( Dibuja una línea negra en el grob )

;

Ejemplo 7 Ifmain

**Título mostrado más arriba en IfMain**

En este ejemplo el parámetro **Titulo** del formulario de entrada es un grob donde el titulo se muestra un píxel mas arriba (debe ser del tamaño exacto de 131x7), lo cual permite mostrar los campos 1 píxel más arriba.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME TituloArriba ( -> ob T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* ETIQUETA

"Etiqueta:" BINT0 BINT9 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO TEXTO

'DROPFALSE ( MH del campo )

BINT38 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT8 ( #y )

BINT93 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

20.14 ( ValorInicial )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" TITULO->GROB131x7\_ARRIBA ( Titulo del IfMain )

FLASHPTR IfMain ( ob T // F )

;

\*\*\* Dada una cadena o bint la pone en un grob de

\*\*\* tamaño 131 x 7 en la parte superior

NULLNAME TITULO->GROB131x7\_ARRIBA ( $/# -> grob131x7 )

:: ( $/# )

DUPTYPEBINT? ( $/# flag )

IT JstGETTHEMSG

( $ )

BINT1 BINT30 SUB$ ( $ ) ( corta la cadena si es mayor a 30 caract )

SPACE$ SWAP&$ ( $ ) ( agrega espacio al inicio de la cadena )

APPEND\_SPACE ( $ ) ( agrega espacio al final de la cadena )

BINT7 BINT131 ( $ 7 131 )

MAKEGROB ( $ grob131x7\_blanco )

0 0 131 5 ( $ grob131x7\_blanco 0 0 131 5 )

FBoxB ( $ grob131x7 ) ( Dibuja rectángulo negro relleno )

OVER ( $ grob131x7 $ )

$>grob ( $ grob131x7 grob )

BINT131 ( $ grob131x7 grob 131 )

BINT4 ( $ grob131x7 grob 131 4 )

5ROLL ( grob131x7 grob 131 4 $ )

LEN$ ( grob131x7 grob 131 4 #ncaract )

#\* ( grob131x7 grob 131 #4·ncaract )

#- ( grob131x7 grob #131-4·ncaract )

#2/ ( grob131x7 grob #[131-4·ncaract]/2 )

BINT0 ( grob131x7 grob #[131-4·ncaract]/2 0 )

Repl ( grob131x7 ) ( Copia grb2 en grb1 en modo REPLACE )

\* Lo siguiente es sólo para redondear las esquinas del grob

ZEROZERO PixonW ( grob131x7\_inversa )

BINT130 BINT0 PixonW ( grob131x7\_inversa )

BINT0 BINT4 PixonW ( grob131x7\_inversa )

BINT130 BINT4 PixonW ( grob131x7\_inversa )

;

Ejemplo 8 IfMain

**Un campo motrado con Minifuente**

En este ejemplo se muestra un formulario de entrada con 1 campo **TEXTO** mostrado en Minifuente.

Para esto, el parámetro **#Decompile** del campo debe contener al BINT1.  
En este ejemplo, el parámetro **#Decompile** del campo es BINT5 (5=4+1), donde el bint 4 indica que el número será representado en formato estándar y el bint 1 indica que el campo se mostrará en Minifuente.

Observa que cuando un campo se muestra en Minifuente, la posición **#y** de la etiqueta es igual a la posición **#y** del campo correspondiente.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMainMinif ( -> ob T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq0:" 0 9 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

\* CAMPO 0

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y )

BINT105 ( #w: 131-#x para 1 columna )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

20.14 ( ValorInicial )

BINT1 ( #Netiq )

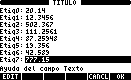
BINT1 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob T // F )

;

Ejemplo 9 IfMain

**Un formulario de entrada con 8 campos en Minifuente**

En este ejemplo se muestra un formulario de entrada con 8 campos **TEXTO** en una columna.

Las etiquetas tienen sus posiciones #x en la ubicación 0.

Los campos tienen sus posiciones #y en las ubicaciones 8, 15, 22, 29, 36, 43, 50 y 57.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMain1ColMini ( -> ob1 ob2 ob3 ob4 ob5 ob6 ob7 ob8 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq0:" 0 8 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

"Etiq1:" 0 15 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

"Etiq2:" 0 22 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

"Etiq3:" 0 29 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

"Etiq4:" 0 36 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

"Etiq5:" 0 43 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

"Etiq6:" 0 50 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

"Etiq7:" 0 57 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

\* CAMPO 0

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT8 ( #y: 8 para fila 1/8 MiniF )

BINT105 ( #w: 131-#x para 1 columna )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT15 ( #y: 15 para fila 2/8 MiniF )

BINT105 ( #w: 131-#x para 1 columna )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

12.3456 ( ValorInicial )

\* CAMPO 2

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT22 ( #y: 22 para fila 3/8 MiniF )

BINT105 ( #w: 131-#x para 1 columna )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

502.367 ( ValorInicial )

\* CAMPO 3

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT29 ( #y: 29 para fila 4/8 MiniF )

BINT105 ( #w: 131-#x para 1 columna )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

111.2561 ( ValorInicial )

\* CAMPO 4

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT36 ( #y: 36 para fila 5/8 MiniF )

BINT105 ( #w: 131-#x para 1 columna )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

87.25948 ( ValorInicial )

\* CAMPO 5

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT43 ( #y: 43 para fila 6/8 MiniF )

BINT105 ( #w: 131-#x para 1 columna )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

19.356 ( ValorInicial )

\* CAMPO 6

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT50 ( #y: 50 para fila 7/8 MiniF )

BINT105 ( #w: 131-#x para 1 columna )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

42.589 ( ValorInicial )

\* CAMPO 7

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT57 ( #y: 57 para fila 8/8 MiniF )

BINT105 ( #w: 131-#x para 1 columna )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

777.15 ( ValorInicial )

BINT8 ( #Netiq )

BINT8 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" TITULO->GROB131x7\_ARRIBA ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 ob4 ob5 ob6 ob7 ob8 T // F )

;

\*\*\* Dada una cadena o bint la pone en un grob de

\*\*\* tamaño 131 x 7 en la parte superior

NULLNAME TITULO->GROB131x7\_ARRIBA ( $/# -> grob131x7 )

:: ( $/# )

DUPTYPEBINT? ( $/# flag )

IT JstGETTHEMSG

( $ )

BINT1 BINT30 SUB$ ( $ ) ( corta la cadena si es mayor a 30 caract )

SPACE$ SWAP&$ ( $ ) ( agrega espacio al inicio de la cadena )

APPEND\_SPACE ( $ ) ( agrega espacio al final de la cadena )

BINT7 BINT131 ( $ 7 131 )

MAKEGROB ( $ grob131x7\_blanco )

0 0 131 5 ( $ grob131x7\_blanco 0 0 131 5 )

FBoxB ( $ grob131x7 ) ( Dibuja rectángulo negro relleno )

OVER ( $ grob131x7 $ )

$>grob ( $ grob131x7 grob )

BINT131 ( $ grob131x7 grob 131 )

BINT4 ( $ grob131x7 grob 131 4 )

5ROLL ( grob131x7 grob 131 4 $ )

LEN$ ( grob131x7 grob 131 4 #ncaract )

#\* ( grob131x7 grob 131 #4·ncaract )

#- ( grob131x7 grob #131-4·ncaract )

#2/ ( grob131x7 grob #[131-4·ncaract]/2 )

BINT0 ( grob131x7 grob #[131-4·ncaract]/2 0 )

Repl ( grob131x7 ) ( Copia grb2 en grb1 en modo REPLACE )

\* Lo siguiente es sólo para redondear las esquinas del grob

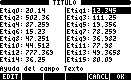
ZEROZERO PixonW ( grob131x7\_inversa )

BINT130 BINT0 PixonW ( grob131x7\_inversa )

BINT0 BINT4 PixonW ( grob131x7\_inversa )

BINT130 BINT4 PixonW ( grob131x7\_inversa )

;

Ejemplo 10 IfMain

**Un formulario de entrada con 16 campos en 2 columnas en Minifuente**

En este ejemplo se muestra un formulario de entrada con 16 campos **TEXTO** en 2 columnas.

Las etiquetas tienen sus posiciones #x en las ubicaciones 0 y 67.

Los campos tienen sus posiciones #y en las ubicaciones 8, 15, 22, 29, 36, 43, 50 y 57.

Se ha colocado una línea vertical (grob de ancho 1) como etiqueta en la posición x=65, y=7

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMain2ColMini ( -> ob1 ob2 ob3 ... ob16 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq0:" 0 8 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo en MiniF )

"Etiq1:" 67 8 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo en MiniF )

"Etiq2:" 0 15 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo en MiniF )

"Etiq3:" 67 15 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo en MiniF )

"Etiq4:" 0 22 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo en MiniF )

"Etiq5:" 67 22 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo en MiniF )

"Etiq6:" 0 29 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo en MiniF )

"Etiq7:" 67 29 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo en MiniF )

"Etiq8:" 0 36 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo en MiniF )

"Etiq9:" 67 36 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo en MiniF )

"Eti10:" 0 43 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo en MiniF )

"Eti11:" 67 43 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo en MiniF )

"Eti12:" 0 50 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo en MiniF )

"Eti13:" 67 50 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo en MiniF )

"Eti14:" 0 57 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo en MiniF )

"Eti15:" 67 57 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo en MiniF )

BINT56 BINT1 MAKEGROB INVGROB 65 7 ( #xetiq=65 ) ( #yetiq=7 )

\* CAMPO 0

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT8 ( #y: 8 para fila 1/8 MiniF )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT8 ( #y: 8 para fila 1/8 MiniF )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

12.345 ( ValorInicial )

\* CAMPO 2

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT15 ( #y: 15 para fila 2/8 MiniF )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

502.36 ( ValorInicial )

\* CAMPO 3

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT15 ( #y: 15 para fila 2/8 MiniF )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

111.25 ( ValorInicial )

\* CAMPO 4

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT22 ( #y: 22 para fila 3/8 MiniF )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

87.259 ( ValorInicial )

\* CAMPO 5

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT22 ( #y: 22 para fila 3/8 MiniF )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

19.356 ( ValorInicial )

\* CAMPO 6

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT29 ( #y: 29 para fila 4/8 MiniF )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

15.23 ( ValorInicial )

\* CAMPO 7

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT29 ( #y: 29 para fila 4/8 MiniF )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

78.259 ( ValorInicial )

\* CAMPO 8

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT36 ( #y: 36 para fila 5/8 MiniF )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

47.251 ( ValorInicial )

\* CAMPO 9

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT36 ( #y: 36 para fila 5/8 MiniF )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

36.982 ( ValorInicial )

\* CAMPO 10

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT43 ( #y: 43 para fila 6/8 MiniF )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

44.512 ( ValorInicial )

\* CAMPO 11

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT43 ( #y: 43 para fila 6/8 MiniF )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

77.98 ( ValorInicial )

\* CAMPO 12

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT50 ( #y: 50 para fila 7/8 MiniF )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

777.369 ( ValorInicial )

\* CAMPO 13

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT50 ( #y: 50 para fila 7/8 MiniF )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

49.3658 ( ValorInicial )

\* CAMPO 14

'DROPFALSE ( MH del campo )

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT57 ( #y: 57 para fila 8/8 MiniF )

BINT38 ( #w: 64-#x para columna 1/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

36.25 ( ValorInicial )

\* CAMPO 15

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT57 ( #y: 57 para fila 8/8 MiniF )

BINT38 ( #w: 131-#x para columna 2/2 )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

80.09 ( ValorInicial )

BINT17 ( #Netiq )

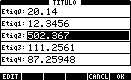
BINT16 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" TITULO->GROB131x7\_ARRIBA ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 ... ob16 T // F )

;

Ejemplo 11 IfMain

**Un formulario de entrada con 5 campos y con líneas que separan los campos**

En este ejemplo se muestra un formulario de entrada con 5 campos **TEXTO** en una columna.

Además se muestran líneas que separan a los campos.

Las etiquetas tienen sus posiciones #x en la ubicación 2.

Las líneas se han colocado como 7 etiquetas adicionales.

Los campos tienen sus posiciones #y en las ubicaciones 8, 20, 32, 44 y 56.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMain1COLRayas ( -> ob1 ob2 ob3 ob4 ob5 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq0:" 2 9 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Etiq1:" 2 21 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Etiq2:" 2 33 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Etiq3:" 2 45 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Etiq4:" 2 57 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

58 1 MAKEGROB INVGROB 0 7 ( Línea vertical izquierda )

58 1 MAKEGROB INVGROB 130 7 ( Línea vertical derecha )

1 131 MAKEGROB INVGROB 0 17

1 131 MAKEGROB INVGROB 0 29

1 131 MAKEGROB INVGROB 0 41

1 131 MAKEGROB INVGROB 0 53

1 131 MAKEGROB INVGROB 0 65

\* CAMPO 0

'DROPFALSE ( MH del campo )

BINT28 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT8 ( #y: 8 para fila 1/5 CON RAYAS )

BINT101 ( #w: 129-#x para 1 columna LIMITADA )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT28 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT20 ( #y: 20 para fila 2/5 CON RAYAS )

BINT101 ( #w: 129-#x para 1 columna LIMITADA )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

12.3456 ( ValorInicial )

\* CAMPO 2

'DROPFALSE ( MH del campo )

BINT28 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT32 ( #y: 32 para fila 3/5 CON RAYAS )

BINT101 ( #w: 129-#x para 1 columna LIMITADA )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

502.367 ( ValorInicial )

\* CAMPO 3

'DROPFALSE ( MH del campo )

BINT28 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT44 ( #y: 44 para fila 4/5 CON RAYAS )

BINT101 ( #w: 129-#x para 1 columna LIMITADA )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

111.2561 ( ValorInicial )

\* CAMPO 4

'DROPFALSE ( MH del campo )

BINT28 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT56 ( #y: 56 para fila 5/5 CON RAYAS )

BINT101 ( #w: 129-#x para 1 columna LIMITADA )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

87.25948 ( ValorInicial )

BINT12 ( #Netiq )

BINT5 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" TITULO->GROB131x7\_ARRIBA2 ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 ob4 ob5 T // F )

;

\*\*\* Dada una cadena o bint la pone en un grob de

\*\*\* tamaño 131 x 7 en la parte superior

NULLNAME TITULO->GROB131x7\_ARRIBA2 ( $/# -> grob131x7 )

:: ( $/# )

DUPTYPEBINT? ( $/# flag )

IT JstGETTHEMSG

( $ )

BINT1 BINT30 SUB$ ( $ ) ( corta la cadena si es mayor a 30 caract )

SPACE$ SWAP&$ ( $ ) ( agrega espacio al inicio de la cadena )

APPEND\_SPACE ( $ ) ( agrega espacio al final de la cadena )

BINT7 BINT131 ( $ 7 131 )

MAKEGROB ( $ grob131x7\_blanco )

0 0 131 5 ( $ grob131x7\_blanco 0 0 131 5 )

FBoxB ( $ grob131x7 ) ( Dibuja rectángulo negro relleno )

OVER ( $ grob131x7 $ )

$>grob ( $ grob131x7 grob )

BINT131 ( $ grob131x7 grob 131 )

BINT4 ( $ grob131x7 grob 131 4 )

5ROLL ( grob131x7 grob 131 4 $ )

LEN$ ( grob131x7 grob 131 4 #ncaract )

#\* ( grob131x7 grob 131 #4·ncaract )

#- ( grob131x7 grob #131-4·ncaract )

#2/ ( grob131x7 grob #[131-4·ncaract]/2 )

BINT0 ( grob131x7 grob #[131-4·ncaract]/2 0 )

Repl ( grob131x7 ) ( Copia grb2 en grb1 en modo REPLACE )

\* Lo siguiente es para redondear las esquinas superiores del grob

ZEROZERO PixonW ( grob131x7\_inversa )

BINT130 BINT0 PixonW ( grob131x7\_inversa )

\* Lo siguiente es para pintar las esquinas inferiores del grob

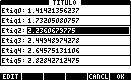
BINT0 BINT5 PixonB ( grob131x7\_inversa )

BINT130 BINT5 PixonB ( grob131x7\_inversa )

BINT0 BINT6 PixonB ( grob131x7\_inversa )

BINT130 BINT6 PixonB ( grob131x7\_inversa )

;

Ejemplo 12 IfMain

**Un formulario de entrada con 6 campos en Minifuente y con líneas que separan los campos**

En este ejemplo se muestra un formulario de entrada con 6 campos **TEXTO** en Minifuente en una columna.

Además se muestran líneas que separan a los campos.

Las etiquetas tienen sus posiciones #x en la ubicación 2.

Las líneas se han colocado como 8 etiquetas adicionales.

Los campos tienen sus posiciones #y en las ubicaciones 8, 18, 28, 38, 48 y 58.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMa1ColMiniRay ( -> ob1 ob2 ob3 ob4 ob5 ob6 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq0:" 2 8 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

"Etiq1:" 2 18 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

"Etiq2:" 2 28 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

"Etiq3:" 2 38 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

"Etiq4:" 2 48 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

"Etiq5:" 2 58 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo en MiniF )

58 1 MAKEGROB INVGROB 0 7 ( Línea vertical izquierda )

58 1 MAKEGROB INVGROB 130 7 ( Línea vertical derecha )

1 131 MAKEGROB INVGROB 0 15

1 131 MAKEGROB INVGROB 0 25

1 131 MAKEGROB INVGROB 0 35

1 131 MAKEGROB INVGROB 0 45

1 131 MAKEGROB INVGROB 0 55

1 131 MAKEGROB INVGROB 0 65

\* CAMPO 0

'DROPFALSE ( MH del campo )

BINT28 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT8 ( #y: 8 para fila 1/6 MiniF CON RAYAS )

BINT101 ( #w: 129-#x para 1 columna LIMITADA )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT28 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 MiniF CON RAYAS )

BINT101 ( #w: 129-#x para 1 columna LIMITADA )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

12.3456 ( ValorInicial )

\* CAMPO 2

'DROPFALSE ( MH del campo )

BINT28 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT28 ( #y: 28 para fila 3/6 MiniF CON RAYAS )

BINT101 ( #w: 129-#x para 1 columna LIMITADA )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

502.367 ( ValorInicial )

\* CAMPO 3

'DROPFALSE ( MH del campo )

BINT28 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT38 ( #y: 38 para fila 4/6 MiniF CON RAYAS )

BINT101 ( #w: 129-#x para 1 columna LIMITADA )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

111.2561 ( ValorInicial )

\* CAMPO 4

'DROPFALSE ( MH del campo )

BINT28 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT48 ( #y: 48 para fila 5/6 MiniF CON RAYAS )

BINT101 ( #w: 129-#x para 1 columna LIMITADA )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

87.25948 ( ValorInicial )

\* CAMPO 5

'DROPFALSE ( MH del campo )

BINT28 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT58 ( #y: 58 para fila 6/6 MiniF CON RAYAS )

BINT101 ( #w: 129-#x para 1 columna LIMITADA )

BINT6 ( #h: 6 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT5 ( #Decompile: 4=STD 1=MiniF ) ( ) ( "$" 'id' 'u' )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

19.356 ( ValorInicial )

BINT14 ( #Netiq )

BINT6 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" TITULO->GROB131x7\_ARRIBA2 ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 ob4 ob5 ob6 T // F )

;

\*\*\* Dada una cadena o bint la pone en un grob de

\*\*\* tamaño 131 x 7 en la parte superior

NULLNAME TITULO->GROB131x7\_ARRIBA2 ( $/# -> grob131x7 )

:: ( $/# )

DUPTYPEBINT? ( $/# flag )

IT JstGETTHEMSG

( $ )

BINT1 BINT30 SUB$ ( $ ) ( corta la cadena si es mayor a 30 caract )

SPACE$ SWAP&$ ( $ ) ( agrega espacio al inicio de la cadena )

APPEND\_SPACE ( $ ) ( agrega espacio al final de la cadena )

BINT7 BINT131 ( $ 7 131 )

MAKEGROB ( $ grob131x7\_blanco )

0 0 131 5 ( $ grob131x7\_blanco 0 0 131 5 )

FBoxB ( $ grob131x7 ) ( Dibuja rectángulo negro relleno )

OVER ( $ grob131x7 $ )

$>grob ( $ grob131x7 grob )

BINT131 ( $ grob131x7 grob 131 )

BINT4 ( $ grob131x7 grob 131 4 )

5ROLL ( grob131x7 grob 131 4 $ )

LEN$ ( grob131x7 grob 131 4 #ncaract )

#\* ( grob131x7 grob 131 #4·ncaract )

#- ( grob131x7 grob #131-4·ncaract )

#2/ ( grob131x7 grob #[131-4·ncaract]/2 )

BINT0 ( grob131x7 grob #[131-4·ncaract]/2 0 )

Repl ( grob131x7 ) ( Copia grb2 en grb1 en modo REPLACE )

\* Lo siguiente es para redondear las esquinas superiores del grob

ZEROZERO PixonW ( grob131x7\_inversa )

BINT130 BINT0 PixonW ( grob131x7\_inversa )

\* Lo siguiente es para pintar las esquinas inferiores del grob

BINT0 BINT5 PixonB ( grob131x7\_inversa )

BINT130 BINT5 PixonB ( grob131x7\_inversa )

BINT0 BINT6 PixonB ( grob131x7\_inversa )

BINT130 BINT6 PixonB ( grob131x7\_inversa )

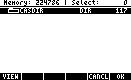
;

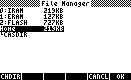
Ejemplo 13 IfMain

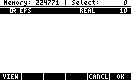
**Un formulario de entrada con un campo CHOOSE que se ha adaptado para que funcione como campo FILER**

En este ejemplo se muestra un formulario de entrada con 1 campo **CHOOSE** que se ha adaptado para que funcione como un campo FILER.

Para eso, se usa al mensaje 17 en el campo. En este ejemplo, sólo se permiten números reales en el campo.



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMaFiler ( -> % T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"Filer:" 0 10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: CAMPO CHOOSE->FILER. CONTIENE UN NUMERO REAL

\* Message handler del campo

' ::

\* Cuando se presiona la tecla CHOOS en un campo CHOOSE o COMBOCHOOSE.

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

BINT17 #=casedrop ( C/F ) ( -> T // F )

:: ( )

Filer\_Home\_S1\_NoDir\_% ( ob T // F )

IT

FLASHPTR IfSetCurrentFieldValue

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT105 ( #w:131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Escoge numero real desde FILER" ( Ayuda )

MINUSONE ( ChooseData: No se necesita en campo CHOOSE->FILER )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

MINUSONE ( ValorReset: campo CHOOSE->FILER en blanco )

MINUSONE ( ValorInicial: campo CHOOSE->FILER en blanco )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( % T // F )

;

\* Busca un NÚMERO REAL en el FILER

\* Inicia en el directorio HOME

\* Se puede desplazar por todos lados

\* No permite selección múltiple

\* Solo retorna un número real seguido de TRUE.

\* Si se cancela, sólo retorna FALSE.

NULLNAME Filer\_Home\_S1\_NoDir\_% ( -> ob T // F )

::

{ # 2933 # 2A96 } ( Filer\_Tipos: Reales y Directorios )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* HAY 6 TECLAS DEL MENU

\* La 1º es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* Esta lista es para quitar la tecla +/- "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT28 ( Tecla\_Atajo: BINT28 = #1C = Tecla +/- "CHECK" )

}

\* Esta lista es para quitar la tecla ALPHA + ENTER "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 133 ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista no hace nada

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

}

\* Esta lista es para la tecla de menú F5 "CANCL"

{ # DF2C ( Nombre: # DF2C "CANCL" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT37 ( Acción: BINT37= Sale del filer dejando FALSE en la pila )

}

\* Esta lista es para la tecla de menú F6 "OK"

\* También es para asignarla a la tecla ENTER

{ # DF2D ( Nombre: # DF2D "OK" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT17 ( Acción: BINT17= Prog Pers que pone Ruta, Objeto y Nombre )

:: ( Ruta Objeto Nombre )

DROPSWAPDROP ( Objeto )

DUPTYPERRP? ( Objeto flag ) ( Evita finalizar con Directorio )

caseDrpBadKy ( )

' TakeOver ( Objeto TakeOver )

; ( Programa\_Extra: Sale del filer, deja en la pila el Objeto y TRUE )

# 33 ( Tecla\_Atajo: BINT51 = #33 = Tecla ENTER )

}

}

FLASHPTR FILER\_MANAGERTYPE ( ob T // F // :0:{} )

TRUE

EQUAL ( ob T // F )

;

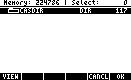
Ejemplo 14 IfMain

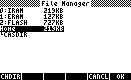
**Un formulario de entrada con un campo COMBOCHOOSE que se ha adaptado para que funcione como campo COMBOFILER**

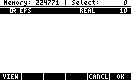
En este ejemplo se muestra un formulario de entrada con 1 campo **COMBOCHOOSE** que se ha adaptado para que funcione como un campo COMBOFILER.

Para eso, se usa al mensaje 17 en el campo. En este ejemplo, sólo se permiten números reales en el campo.

También se usa el mensaje 0 en el formulario para que se puedan usar campos **COMBOCHOOSE**.



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMaComboFiler ( -> % T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"Filer:" 0 10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: CAMPO COMBOCHOOSE->COMBOFILER. CONTIENE UN NUMERO REAL

\* Message handler del campo

' ::

\* Cuando se presiona la tecla CHOOS en un campo CHOOSE o COMBOCHOOSE.

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

BINT17 #=casedrop ( C/F ) ( -> T // F )

:: ( )

Filer\_Home\_S1\_NoDir\_% ( ob T // F )

IT

FLASHPTR IfSetCurrentFieldValue

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT105 ( #w:131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT2 ( #TipoDeCampo: COMBOCHOOSE )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Escoge real desde FILER o escribe" ( Ayuda )

MINUSONE ( ChooseData: No se necesita en campo COMCHOOSE->COMFILER )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

MINUSONE ( ValorReset: campo COMCHOOSE->COMFILER en blanco )

MINUSONE ( ValorInicial: campo COMCHOOSE->COMFILER en blanco )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona alguna tecla

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Permite que se puedan usar campos COMBOCHOOSE

BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

FALSE ( #ct #p F )

LAM 'CurrentField ( #ct #p F #c )

FLASHPTR IfGetFieldType ( #ct #p F #TipoCampo )

#2= ( #ct #p F flag ) ( CampoActual=COMBOCHOOSE? )

NOT?SEMI

( #ct #p F )

R> ( #ct #p F prog )

FLASHPTR 002 0A5 ( #ct #p F $ ) ( Equivale al comando x->H )

"52133" ( #ct #p F $ "52133" ) ( dirección del BINT3 )

"B1133" ( #ct #p F $ "52133" "B1133" ) ( dirección de BINT2 )

FLASHPTR 00F 01A ( #ct #p F $' %3 ) ( Equivale al comando xSREPL )

DROP ( #ct #p F $' )

FLASHPTR 002 0A4 ( #ct #p F prog' ) ( Equivale al comando xH-> )

>R ( #ct #p F )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( % T // F )

;

\* Busca un NÚMERO REAL en el FILER

\* Inicia en el directorio HOME

\* Se puede desplazar por todos lados

\* No permite selección múltiple

\* Solo retorna un número real seguido de TRUE.

\* Si se cancela, sólo retorna FALSE.

NULLNAME Filer\_Home\_S1\_NoDir\_% ( -> ob T // F )

::

{ # 2933 # 2A96 } ( Filer\_Tipos: Reales y Directorios )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* HAY 6 TECLAS DEL MENU

\* La 1º es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* Esta lista es para quitar la tecla +/- "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT28 ( Tecla\_Atajo: BINT28 = #1C = Tecla +/- "CHECK" )

}

\* Esta lista es para quitar la tecla ALPHA + ENTER "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 133 ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista no hace nada

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

}

\* Esta lista es para la tecla de menú F5 "CANCL"

{ # DF2C ( Nombre: # DF2C "CANCL" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT37 ( Acción: BINT37= Sale del filer dejando FALSE en la pila )

}

\* Esta lista es para la tecla de menú F6 "OK"

\* También es para asignarla a la tecla ENTER

{ # DF2D ( Nombre: # DF2D "OK" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT17 ( Acción: BINT17= Prog Pers que pone Ruta, Objeto y Nombre )

:: ( Ruta Objeto Nombre )

DROPSWAPDROP ( Objeto )

DUPTYPERRP? ( Objeto flag ) ( Evita finalizar con Directorio )

caseDrpBadKy ( )

' TakeOver ( Objeto TakeOver )

; ( Programa\_Extra: Sale del filer, deja en la pila el Objeto y TRUE )

# 33 ( Tecla\_Atajo: BINT51 = #33 = Tecla ENTER )

}

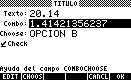
}

FLASHPTR FILER\_MANAGERTYPE ( ob T // F // :0:{} )

TRUE

EQUAL ( ob T // F )

;

Ejemplo 15 IfMain

**Un formulario de entrada con los 4 tipos de campos en IfMain**

En este ejemplo se muestra un formulario de entrada con los cuatro tipos de campos: TEXTO, COMBOCHOOSE, CHOOSE y CHECK.

Para que pueda funcionar el campo COMBOCHOOSE, se altera el código que debe ejecutarse (correspondiente al parámetro AppKeys del POL) cada vez que se presiona una tecla y el campo actual es un COMBOCHOOSE.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMain4Tipos ( -> ob1 ob2 ob3 ob4 T // F )

:: CK0 ( ) ( No se requieren argumentos )

" Texto:" 0 11 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

" Combo:" 0 21 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Choose:" 0 31 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Check" 8 41 ( #xetiq=#xcampo+7 en CHK ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: TEXTO

'DROPFALSE ( MH del campo )

BINT30 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT10 ( #y: 10 para fila 1/5 )

BINT101 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo TEXTO" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 1: COMBOCHOOSE

'DROPFALSE ( MH del campo )

BINT30 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT20 ( #y: 20 para fila 2/5 )

BINT101 ( #w:131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT2 ( #TipoDeCampo: COMBOCHOOSE )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda del campo COMBOCHOOSE" ( Ayuda )

{ 12. 13. 14. 15. } ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

18. ( ValorReset )

19. ( ValorInicial )

\* CAMPO 2: CHOOSE

'DROPFALSE ( MH del campo )

BINT30 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT30 ( #y: 30 para fila 3/5 )

BINT101 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT0 ( #Decompile: 0=NoDesc No1=SysF ) ( $ #[ME] ) ( $ )

"Ayuda del campo CHOOSE" ( Ayuda )

{ "OPCION A" "OPCION B" "OPCION C" } ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

"OPCION A" ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 3: CHECKBOX

'DROPFALSE ( MH del campo )

BINT1 ( #x: mayor a 0 en CHECKBOX )

BINT40 ( #y: 40 para fila 4/5 )

MINUSONE ( #w: MINUSONE siempre en campo CHECKBOX )

MINUSONE ( #h: MINUSONE siempre en campo CHECKBOX )

BINT32 ( #TipoDeCampo: CHECKBOX )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHECKBOX )

MINUSONE ( #Decompile: MINUSONE siempre en campo CHECKBOX )

"Ayuda del campo CHECKBOX" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo CHECKBOX )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

FALSE ( ValorReset : campo CHECKBOX desactivado )

FALSE ( ValorInicial: campo CHECKBOX desactivado )

BINT4 ( #Netiq )

BINT4 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona alguna tecla

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Permite que se puedan usar campos COMBOCHOOSE

BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

FALSE ( #ct #p F )

LAM 'CurrentField ( #ct #p F #c )

FLASHPTR IfGetFieldType ( #ct #p F #TipoCampo )

#2= ( #ct #p F flag ) ( CampoActual=COMBOCHOOSE? )

NOT?SEMI

( #ct #p F )

R> ( #ct #p F prog )

FLASHPTR 002 0A5 ( #ct #p F $ ) ( Equivale al comando x->H )

"52133" ( #ct #p F $ "52133" ) ( dirección del BINT3 )

"B1133" ( #ct #p F $ "52133" "B1133" ) ( dirección de BINT2 )

FLASHPTR 00F 01A ( #ct #p F $' %3 ) ( Equivale al comando xSREPL )

DROP ( #ct #p F $' )

FLASHPTR 002 0A4 ( #ct #p F prog' ) ( Equivale al comando xH-> )

>R ( #ct #p F )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 ob4 T // F )

;

Ejemplo 16 IfMain

**Un campo COMBOCHOOSE que muestre los ítems con una breve explicación (Usando Browser 49)**

En este ejemplo se muestra un campo **COMBOCHOOSE**.

En este campo puedes escribir un valor directamente con el teclado o también escoger un valor desde una lista accesible mediante la tecla de menú CHOOS.

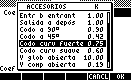
El parámetro **ChooseData** del campo **COMBOCHOOSE** tiene como elementos a listas de la forma:

{ $ ob }

Donde ob es un número real que se fijará como el valor del campo actual, si uno escoge esa opción.

Se llama al mensaje 17 para mostrar cada una de las opciones del parámetro **ChooseData** y al mensaje 0 para que el campo **COMBOCHOOSE** funcione correctamente.

En este ejemplo se usa el browser 49.

** **

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMaComboB49 ( -> % T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

" Coef:" 0 10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: COMBOCHOOSE

\* Message handler del campo

' ::

\* Cuando se presiona la tecla CHOOS en un campo CHOOSE o COMBOCHOOSE.

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

BINT17 #=casedrop ( C/F ) ( -> T // F )

:: ( )

LAM 'CurrentField ( #campo )

FLASHPTR IfGetFieldChooseData ( {}ChooseDATA )

INNERCOMP ( meta )

dup\_ ( meta meta )

DUP ZERO\_DO (DO)

ROLL ( meta{} ... {$ %} )

TWONTHCOMPDROP\_ ( meta{} ... % )

ISTOP@ ( meta{} ... % #nitems )

LOOP ( meta{} meta% )

{}N ( meta{} {%} )

FLASHPTR IfGetCurrentFieldValue ( meta{} {%} %ValorCampo )

EQUALPOSCOMP ( meta{} #pos/#0 )

DUP#0=IT

DROPONE ( meta{} #pos' )

#1- ( meta{} #i )

" ACCESORIOS K" ( meta{} #i $ )

SWAP ( meta{} $ #i )

FLASHPTR Choose2\_ ( {$ %} T // F ) ( BROWSER 49 )

IT

:: TWONTHCOMPDROP\_ ( % )

FLASHPTR IfSetCurrentFieldValue ( )

;

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT105 ( #w:131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT2 ( #TipoDeCampo: COMBOCHOOSE )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Coef de pérdidas locales" ( Ayuda )

{ { "Entrada b agud 0.50" %.5 }

{ "Entr b acampan 0.04" % .04 }

{ "Entr b entrant 1.00" %1 }

{ "Salida a depÓs 1.00" %1 }

{ "Codo a 90° 0.90" % .9 }

{ "Codo a 45° 0.42" % .42 }

{ "Codo curv fuerte 0.75" % .75 }

{ "Codo curv suave 0.60" % .6 }

{ "V glob abierta 10.00" %10 }

{ "V comp abierta 0.19" % .19 }

{ "V check abierta 2.50" % 2.5 } } ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

12. ( ValorReset )

13. ( ValorInicial )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona alguna tecla

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Permite que se puedan usar campos COMBOCHOOSE

BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

FALSE ( #ct #p F )

LAM 'CurrentField ( #ct #p F #c )

FLASHPTR IfGetFieldType ( #ct #p F #TipoCampo )

#2= ( #ct #p F flag ) ( CampoActual=COMBOCHOOSE? )

NOT?SEMI

( #ct #p F )

R> ( #ct #p F prog )

FLASHPTR 002 0A5 ( #ct #p F $ ) ( Equivale al comando x->H )

"52133" ( #ct #p F $ "52133" ) ( dirección del BINT3 )

"B1133" ( #ct #p F $ "52133" "B1133" ) ( dirección de BINT2 )

FLASHPTR 00F 01A ( #ct #p F $' %3 ) ( Equivale al comando xSREPL )

DROP ( #ct #p F $' )

FLASHPTR 002 0A4 ( #ct #p F prog' ) ( Equivale al comando xH-> )

>R ( #ct #p F )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( % T // F )

;

Ejemplo 17 IfMain

**Un campo COMBOCHOOSE que muestre los ítems con una breve explicación (Usando Browser 48)**

Este ejemplo es similar al anterior, pero aquí se usa el browser 48 para mostrar los ítems.

En este ejemplo se muestra un campo **COMBOCHOOSE**.

En este campo puedes escribir un valor directamente con el teclado o también escoger un valor desde una lista accesible mediante la tecla de menú CHOOS.

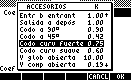
El parámetro **ChooseData** del campo **COMBOCHOOSE** tiene como elementos a listas de la forma:

{ $ ob }

Donde ob es un número real que se fijará como el valor del campo actual, si uno escoge esa opción.

Se llama al mensaje 17 para mostrar cada una de las opciones del parámetro **ChooseData** y al mensaje 0 para que el campo **COMBOCHOOSE** funcione correctamente.

En este ejemplo se usa el browser 48.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMaComboB48 ( -> % T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

" Coef:" 0 10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: COMBOCHOOSE

\* Message handler del campo

' ::

\* Cuando se presiona la tecla CHOOS en un campo CHOOSE o COMBOCHOOSE.

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

BINT17 #=casedrop ( C/F ) ( -> T // F )

:: ( )

'DROPFALSE ( MH )

" ACCESORIOS K" ( MH $ )

BINT17 ( MH $ conv )

LAM 'CurrentField ( MH $ conv #campo )

FLASHPTR IfGetFieldChooseData ( MH $ conv {}ChooseDATA )

DUPINCOMP ( MH $ conv {}ChooseDATA meta{} )

DUP ZERO\_DO (DO)

ROLL ( MH $ conv {}ChooseDATA ... {$ %} )

TWONTHCOMPDROP\_ ( MH $ conv {}ChooseDATA ... % )

ISTOP@ ( MH $ conv {}ChooseDATA ... % #nitems )

LOOP ( MH $ conv {}ChooseDATA meta% )

{}N ( MH $ conv {}ChooseDATA {%} )

FLASHPTR IfGetCurrentFieldValue ( MH $ conv {}ChooseDATA {%} % )

EQUALPOSCOMP ( MH $ conv {}ChooseDATA #pos/#0 )

DUP#0=IT

DROPONE ( MH $ conv {}ChooseDATA #indice )

ROMPTR Choose ( {$ %} T // F ) ( BROWSER 48 )

IT

:: TWONTHCOMPDROP\_ ( % )

FLASHPTR IfSetCurrentFieldValue ( )

;

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT26 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT105 ( #w:131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT2 ( #TipoDeCampo: COMBOCHOOSE )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Coef de pérdidas locales" ( Ayuda )

{ { "Entrada b agud 0.50" %.5 }

{ "Entr b acampan 0.04" % .04 }

{ "Entr b entrant 1.00" %1 }

{ "Salida a depÓs 1.00" %1 }

{ "Codo a 90° 0.90" % .9 }

{ "Codo a 45° 0.42" % .42 }

{ "Codo curv fuerte 0.75" % .75 }

{ "Codo curv suave 0.60" % .6 }

{ "V glob abierta 10.00" %10 }

{ "V comp abierta 0.19" % .19 }

{ "V check abierta 2.50" % 2.5 } } ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

12. ( ValorReset )

13. ( ValorInicial )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona alguna tecla

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Permite que se puedan usar campos COMBOCHOOSE

BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

FALSE ( #ct #p F )

LAM 'CurrentField ( #ct #p F #c )

FLASHPTR IfGetFieldType ( #ct #p F #TipoCampo )

#2= ( #ct #p F flag ) ( CampoActual=COMBOCHOOSE? )

NOT?SEMI

( #ct #p F )

R> ( #ct #p F prog )

FLASHPTR 002 0A5 ( #ct #p F $ ) ( Equivale al comando x->H )

"52133" ( #ct #p F $ "52133" ) ( dirección del BINT3 )

"B1133" ( #ct #p F $ "52133" "B1133" ) ( dirección de BINT2 )

FLASHPTR 00F 01A ( #ct #p F $' %3 ) ( Equivale al comando xSREPL )

DROP ( #ct #p F $' )

FLASHPTR 002 0A4 ( #ct #p F prog' ) ( Equivale al comando xH-> )

>R ( #ct #p F )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( % T // F )

;

Ejemplo 18 Ifmain

**Actualizar el valor de un campo al cambiar el valor de otro campo.**

Para hacer esto debes usar el mensaje número 5 en el message handler de un campo.

Este mensaje es llamado cuando el valor del campo cambia y también al inicio del formulario.

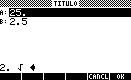
Además debes usar el comando **FLASHPTR IfSetFieldValue** que permite cambiar el valor de cualquier campo de la siguiente manera:

1) Poner el mensaje nº 5 en el campo cuyos cambios afectarán el valor de otros campos.

2) En este debemos obtener el valor(es) que queremos asignar al otro(s) campo(s) y luego usar el comando **FLASHPTR IfSetFieldValue**.

A continuación un ejemplo sencillo con dos campos y dos etiquetas con su respectiva explicación que puedes copiar y pegar en el editor de Debug4x.

Si se actualiza el valor del primer campo, el segundo campo se actualizará de manera automática, fijándose su valor como la décima parte del valor del primer campo.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME ActualizaValor ( -> % % T // F )

:: CK0 ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"B:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: A

\* ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL. CUANDO EL VALOR DE

\* ESTE CAMPO CAMBIA, EL VALOR DEL OTRO CAMPO TAMBIÉN CAMBIARÁ.

\* Message handler del campo

' ::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia

\* Convierte Vext en Vint (será guardado en la pila virtual)

BINT5 #=casedrop ( C ) ( Vext -> Vint flag )

:: ( valor )

DUP ( valor valor )

%10 %/ ( valor valor/10 )

BINT1 ( valor valor/10 #1 )

FLASHPTR IfSetFieldValue ( valor )

TRUE ( valor TRUE )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT2 ( #Decompile: 2=Actual No1=SysF ) ( ) ( "$" 'id' u )

"Escribe un número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

25. ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 1: B

\* ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* CUANDO EL VALOR DEL OTRO CAMPO CAMBIE, EL VALOR

\* DE ESTE CAMBIA, SE ACTUALIZA A LA DÉCIMA PARTE DEL OTRO

'DROPFALSE ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT2 ( #Decompile: 2=Actual No1=SysF ) ( ) ( "$" 'id' u )

"Escribe otro número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

17. ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( % % T // F )

;

Ejemplo 19 IfMain

**Bloquear un campo.**

Para hacer que un campo esté bloqueado en un formulario de entrada se puede usar el mensaje número 3 en el campo que se desea bloquear (este mensaje es llamado cuando el campo ha recibido el enfoque).

De acuerdo a la última tecla que haya sido presionada por el usuario, se decidirá a que otro campo se le dará el enfoque.

Para saber cual ha sido la última tecla presionada, se aprovecha el espacio del parámetro **ChooseDecompile** del campo 0. En ese espacio se coloca a una lista con el código de tecla y el plano de tecla. Esta lista es actualizada cada vez que se presiona una tecla, llamando al mensaje 0 en el message handler del formulario.



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMaBlockField ( -> ob1 ob2 ob3 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"B:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"C:" BINT0 BINT28 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT2 ( #Decompile: 2=Actual No1=SysF ) ( ) ( "$" 'id' u )

"A: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

12.16 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT2 ( #Decompile: 2=Actual No1=SysF ) ( ) ( "$" 'id' u )

"B: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

98.57 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* CON EL MESSAGE HANDLER 3, PODEMOS BLOQUEAR ESTE CAMPO.

\* Message handler del campo

' ::

\* Es llamado justo cuando un campo ha recibido el enfoque.

\* Este mensaje es llamado después del mensaje número 2

BINT3 #=casedrop ( C ) ( -> flag )

:: ( )

BlockCurrentField ( )

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT27 ( #y: 27 para fila 3/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT2 ( #Decompile: 2=Actual No1=SysF ) ( ) ( "$" 'id' u )

"C: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

47.29 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT3 ( #Netiq )

BINT3 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona alguna tecla

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

2DUP ( #ct #p #ct #p )

TWO{}N ( #ct #p {#ct #p} )

BINT5 ( #ct #p {#ct #p} #5 )

PutElemTopVStack ( #ct #p ) ( Pone en ChooseDecompile del campo 0 )

FALSE ( #ct #p F )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 T // F )

;

NULLNAME BlockCurrentField ( -> )

:: ( )

BINT5 ( #5 )

GetElemTopVStack ( {#ct #p} ) ( Retorna ChooseDecompile del campo 0 )

{ { BINT16 BINT1 } ( tecla DERECHA )

{ BINT15 BINT1 } ( tecla ABAJO )

{ BINT51 BINT1 } ( tecla ENTER )

{ BINT6 BINT1 } ( tecla OK )

} ( {#ct #p} {{# #}} )

matchob? ( T // {#ct #p} F )

case

:: ( )

LAM 'CurrentField ( #c )

#1+ ( #c+1 )

DUP ( #c+1 #c+1 )

FLASHPTR IfGetNbFields ( #c+1 #c+1 #nc )

#= ( #c+1 ¿#c+1=#nc? )

IT

DROPZERO ( #proximocampo )

( #proximocampo )

FLASHPTR IfSetField ( )

;

( {#ct #p} )

{ { BINT16 BINT3 } ( tecla ShiftDer+DERECHA )

{ BINT15 BINT3 } ( tecla ShiftDer+ABAJO )

} ( {#ct #p} {{# #}} )

matchob? ( T // {#ct #p} F )

case

:: ( )

LAM 'CurrentField ( #c )

#1- ( #c-1 ) ( #proximocampo )

FLASHPTR IfSetField ( )

;

( {#ct #p} )

{ { BINT10 BINT3 } ( tecla ShiftDer+ARRIBA )

{ BINT14 BINT3 } ( tecla ShiftDer+IZQUIERDA )

} ( {#ct #p} {{# #}} )

matchob? ( T // {#ct #p} F )

case

:: ( )

LAM 'CurrentField ( #c )

#1+ ( #c+1 ) ( #proximocampo )

FLASHPTR IfSetField ( )

;

( {#ct #p} )

{ { BINT10 BINT1 } ( tecla ARRIBA )

{ BINT14 BINT1 } ( tecla IZQUIERDA )

} ( {#ct #p} {{# #}} )

matchob? ( T // {#ct #p} F )

NOTcaseDROP

( )

LAM 'CurrentField ( #c )

#1- ( #c-1 )

DUP ( #c-1 #c-1 )

MINUSONE ( #c-1 #c-1 )

EQUAL ( #c-1 ¿#c-1=MINUSONE? )

IT

:: DROP FLASHPTR IfGetNbFields #1- ;

( #proximocampo )

FLASHPTR IfSetField ( )

;

Ejemplo 20 IfMain

**Bloquear campos con una condición.**

Actualizar el valor de un campo al cambiar el valor de otro campo.

En este ejemplo se verá como bloquear campos sólo si se cumple una condición.

Aquí hay 4 campos (0, 1, 2 y 3)

El campo 0 contiene a un campo **TEXTO** y no es bloqueado.

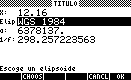
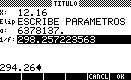
El campo 1 es un campo **CHOOSE**.

Los campos 2 y 3 son campos **TEXTO** y serán bloqueados cuando en el campo 1 la opción escogida no sea la opción “ESCRIBE PARÁMETROS”.

Para esto se usa el mensaje 3 en los campos que serán bloqueados (campos 2 y 3).

Para saber cual ha sido la última tecla presionada, se aprovecha el espacio del parámetro **ChooseDecompile** del campo 0. En ese espacio se coloca a una lista con el código de tecla y el plano de tecla. Esta lista es actualizada cada vez que se presiona una tecla, llamando al mensaje 0 en el message handler del formulario.

Además, cada vez que se escoge en el campo **CHOOSE** a una opción que no sea “ESCRIBE PARÁMETROS”, los campos 2 y 3 se actualizarán con los valores correspondientes al elipsoide escogido. Para esto, se usa el mensaje 5 en el campo CHOOSE.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMaBlockCond ( -> ob1 ob2 ob3 ob4 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"X:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Elip" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"a:" BINT0 BINT28 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"1/f:" BINT0 BINT37 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT18 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT113 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"X: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

12.16 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 1: ES UN CAMPO CHOOSE

\* Message handler del campo

' ::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia.

\* Convierte valor ext en valor int (será guardado en la pila virtual)

BINT5 #=casedrop ( C ) ( Vext -> Vint flag )

:: ( {$ % %'} )

BINT1 ( {$ % %'} #1 )

FLASHPTR IfGetFieldChooseData ( {$ % %'} {{}} )

FLASHPTR LASTCOMP ( {$ % %'} {$ % %'} )

OVER ( {$ % %'} {$ % %'} {$ % %'} )

EQUALNOT ( {$ % %'} flag )

IT

:: DUP ( {$ % %'} {$ % %'} )

TWONTHCOMPDROP\_ ( {$ % %'} % )

BINT2 ( {$ % %'} % #c )

FLASHPTR IfSetFieldValue ( {$ % %'} )

DUP ( {$ % %'} {$ % %'} )

FLASHPTR LASTCOMP ( {$ % %'} %' )

BINT3 ( {$ % %'} %' #c )

FLASHPTR IfSetFieldValue ( {$ % %'} )

;

TRUE ( {$ % %'} T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT18 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT113 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT16 ( #Decompile: 16=1ºObjC 0=NoDesc No1=SysF ) ( $ #[ME] ) ( $ )

"Escoge un elipsoide" ( Ayuda )

{ { "SouthAmerican 1969" % 6378160. % 298.25 }

{ "WGS 1960" % 6378165. % 298.3 }

{ "WGS 1966" % 6378145. % 298.25 }

{ "WGS 1972" % 6378135. % 298.26 }

{ "WGS 1984" % 6378137. % 298.257223563 }

{ "ESCRIBE PARÁMETROS" %-1 %-1 } } ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

OVER BINT5 NTHCOMPDROP ( ValorReset: 5º obj de ChooseData en CHOOSE )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* CON EL MESSAGE HANDLER 3, PODEMOS BLOQUEAR ESTE CAMPO.

\* Message handler del campo

' ::

\* Es llamado justo cuando un campo ha recibido el enfoque.

\* Este mensaje es llamado después del mensaje número 2

BINT3 #=casedrop ( C ) ( -> flag )

:: ( )

BINT1 ( #1 )

FLASHPTR IfGetFieldValue ( {$ % %'} )

BINT1 ( {$ % %'} #1 )

FLASHPTR IfGetFieldChooseData ( {$ % %'} {{}} )

FLASHPTR LASTCOMP ( {$ % %'} {$ % %'} )

EQUALNOT ( flag )

IT

BlockCurrentField ( )

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT18 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT27 ( #y: 27 para fila 3/6 )

BINT113 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"a: semieje mayor" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

98.57 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 3: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* CON EL MESSAGE HANDLER 3, PODEMOS BLOQUEAR ESTE CAMPO.

\* Message handler del campo

' ::

\* Es llamado justo cuando un campo ha recibido el enfoque.

\* Este mensaje es llamado después del mensaje número 2

BINT3 #=casedrop ( C ) ( -> flag )

:: ( )

BINT1 ( #1 )

FLASHPTR IfGetFieldValue ( {$ % %'} )

BINT1 ( {$ % %'} #1 )

FLASHPTR IfGetFieldChooseData ( {$ % %'} {{}} )

FLASHPTR LASTCOMP ( {$ % %'} {$ % %'} )

EQUALNOT ( flag )

IT

BlockCurrentField ( )

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT18 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT36 ( #y: 36 para fila 4/6 )

BINT113 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"f: achatamiento" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

47.29 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT4 ( #Netiq )

BINT4 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona alguna tecla

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

2DUP ( #ct #p #ct #p )

TWO{}N ( #ct #p {#ct #p} )

BINT5 ( #ct #p {#ct #p} #5 )

PutElemTopVStack ( #ct #p ) ( Pone en ChooseDecompile del campo 0)

FALSE ( #ct #p F )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 ob4 T // F )

;

NULLNAME BlockCurrentField ( -> )

:: ( )

BINT5 ( #5 )

GetElemTopVStack ( {#ct #p} ) ( Retorna ChooseDecompile del campo 0 )

{ { BINT16 BINT1 } ( tecla DERECHA )

{ BINT15 BINT1 } ( tecla ABAJO )

{ BINT51 BINT1 } ( tecla ENTER )

{ BINT6 BINT1 } ( tecla OK )

} ( {#ct #p} {{# #}} )

matchob? ( T // {#ct #p} F )

case

:: ( )

LAM 'CurrentField ( #c )

#1+ ( #c+1 )

DUP ( #c+1 #c+1 )

FLASHPTR IfGetNbFields ( #c+1 #c+1 #nc )

#= ( #c+1 ¿#c+1=#nc? )

IT

DROPZERO ( #proximocampo )

( #proximocampo )

FLASHPTR IfSetField ( )

;

( {#ct #p} )

{ { BINT16 BINT3 } ( tecla ShiftDer+DERECHA )

{ BINT15 BINT3 } ( tecla ShiftDer+ABAJO )

} ( {#ct #p} {{# #}} )

matchob? ( T // {#ct #p} F )

case

:: ( )

LAM 'CurrentField ( #c )

#1- ( #c-1 ) ( #proximocampo )

FLASHPTR IfSetField ( )

;

( {#ct #p} )

{ { BINT10 BINT3 } ( tecla ShiftDer+ARRIBA )

{ BINT14 BINT3 } ( tecla ShiftDer+IZQUIERDA )

} ( {#ct #p} {{# #}} )

matchob? ( T // {#ct #p} F )

case

:: ( )

LAM 'CurrentField ( #c )

#1+ ( #c+1 ) ( #proximocampo )

FLASHPTR IfSetField ( )

;

( {#ct #p} )

{ { BINT10 BINT1 } ( tecla ARRIBA )

{ BINT14 BINT1 } ( tecla IZQUIERDA )

} ( {#ct #p} {{# #}} )

matchob? ( T // {#ct #p} F )

NOTcaseDROP

( )

LAM 'CurrentField ( #c )

#1- ( #c-1 )

DUP ( #c-1 #c-1 )

MINUSONE ( #c-1 #c-1 )

EQUAL ( #c-1 ¿#c-1=MINUSONE? )

IT

:: DROP FLASHPTR IfGetNbFields #1- ;

( #proximocampo )

FLASHPTR IfSetField ( )

;

Ejemplo 21 Ifmain

**Hacer aparecer o desaparecer campos y etiquetas.**

Para hacer aparecer o desaparecer (volver visible o invisible) a un campo o a una etiqueta debes usar el mensaje número 5 en el message handler de un campo.

Este mensaje es llamado cuando el valor del campo cambia y también al inicio del formulario.

Además debes usar el comando **FLASHPTR IfSetFieldVisible** que permite volver visible o invisible a una etiqueta o a un campo.

Debes programar el message handler de la siguiente manera:

1) Poner el mensaje nº 5 en el campo cuyos cambios afectan la visibilidad de los otros campos o etiquetas.

2) En este debes hacer un test que tome como argumento al valor recien cambiado del campo.

3) Según el estado del FLAG devuelto, decidiremos si algún campo o etiqueta debe ser visible o invisible.

A continuación un ejemplo sencillo con dos campos y dos etiquetas con su respectiva explicación que puedes copiar y pegar en el editor de Debug4x.

Si en el primer campo hay un número positivo, entonces el segundo campo es invisible.

Si en el primer campo hay un número no positivo, entonces el segundo campo será visible.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMaVisibleONo ( -> ob1 ob2 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

\* ETIQUETA 0

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* ETIQUETA 1

"B:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* SI EL VALOR DE ESTE CAMPO ES MAYOR QUE CERO, DESAPARECE LA

\* ETIQUETA 1 Y EL CAMPO 1

\* DE LO CONTRARIO, LA ETIQUETA 1 Y EL CAMPO 1 SE VERÁN

\* Message handler del campo

' ::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia.

\* Convierte valor ext en valor int (será guardado en la pila virtual)

BINT5 #=casedrop ( C ) ( Vext -> Vint flag )

:: ( % )

DUP ( % % )

%0> ( % flag )

ITE

:: BINT1 FalseFalse ( % #1 F F )

FLASHPTR IfSetFieldVisible ( % ) ( Vuelve etiqueta 1 invisible )

BINT1 TrueFalse ( % #1 T F )

FLASHPTR IfSetFieldVisible ( % ) ( Vuelve campo 1 invisible )

;

:: BINT1 FalseTrue ( % #1 F T )

FLASHPTR IfSetFieldVisible ( % ) ( Vuelve etiqueta 1 invisible )

BINT1 TrueTrue ( % #1 T T )

FLASHPTR IfSetFieldVisible ( % ) ( Vuelve campo 1 invisible )

;

TRUE ( % T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT2 ( #Decompile: 2=Actual No1=SysF ) ( ) ( "$" 'id' u )

"Escribe un número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%-5 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* PARA QUE PUEDA SER VISIBLE ESTE CAMPO, EL CAMPO 0 DEBE CONTENER UN

\* NÚMERO NEGATIVO ó CERO

'DROPFALSE ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT2 ( #Decompile: 2=Actual No1=SysF ) ( ) ( "$" 'id' u )

"Escribe otro número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%17 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 T // F )

;

Ejemplo 22 Ifmain

**Desaparecer y hacer aparecer campos y etiquetas en el mismo lugar en que estaban otros campos y etiquetas.**

Problema:

Si tengo una etiqueta con su campo CHOOSE, dentro de CHOOSE tengo tres opciones

{ {"cero"} {"uno"} {"dos"} }

Cuando escoja “cero”, no aparezca ninguna ETIQUETA/CAMPO.

Cuando escoja “uno”, aparezca la etiqueta\_1 y campo\_1.

Cuando escoja “dos”, aparezca la etiqueta\_2 y campo\_2 (en el mismo lugar que etiqueta\_1/campo\_1).

Solución:

Para hacer aparecer o desaparecer (volver visible o invisible) a un campo o a una etiqueta debes usar el mensaje número 5 en el message handler de un campo.

Este mensaje es llamado cuando el valor del campo cambia y también al inicio del formulario.

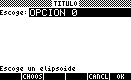
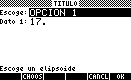
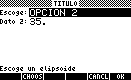
Además debes usar el comando **FLASHPTR IfSetFieldVisible** que permite volver visible o invisible a una etiqueta o a un campo.

Entonces:

1) Poner el mensaje nº 5 en el campo CHOOSE cuyo cambio afecta la visibilidad de los otros campos o etiquetas.

2) Según el contenido actual del campo CHOOSE, decidiremos si algún campo o etiqueta debe ser visible o invisible.

A continuación el código en System RPL con su respectiva explicación que puedes copiar y pegar en el editor de Debug4x

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME VisibleEncima ( -> ob1 ob2 ob3 T // F )

:: CK0 ( No se requieren argumentos )

\* Definiciones de etiquetas

\* ETIQUETA 0

"Escoge:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* ETIQUETA 1

"Dato 1:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* ETIQUETA 2

"Dato 2:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: ES UN CAMPO CHOOSE.

\* CON LA OPCION CERO, SE HACEN INVISIBLES LAS ETIQUETAS Y LOS CAMPOS 1 Y 2

\* CON LA OPCION UNO, SE HACE VISIBLES LA ETIQUETA Y EL CAMPO 1

\* CON LA OPCION DOS, SE HACE VISIBLES LA ETIQUETA Y EL CAMPO 2

\* Message handler del campo

' ::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia.

\* Convierte valor ext en valor int (será guardado en la pila virtual)

BINT5 #=casedrop ( C ) ( Vext -> Vint flag )

:: ( {} )

DUP ( {} {} )

TWONTHCOMPDROP\_ ( {} # )

:: BINT0 #=casedrop

:: BINT1 FalseFalse ( {} #1 F F )

FLASHPTR IfSetFieldVisible ( {} ) ( Vuelve etiqueta 1 invisible )

BINT1 TrueFalse ( {} #1 T F )

FLASHPTR IfSetFieldVisible ( {} ) ( Vuelve campo 1 invisible )

BINT2 FalseFalse ( {} #1 F F )

FLASHPTR IfSetFieldVisible ( {} ) ( Vuelve etiqueta 2 invisible )

BINT2 TrueFalse ( {} #1 T F )

FLASHPTR IfSetFieldVisible ( {} ) ( Vuelve campo 2 invisible )

;

BINT1 #=casedrop

:: BINT1 FalseTrue ( {} #1 F T )

FLASHPTR IfSetFieldVisible ( {} ) ( Vuelve etiqueta 1 visible )

BINT1 TrueTrue ( {} #1 T T )

FLASHPTR IfSetFieldVisible ( {} ) ( Vuelve campo 1 visible )

BINT2 FalseFalse ( {} #1 F F )

FLASHPTR IfSetFieldVisible ( {} ) ( Vuelve etiqueta 2 invisible )

BINT2 TrueFalse ( {} #1 T F )

FLASHPTR IfSetFieldVisible ( {} ) ( Vuelve campo 2 invisible )

;

BINT2 #=casedrop

:: BINT1 FalseFalse ( {} #1 F F )

FLASHPTR IfSetFieldVisible ( {} ) ( Vuelve etiqueta 1 invisible )

BINT1 TrueFalse ( {} #1 T F )

FLASHPTR IfSetFieldVisible ( {} ) ( Vuelve campo 1 invisible )

BINT2 FalseTrue ( {} #1 F F )

FLASHPTR IfSetFieldVisible ( {} ) ( Vuelve etiqueta 2 visible )

BINT2 TrueTrue ( {} #1 T F )

FLASHPTR IfSetFieldVisible ( {} ) ( Vuelve campo 2 visible )

;

;

TRUE ( {} TRUE )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT30 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 2/6 )

BINT101 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT16 ( #Decompile: 16=1ºObjC 0=NoDesc No1=SysF ) ( $ #[ME] ) ( $ )

"Escoge una opción" ( Ayuda )

{ { "OPCION 0" BINT0 }

{ "OPCION 1" BINT1 }

{ "OPCION 2" BINT2 }

} ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

OVER CARCOMP ( ValorReset: 1º obj de ChooseData en CHOOSE )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* PARA QUE PUEDA SER VISIBLE ESTE CAMPO, EL CAMPO 0 DEBE CONTENER LA OPCION 1

'DROPFALSE ( MH del campo )

BINT30 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT101 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT2 ( #Decompile: 2=Actual No1=SysF ) ( ) ( "$" 'id' u )

"Campo 1: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%17 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* PARA QUE PUEDA SER VISIBLE ESTE CAMPO, EL CAMPO 0 DEBE CONTENER LA OPCION 2

'DROPFALSE ( MH del campo )

BINT30 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT101 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT2 ( #Decompile: 2=Actual No1=SysF ) ( ) ( "$" 'id' u )

"Campo 2: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%35\_ ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT3 ( #Netiq )

BINT3 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 T // F )

;

Ejemplo 23 Ifmain

**Insertar un grob en un formulario de entrada**

Ahora veremos como insertar un grob en un formulario de entrada.

Simplemente lo insertamos como una etiqueta más.

A continuación un ejemplo sencillo con dos etiquetas y un campo que puedes copiar y pegar en el editor de Debug4x.



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMaPoneGrob ( -> ob T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

\* ETIQUETA 0: ES UNA CADENA

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* ETIQUETA 1: ES UN GROB

GROB 00036 B0000B00008F00401020209D401040154010409F40202040108F00

BINT0 BINT19 ( #xetiq ) ( #yetiq )

\* CAMPO 0: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Escribe un número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%25 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT2 ( #Netiq )

BINT1 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

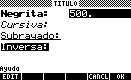
FLASHPTR IfMain ( ob T // F )

;

Ejemplo 24 Ifmain

**Dar formato a las etiquetas en IfMain**

En este ejemplo damos formatos a las etiquetas del formulario de entrada.



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME EtiquetasEstilo ( -> ob T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"Negrita:" $->NEGRITA $>GROB BINT1 BINT10

"Cursiva:" $->CURSIVA $>GROB BINT1 BINT21

"Subrayado:" $->SUBRAYADA $>GROB BINT1 BINT32

"Inversa:" $->INVERSA $>GROB BINT1 BINT43

\* CAMPO 0: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT70 ( #x >= #xetiq + 6\*#ncetiq + 2 )

BINT10 ( #y: )

BINT61 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%25 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT4 ( #Netiq )

BINT1 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob T // F )

;

NULLNAME $->NEGRITA ( $ -> $ en negrita )

:: "\13\01\13" SWAPOVER &$ &$ ;

NULLNAME $->CURSIVA ( $ -> $ en cursiva )

:: "\13\02\13" SWAPOVER &$ &$ ;

NULLNAME $->SUBRAYADA ( $ -> $ subrayada )

:: "\13\03\13" SWAPOVER &$ &$ ;

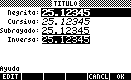
NULLNAME $->INVERSA ( $ -> $ fondo oscuro )

:: "\13\04\13" SWAPOVER &$ &$ ;

Ejemplo 25 Ifmain

**Dar formato a los campos en IfMain**

En este ejemplo damos formatos a los campo usando el mensaje número 6 en cada campo en el que se desea dar formato a su contenido antes de presentarlo en la pantalla.



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME CamposEstilo ( -> ob1 ob2 ob3 ob4 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

" Negrita:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

" Cursiva:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Subrayado:" BINT0 BINT28 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

" Inversa:" BINT0 BINT37 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: TEXTO. SE MOSTRARÁ EN NEGRITA

\* Message handler del campo

' ::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia

\* Debe mostrar el grob del campo (usando el comando ˆIfSetGrob)

BINT6 #=casedrop ( C ) ( #c Vext -> flag T // #c Vext F )

:: ( #c valor )

TRUESWAP\_ ( #c T valor )

DecompEdit ( #c T $ ) ( descompila con formato estándar )

$->NEGRITA ( #c T $' ) ( convierte a negrita )

$>GROB ( #c T grob ) ( convierte a grob )

FLASHPTR IfSetGrob ( ) ( dibuja campo en pantalla )

TrueTrue ( T T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT42 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT89 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

25.12345 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 1: TEXTO. SE MOSTRARÁ EN CURSIVA

\* Message handler del campo

' ::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia

\* Debe mostrar el grob del campo (usando el comando ˆIfSetGrob)

BINT6 #=casedrop ( C ) ( #c Vext -> flag T // #c Vext F )

:: ( #c valor )

TRUESWAP\_ ( #c T valor )

DecompEdit ( #c T $ ) ( descompila con formato estándar )

$->CURSIVA ( #c T $' ) ( convierte a cursiva )

$>GROB ( #c T grob ) ( convierte a grob )

FLASHPTR IfSetGrob ( ) ( dibuja campo en pantalla )

TrueTrue ( T T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT42 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT89 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

25.12345 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: TEXTO. SE MOSTRARÁ EN SUBRAYADA

\* Message handler del campo

' ::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia

\* Debe mostrar el grob del campo (usando el comando ˆIfSetGrob)

BINT6 #=casedrop ( C ) ( #c Vext -> flag T // #c Vext F )

:: ( #c valor )

TRUESWAP\_ ( #c T valor )

DecompEdit ( #c T $ ) ( descompila con formato estándar )

$->SUBRAYADA ( #c T $' ) ( convierte a subrayada )

$>GROB ( #c T grob ) ( convierte a grob )

FLASHPTR IfSetGrob ( ) ( dibuja campo en pantalla )

TrueTrue ( T T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT42 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT27 ( #y: 27 para fila 3/6 )

BINT89 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

25.12345 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 3: TEXTO. SE MOSTRARÁ EN INVERSA

\* Message handler del campo

' ::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia

\* Debe mostrar el grob del campo (usando el comando ˆIfSetGrob)

BINT6 #=casedrop ( C ) ( #c Vext -> flag T // #c Vext F )

:: ( #c valor )

TRUESWAP\_ ( #c T valor )

DecompEdit ( #c T $ ) ( descompila con formato estándar )

$->INVERSA ( #c T $' ) ( convierte a inversa )

$>GROB ( #c T grob ) ( convierte a grob )

FLASHPTR IfSetGrob ( ) ( dibuja campo en pantalla )

TrueTrue ( T T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT42 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT36 ( #y: 36 para fila 4/6 )

BINT89 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

25.12345 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT4 ( #Netiq )

BINT4 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 ob4 T // F )

;

NULLNAME $->NEGRITA ( $ -> $ en negrita )

:: "\13\01\13" SWAPOVER &$ &$ ;

NULLNAME $->CURSIVA ( $ -> $ en cursiva )

:: "\13\02\13" SWAPOVER &$ &$ ;

NULLNAME $->SUBRAYADA ( $ -> $ subrayada )

:: "\13\03\13" SWAPOVER &$ &$ ;

NULLNAME $->INVERSA ( $ -> $ fondo oscuro )

:: "\13\04\13" SWAPOVER &$ &$ ;

Ejemplo 26 Ifmain

**Verificar que todos los campos estén llenos al terminar el formulario con OK.**

En este ejemplo, al presionar OK o ENTER se verifica que todos los campos estén llenos.

Si hay algún campo vacío, se evita la finalización del formulario (y se muestra el mensaje “escribe todos los datos”).

El comando **FLASHPTR IfPutFieldsOnStack** ( -> ob1 ob2 ... obn )

pone todos los objetos de los campos en la pila.

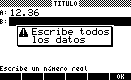
El comando **FLASHPTR IfGetNbFields** ( -> #n )

pone el número de campos en la pila como un BINT.

El comando **FLASHPTR ListPos** ( ob {} -> #i/#0 )

pone en la pila la ubicación de un objeto dentro de una lista como un BINT,

si el objeto no está presente en la lista, pone BINT0.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME CamposLlenos ( -> ob1 ob2 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"Dato 1:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"Dato 2:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT30 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT101 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Escribe un número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

MINUSONE ( ValorReset: campo TEXTO en blanco )

MINUSONE ( ValorInicial: campo TEXTO en blanco )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT30 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT101 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"Escribe un número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

MINUSONE ( ValorReset: campo TEXTO en blanco )

MINUSONE ( ValorInicial: campo TEXTO en blanco )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Si hay algún campo en blanco, evita su finalizacion con ENTER u OK

\*\*\* y muestra el mensaje "Escribe todos los datos"

BINT16 #=casedrop ( F ) ( -> flag T // F )

:: ( )

FLASHPTR IfPutFieldsOnStack ( ob1 ob2...obn )

FLASHPTR IfGetNbFields ( ob1 ob2...obn #n )

{}N ( { ob1 ob2...obn } )

' xNOVAL ( { ob1 ob2...obn } xNOVAL )

SWAP ( xNOVAL { ob1 ob2...obn } )

FLASHPTR ListPos ( #pos/#0 )

#0=ITE

TrueTrue

:: "Escribe todos los datos" FlashWarning FalseTrue ;

( flag T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 T // F )

;

Ejemplo 27 Ifmain

**Verificar que los valores de los campos cumplan alguna condición antes de confirmar la salida con OK o ENTER.**

En este ejemplo, al presionar OK o ENTER se verifica que los valores de los campos cumplan alguna condición.

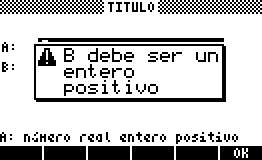
Si algún campo no cumple una condición, entonces se muestra un mensaje de advertencia y no se termina el formulario. Además, el enfoque va hacia ese campo.

El comando **FLASHPTR IfGetFieldValue** ( #campo -> ob )

retorna el valor externo de un campo.

El comando **FLASHPTR IfSetField** ( #campo -> )

fija al campo especificado como el nuevo campo actual (el que tendrá el enfoque).

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME CamposValidosOK ( -> ob1 ob2 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"B:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"A: número real entero positivo" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

MINUSONE ( ValorReset: campo TEXTO en blanco )

MINUSONE ( ValorInicial: campo TEXTO en blanco )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"B: número real entero positivo" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

MINUSONE ( ValorReset: campo TEXTO en blanco )

MINUSONE ( ValorInicial: campo TEXTO en blanco )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

BINT16 #=casedrop ( F ) ( -> flag T // F )

:: ( )

BINT0 ( #0 )

FLASHPTR IfGetFieldValue ( ob )

TestRealEnteroPositivo ( flag )

NOTcase

:: ( )

"A debe ser un entero positivo" ( $ )

FlashWarning ( )

BINT0 ( #0 )

FLASHPTR IfSetField ( )

FalseTrue ( F T )

;

BINT1 ( #1 )

FLASHPTR IfGetFieldValue ( ob )

TestRealEnteroPositivo ( flag )

NOTcase

:: ( )

"B debe ser un entero positivo" ( $ )

FlashWarning ( )

BINT1 ( #1 )

FLASHPTR IfSetField ( )

FalseTrue ( F T )

;

( )

TrueTrue ( T T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 T // F )

;

\* Test realizado a un objeto.

\* Retorna TRUE si en la pila hay un número real que sea entero y positivo.

NULLNAME TestRealEnteroPositivo ( ob -> flag )

:: ( ob )

DUPTYPEREAL? ( ob flag )

NOTcase DROPFALSE ( sale con FALSE )

( % )

DUP ( % % )

%0> ( % flag )

NOTcase DROPFALSE ( sale con FALSE )

( %positivo )

%FP ( %ParteDecimal )

%0= ( flag )

;

Ejemplo 28 Ifmain

**Mostrar el valor de un campo como grob.**

Este es un message handler para un campo (**TEXTO**, **CHOOSE** o **COMBOCHOOSE**). Este tiene el mismo efecto que el código por defecto para mostrar el valor del campo en la pantalla.

Lo presentamos para que hagas los cambios que creas conveniente para tu programa.

NULLNAME MH6\_AccionesPorDefecto

:: BINT6 #=casedrop ( IfMsgGetFieldGrob ) ( #c val -> flag T // #c val F )

:: ( #campo valor )

OVER ( #campo valor #campo )

FLASHPTR IfGetFieldDecompObject ( #campo valor #Decompile )

OVER ( #campo valor #Decompile valor )

' xNOVAL

EQUAL ( #campo valor #Decompile flag )

IT

:: ( #campo valor #Decompile )

SWAPDROP ( #campo #Decompile )

MINUSONE ( #campo #Decompile MINUSONE )

SWAP ( #campo MINUSONE #Decompile )

;

OVER ( #campo valor #Decompile valor )

MINUSONE

#=case

:: ( #campo valor #Decompile )

FLASHPTR 2DROPTRUE ( #campo T )

GROB 0000A 0000000000 ( #campo T grob )

FLASHPTR IfSetGrob ( )

TrueTrue ( T T )

;

( #campo valor #Decompile )

SWAP ( #campo #Decompile valor )

OVER BINT32 #AND #0<> ( #campo #Decompile valor flag )

IT

:: BINT2

NTHELCOMP

?SEMI

NULL$

;

( #campo #Decompile valor )

OVER BINT16 #AND #0<> ( #campo #Decompile valor flag )

IT

:: BINT1 NTHELCOMP

?SEMI

NULL$

;

( #campo #Decompile valor )

?GetMsg\_ ( #campo #Decompile valor )

OVER BINT4 #AND #0<> ( #campo #Decompile valor flag )

IT

DecompEdit

( #campo #Decompile valor )

OVER BINT2 #AND #0<> ( #campo #Decompile valor flag )

IT

DecompStd1Line

( #campo #Decompile $ )

OVER BINT8 #AND #0<> ( #campo #Decompile $ flag )

IT

:: ONEONE

SUB$

;

( #campo #Decompile $ )

SWAP BINT1 #AND #0<> ( #campo $ flag )

ITE

$>grob

$>GROB

( #campo grob )

TRUESWAP\_ ( #campo T grob )

FLASHPTR IfSetGrob ( )

TrueTrue ( T T )

;

DROPFALSE

;

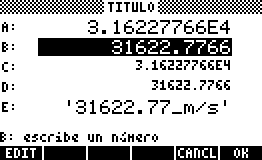
Ejemplo 29 Ifmain

**Alinear campos a la derecha. Caso General.**

Este es un message handler para un campo (texto, choose o combochoose).

Se ha modificando el mensaje número seis del ejemplo anterior. Ahora se podrán alinear los campos a la derecha, respetando el parámetro **#Decompile** del campo, cualquiera que sea este parámetro.

Puedes poner este NULLNAME como el message handler en cada campo donde desees que su contenido se muestre pegado a la derecha.



NULLNAME MH6\_AlinearCamposDerecha

:: BINT6 #=casedrop ( IfMsgGetFieldGrob ) ( #c val -> flag T // #c val F )

:: ( #campo valor )

OVER ( #campo valor #campo )

FLASHPTR IfGetFieldDecompObject ( #campo valor #Decompile )

OVER ( #campo valor #Decompile valor )

' xNOVAL

EQUAL ( #campo valor #Decompile flag )

IT

:: ( #campo valor #Decompile )

SWAPDROP ( #campo #Decompile )

MINUSONE ( #campo #Decompile MINUSONE )

SWAP ( #campo MINUSONE #Decompile )

;

OVER ( #campo valor #Decompile valor )

MINUSONE

#=case

:: ( #campo valor #Decompile )

FLASHPTR 2DROPTRUE ( #campo T )

GROB 0000A 0000000000 ( #campo T grob )

FLASHPTR IfSetGrob ( )

TrueTrue ( T T )

;

( #campo valor #Decompile )

SWAP ( #campo #Decompile valor )

OVER BINT32 #AND #0<> ( #campo #Decompile valor flag )

IT

:: BINT2

NTHELCOMP

?SEMI

NULL$

;

( #campo #Decompile valor )

OVER BINT16 #AND #0<> ( #campo #Decompile valor flag )

IT

:: BINT1 NTHELCOMP

?SEMI

NULL$

;

( #campo #Decompile valor )

?GetMsg\_ ( #campo #Decompile valor ) ( # -> $ )

OVER BINT4 #AND #0<> ( #campo #Decompile valor flag )

IT

DecompEdit

( #campo #Decompile valor )

OVER BINT2 #AND #0<> ( #campo #Decompile valor flag )

IT

DecompStd1Line

( #campo #Decompile $ )

OVER BINT8 #AND #0<> ( #campo #Decompile $ flag )

IT

:: ONEONE

SUB$

;

( #campo #Decompile $ )

SWAP BINT1 #AND #0<> ( #campo $ flag )

ITE

:: ( #campo $ )

OVER

TRUE ( #campo $ #campo T )

FLASHPTR IfGetFieldPos ( #campo $ #x #y #w #h )

DROP ( #campo $ #x #y #w )

UNROT2DROP ( #campo $ #w )

BINT4 ( #campo $ #w #4 )

#/ ( #campo $ #r #q )

SWAPDROP ( #campo $ #q )

FUERZA$ ( #campo $' )

$>grob ( #campo $' grob )

;

:: ( #campo $ )

OVER

TRUE ( #campo $ #campo T )

FLASHPTR IfGetFieldPos ( #campo $ #x #y #w #h )

DROP ( #campo $ #x #y #w )

UNROT2DROP ( #campo $ #w )

BINT6 ( #campo $ #w #4 )

#/ ( #campo $ #r #q )

SWAPDROP ( #campo $ #q )

FUERZA$ ( #campo $' )

$>GROB ( #campo $' grob )

;

( #campo grob )

TRUESWAP\_ ( #campo T grob )

FLASHPTR IfSetGrob ( )

TrueTrue ( T T )

;

DROPFALSE

;

\* Este subprograma retorna siempre una cadena que tenga #max caracteres

\* Si $ tiene pocos caracteres, le agrega caract en blanco a la izquierda

\* Si $ tiene muchos caracteres, lo corta y agrega "..." al final

NULLNAME FUERZA$ ( $ #max -> $ )

:: ( $ #max )

OVERLEN$ ( $ #max #len )

2DUP#= ( $ #max #len flag )

case2DROP

( $ #max #len )

2DUP#< ( $ #max #len flag )

casedrop

:: 1\_#1-SUB$ "\1E" &$ ;

( $ #max #len )

#- ( $ #adicionales )

Blank$ ( $ $' )

SWAP&$ ( $'' )

;

Ejemplo 30 Ifmain

**Alinear campos a la derecha.**

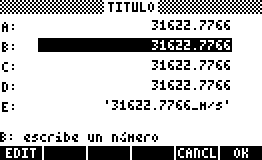
**Caso especial. Formato numérico estándar y minifuente.**

Usando este message handler, el valor del campo (texto, choose o combochoose) será mostrado a la derecha, siempre en formato numérico estándar y con minifuente.

Para verlo en formato númerico actual cambia **DecompEdit** por **DecompStd1Line**.

Para verlo en fuente normal cambia BINT4 por BINT6 y **$>grob** por **$>GROB**.

Puedes poner este NULLNAME como el message handler en cada campo donde desees que su contenido se muestre pegado a la derecha.



\* Este es el message handler que puedes usar para alinear el

\* contenido del campo a la derecha.

\* Pone los números en minifuente con el formato actual.

\* Ponlo como el message handler de cada campo que desees.

NULLNAME MH6\_AlinCampDerMiniSTD

:: BINT6 #=casedrop

:: ( #campo valor )

DecompEdit ( #campo $ ) ( formato numérico estándar )

TRUESWAP\_ ( #campo T $ )

3PICK ( #campo T $ #campo )

TRUE ( #campo T $ #campo T )

FLASHPTR IfGetFieldPos ( #campo T $ #x #y #w #h )

DROP ( #campo T $ #x #y #w )

UNROT2DROP ( #campo T $ #w )

BINT4 ( #campo T $ #w #4 )

#/ ( #campo T $ #r #q )

SWAPDROP ( #campo T $ #q )

FUERZA$ ( #campo T $' )

$>grob ( #campo T grob )

FLASHPTR IfSetGrob ( )

TrueTrue ( T T )

;

DROPFALSE

;

\* Este subprograma retorna siempre una cadena que tenga #max caracteres

\* Si $ tiene pocos caracteres, le agrega caract en blanco a la izquierda

\* Si $ tiene muchos caracteres, lo corta y agrega "..." al final

NULLNAME FUERZA$ ( $ #max -> $ )

:: ( $ #max )

OVERLEN$ ( $ #max #len )

2DUP#= ( $ #max #len flag )

case2DROP

( $ #max #len )

2DUP#< ( $ #max #len flag )

casedrop

:: 1\_#1-SUB$ "\1E" &$ ;

( $ #max #len )

#- ( $ #adicionales )

Blank$ ( $ $' )

SWAP&$ ( $'' )

;

Ejemplo 31 Ifmain

**Validación de datos con el mensaje número 22.**

**Número real y positivo.**

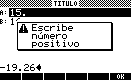
En este ejemplo veremos como usar el mensaje 22 para validar un campo que acepte a números reales.

Si el valor de la línea de edición es del tipo correcto (número real), decidiremos si su valor es válido o no para el campo.

En este caso hay dos campos que contienen a números reales. La validación se hará solamente en el primer campo. Por ejemplo, para este campo el número real será válido si es positivo. De esta manera, si el objeto es de un tipo correcto, pero de un valor inválido, deberemos de retornar TRUE. En otro caso, se deberá retornar FALSE, para que se realice la acción por defecto.

Por ejemplo, si el usuario ingresa -19.26 (como se muestra en la figura), en el campo no se guardará el número y se mostrará la alerta “Escribe número positivo”.

Sin embargo la validación con el mensaje 22 tiene muchas desventajas respecto a la validación con el mensaje 5 (ejemplo siguiente). Por ejemplo, al ingresar objetos para varios campos en una sóla línea de comandos, la validación sólo es hecha en el campo actual. También la validación no es hecha al calcular un valor en la pila con la tecla de menú CALC Por esto, para asegurar que al salir del formulario con ENTER u OK, siempre sea retornado un valor válido, deberás usar el mensaje 16 en el message handler del formulario como en el ejemplo 21.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME ValidPosit22 ( -> ob1 ob2 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"B:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* Message handler del campo

' ::

\* Es llamado cuando hay línea de edición activa y es presionada ENTER u OK

\* Efecto: cuando hay objeto de tipo válido pero con valor incorrecto

BINT22 #=casedrop ( C ) ( Vext -> T // F )

:: ( )

RCL\_CMD ( $ )

palparse ( ob T // $ #pos $' F )

NOTcase2drop DROPFALSE ( NOTA: SALE CON FALSE )

( ob )

EVAL ( ??? )

DEPTH ( ob1 ... obn #npila // #0 )

DUP#0=case

DROPFALSE

( ob1 ... obn #npila )

#1- NDROP ( ob )

DUPTYPEZINT? ( Z T // ob F )

IT

FLASHPTR Z>R

( % // ob )

DUPTYPEREAL? ( % T // ob F )

NOTcase

DROPFALSE

( % )

%0 %<= ( flag )

DUP ( flag flag )

IT :: "Escribe número positivo" FlashWarning ;

( flag )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"A: número positivo" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"B: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%16 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 ob4 T // F )

;

Ejemplo 32 Ifmain

**Validación de datos con el mensaje número 5.**

**Número real y positivo.**

En este ejemplo veremos como usar el mensaje 5 para validar un campo que acepte a números reales.

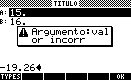
Decidiremos si su valor es válido o no para el campo.

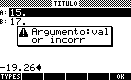
En este caso hay dos campos que contienen a números reales. La validación se hará solamente en el primer campo. Por ejemplo, para este campo el número real será válido si es positivo. De esta manera, si el objeto es de un valor inválido, deberemos de generar un error (por ejemplo, usando los comandos **ERRJMP** o **ERROROUT**).

Por ejemplo, si el usuario ingresa -19.26 (como se muestra en la figura), en el campo no se guardará el número y se mostrará una alerta con el texto correspondiente al mensaje de error generado.

Esta validación con el mensaje 5 tiene muchas ventajas respecto a la validación con el mensaje 22 (ejemplo anterior). Por ejemplo, al ingresar objetos para varios campos en una sóla línea de comandos, la validación es hecha en todos los campos. También la validación es hecha al calcular un valor en la pila con la tecla de menú CALC

Por esto, cuando usamos validación con el mensaje 5, está asegurado que al salir del formulario con ENTER u OK, siempre será retornado un valor válido; y ya no será necesario usar el mensaje 16 en el message handler del formulario.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME ValidPosit5 ( -> ob1 ob2 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"B:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* Message handler del campo

' ::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia.

\* Convierte valor ext en valor int (será guardado en la pila virtual)

BINT5 #=casedrop ( C ) ( Vext -> Vint flag )

:: ( valor )

DUP ( valor valor )

' xNOVAL ( valor valor NOVAL )

EQUAL ( valor flag )

caseTRUE ( sale con TRUE )

( % )

DUP ( % % )

%0 %<= ( % flag )

IT

:: # 203 ERROROUT ; ( genera error: argumento valor incorr )

( % )

TRUE ( % T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"A: número positivo" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT18 ( #y: 18 para fila 2/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"B: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%16 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 ob3 ob4 T // F )

;

Ejemplo 33 Ifmain

**Cambiando automáticamente una entrada inválida.**

**Número real entero y positivo.**

En este ejemplo hay un campo texto que acepta sólo números reales.

Sólo se aceptan números reales que sean enteros y positivos.

Si el usuario escribe un número negativo, la calculadora automáticamente le cambia de signo. También, si el número tiene parte decimal, automáticamente el número es redondeado de manera que el valor guardado en el campo (valor interno) sea siempre un número real entero y positivo.

Por ejemplo, si el usuario ingresa -3.74 (como se muestra en la figura), en el campo se guardará el número 4 en lugar del número ingresado.

Por lo tanto, el valor del campo devuelto al finalizar con OK o ENTER será siempre un número real que sea entero y positivo.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME CampoEntPosit ( -> ob T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* Message handler del campo

' ::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia.

\* Convierte valor ext en valor int (será guardado en la pila virtual)

BINT5 #=casedrop ( C ) ( Vext -> Vint flag )

:: ( valor )

DUP ( valor valor )

' xNOVAL ( valor valor NOVAL )

EQUAL ( valor flag )

caseTRUE ( sale con TRUE )

( % )

%ABS %0 RNDXY ( %' )

DUP%0= ( %' flag )

IT %1+

TRUE ( %'' T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"A: número real entero positivo" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

'DROPFALSE ( MH del IfMain )

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob T // F )

;

Ejemplo 34 Ifmain

**Campo COMBOCHOOSE transformado en campo COMBOFILER que contiene a una FORMACIÓN.**

**Manejando los mensajes 6, 17 y 25 en el campo.**

**Manejando los mensajes 0 y 16 en el formulario.**

En este ejemplo hay un campo COMBOCHOOSE🡪COMBOFILER que contiene a una formación (matriz o arreglo).

En el campo COMBOCHOOSE🡪COMBOFILER se llaman a los mensajes 6, 17 y 25.

Con el mensaje 6, podemos ver la formación como un grob. Ya no será necesario el uso del parámetro **#Decompile** en ese campo.

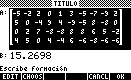
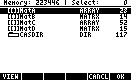
Con el mensaje 17, se podrá llamar al FILER cada vez que se presione la tecla CHOOS (F2).

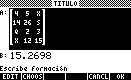
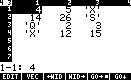
Con el mensaje 25, podemos editar la formación en el MTRW al presionar EDIT.

En el Message Handler del formulario se llaman a los mensajes 0 y 16.

Con el mensaje 0, podremos hacer que los campos COMBOCHOOSE funcionen correctamente.

Con el mensaje 16, podemos evitar la finalización del formulario (al presionar ENTER u OK), si hay un campo que esté vacío.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMaFormacion ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"B:" BINT0 BINT53 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: CAMPO COMBOCHOOSE->COMBOFILER. CONTIENE A UNA FORMACIÓN

' MH6\_17\_25\_IfMaFormacion ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT41 ( #h: )

BINT2 ( #TipoDeCampo: COMBOCHOOSE )

{ BINT3 BINT4 } ( TiposPermitidos: ArregReales, Matr; ArregNoR )

MINUSONE ( #Decompile: No es necesario, pues mensaje 6 es manejado )

"Escribe formación" ( Ayuda )

MINUSONE ( ChooseData: No se necesita en campo COMCHOOSE->COMFILER )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

MINUSONE ( ValorReset: campo COMCHOOSE->COMFILER en blanco )

MINUSONE ( ValorInicial: campo COMCHOOSE->COMFILER en blanco )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT52 ( #y: )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"B: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona alguna tecla

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Permite que se puedan usar campos COMBOCHOOSE

BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

FALSE ( #ct #p F )

LAM 'CurrentField ( #ct #p F #c )

FLASHPTR IfGetFieldType ( #ct #p F #TipoCampo )

#2= ( #ct #p F flag ) ( CampoActual=COMBOCHOOSE? )

NOT?SEMI

( #ct #p F )

R> ( #ct #p F prog )

FLASHPTR 002 0A5 ( #ct #p F $ ) ( Equivale al comando x->H )

"52133" ( #ct #p F $ "52133" ) ( dirección del BINT3 )

"B1133" ( #ct #p F $ "52133" "B1133" ) ( dirección de BINT2 )

FLASHPTR 00F 01A ( #ct #p F $' %3 ) ( Equivale al comando xSREPL )

DROP ( #ct #p F $' )

FLASHPTR 002 0A4 ( #ct #p F prog' ) ( Equivale al comando xH-> )

>R ( #ct #p F )

;

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Si hay algún campo en blanco, evita su finalizacion con ENTER u OK

\*\*\* y muestra el mensaje "Escribe todos los datos"

BINT16 #=casedrop ( F ) ( -> flag T // F )

:: ( )

FLASHPTR IfPutFieldsOnStack ( ob1 ob2...obn )

FLASHPTR IfGetNbFields ( ob1 ob2...obn #n )

{}N ( { ob1 ob2...obn } )

' xNOVAL ( { ob1 ob2...obn } xNOVAL )

SWAP ( xNOVAL { ob1 ob2...obn } )

FLASHPTR ListPos ( #pos/#0 )

#0=ITE

TrueTrue

:: "Escribe todos los datos" FlashWarning FalseTrue ;

( flag T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 T // F )

;

NULLNAME MH6\_17\_25\_IfMaFormacion

::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia

\* Debe mostrar el grob del campo (usando el comando ˆIfSetGrob)

BINT6 #=casedrop ( C ) ( #c Vext -> flag T // #c Vext F )

:: ( #c valor )

DUP ( #c valor valor )

' xNOVAL ( #c valor valor xNOVAL )

EQUAL ( #c valor flag )

caseFALSE

( #c valor )

TRUESWAP\_ ( #c T valor )

ob>grobmini ( #c T grob )

3PICK3PICK ( #c T grob #c T )

NULLPAINT\_ ( #c T grob #c T grob )

FLASHPTR IfSetGrob ( #c T grob )

FLASHPTR IfSetGrob ( )

TrueTrue ( T T )

;

\* Cuando se presiona la tecla CHOOS en un campo CHOOSE o COMBOCHOOSE.

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

BINT17 #=casedrop ( C/F ) ( -> T // F )

:: ( )

Filer\_Home\_S1\_NoDir\_FORMACION ( ob T // F )

IT

FLASHPTR IfSetCurrentFieldValue

TRUE ( T )

;

\* Cuando se presiona EDIT en un campo TEXTO, COMBOCHOOSE o CHECK.

\* Primero es llamado el mensaje del campo, luego el del formulario.

\* Si se maneja el mensaje del campo, ya no es llamado el mensaje del

\* formulario ni tampoco el código estándar.

\* Si se maneja el mensaje del formulario, ya no se llama al códig est

\* Para manejar a este mensaje debes dejar TRUE.

\* La entrada es el valor externo del campo actual.

\* Puedes iniciar la línea de edición con ese valor.

\* O también puedes editar el valor del campo actual con otro editor

\* y luego modificar el campo actual.

\* O también puedes editar el valor del campo actual con otro editor

\* y dejar el valor modificado en la pila seguido por TRUE

\* (ese valor será puesto en el campo actual de forma automática).

\* Si este mensaje es manejado, ya no se llamará al mensaje número 23.

BINT25 #=casedrop ( C/F ) ( Vext -> T // Vext T // Vext F )

:: ( ob )

EditaFormacionEnMTRW ( RealArry/CArry/MATRIX T // F )

DROPTRUE ( RealArry/CArry/MATRIX T // T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

\* CONVIERTE ob A GROB PEQUEÑO USANDO MINIFUENTE

\* Intenta convertir a grob con FLASHPTR EQW3GROBmini

\* Si no es posible hace lo siguiente:

\* Si ob es un grob, no hace nada.

\* Si ob es una cadena, hace $>grobCR

\* Si ob es de otro tipo, hace FLASHPTR FSTR11 $>grobCR

NULLNAME ob>grobmini ( ob -> grob )

:: ( ob )

BEGIN

:: DUP ( ob ob )

FLASHPTR EQW3GROBmini ( ob ext grob #0 // ob ob #1/2 )

BINT0 #=casedrop

:: UNROT2DROP TRUE ; ( OBS: SALE CON grob T ) ( SALE DE BUCLE )

( ob ob #1/2 )

BINT1 #=case

:: GARBAGE DROPFALSE ; ( OBS: SALE CON ob F ) ( REPITE BUCLE )

( ob ob )

TYPEGROB? ( ob flag )

caseTRUE

( ob )

DUPTYPECSTR? ( ob flag )

?SKIP

FLASHPTR FSTR11

( $ )

$>grobCR ( grob )

TRUE ( grob T ) ( SALE DE BUCLE )

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( grob )

;

\* Si en la pila se encuentra xNOVAL o MINUSONE, abre el MTRW.

\* Si hay otro objeto, lo edita en MTRW (de ser posible)

NULLNAME EditaFormacionEnMTRW ( ob -> RealArry/CArry/MATRIX T // F )

::

RunSafeFlags

:: BEGIN

BINT91 ClrSysFlag ( ob ) ( 91 MTRW: no list of list )

DUPTYPEBINT? ( ob flag )

OVER ( ob flag ob )

TYPECOL? ( ob flag flag' )

OR ( ob flag'' )

ITE

:: DROP ( )

FLASHPTR RunDoNewMatrix ( ob' T // F ) ( abre MTRW )

;

FLASHPTR RunDoOldMatrix ( ob' T // F ) ( edita ob en MTRW )

ITE

:: DUPTYPELIST? ( ob' flag )

ITE

FALSE ( ob' F )

TrueTrue ( ob' T T )

;

FalseTrue

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( RealArry/CArry/MATRIX T // F )

;

;

\* Busca una FORMACIÓN en el FILER

\* Inicia en el directorio HOME

\* Se puede desplazar por todos lados

\* No permite selección múltiple

\* Solo retorna una formación seguida de TRUE.

\* Si se cancela, sólo retorna FALSE.

NULLNAME Filer\_Home\_S1\_NoDir\_FORMACION ( -> ob T // F )

::

{ # 29E8 # 2686 # 2A96 } ( Filer\_Tipos: ARRAY, MATRIX y Directorios )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* HAY 6 TECLAS DEL MENU

\* La 1º es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* Esta lista es para quitar la tecla +/- "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT28 ( Tecla\_Atajo: BINT28 = #1C = Tecla +/- "CHECK" )

}

\* Esta lista es para quitar la tecla ALPHA + ENTER "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 133 ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista no hace nada

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

}

\* Esta lista es para la tecla de menú F5 "CANCL"

{ # DF2C ( Nombre: # DF2C "CANCL" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT37 ( Acción: BINT37= Sale del filer dejando FALSE en la pila )

}

\* Esta lista es para la tecla de menú F6 "OK"

\* También es para asignarla a la tecla ENTER

{ # DF2D ( Nombre: # DF2D "OK" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT17 ( Acción: BINT17= Prog Pers que pone Ruta, Objeto y Nombre )

:: ( Ruta Objeto Nombre )

DROPSWAPDROP ( Objeto )

DUPTYPERRP? ( Objeto flag ) ( Evita finalizar con Directorio )

caseDrpBadKy ( )

' TakeOver ( Objeto TakeOver )

; ( Programa\_Extra: Sale del filer, deja en la pila el Objeto y TRUE )

# 33 ( Tecla\_Atajo: BINT51 = #33 = Tecla ENTER )

}

}

FLASHPTR FILER\_MANAGERTYPE ( ob T // F // :0:{} )

TRUE

EQUAL ( ob T // F )

;

Ejemplo 35 Ifmain

**Campo COMBOCHOOSE transformado en campo COMBOFILER que contiene a un arreglo real.**

**Manejando los mensajes 5, 6, 17 y 25 en el campo.**

**Manejando los mensajes 0 y 16 en el formulario.**

En este ejemplo hay un campo COMBOCHOOSE🡪COMBOFILER que contiene a un arreglo real.

En el campo COMBOCHOOSE🡪COMBOFILER se llaman a los mensajes 5, 6, 17 y 25.

Con el mensaje 5, podemos validar la línea de comandos al escribir un argumento desde el editor de texto.

Con el mensaje 6, podemos ver la formación como un grob. Ya no será necesario el uso del parámetro **#Decompile** en ese campo.

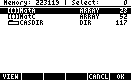
Con el mensaje 17, se podrá llamar al FILER cada vez que se presione la tecla CHOOS (F2).

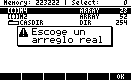
Con el mensaje 25, podemos editar la formación en el MTRW al presionar EDIT.

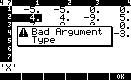
En el Message Handler del formulario se llaman a los mensajes 0 y 16.

Con el mensaje 0, podremos hacer que los campos COMBOCHOOSE funcionen correctamente.

Con el mensaje 16, podemos evitar la finalización del formulario (al presionar ENTER u OK), si hay un campo que esté vacío.



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMaArregloReal ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"B:" BINT0 BINT53 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: CAMPO COMBOCHOOSE->COMBOFILER. CONTIENE A UN ARREGLO REAL

' MH5\_6\_17\_25\_ArregloReal ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT41 ( #h: )

BINT2 ( #TipoDeCampo: COMBOCHOOSE )

{ BINT3 } ( TiposPermitidos: Arreglos reales y matrices simbólicas )

MINUSONE ( #Decompile: No es necesario, pues mensaje 6 es manejado )

"Escribe arreglo real" ( Ayuda )

MINUSONE ( ChooseData: No se necesita en campo COMCHOOSE->COMFILER )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

MINUSONE ( ValorReset: campo COMCHOOSE->COMFILER en blanco )

MINUSONE ( ValorInicial: campo COMCHOOSE->COMFILER en blanco )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT52 ( #y: )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"B: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona alguna tecla

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Permite que se puedan usar campos COMBOCHOOSE

BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

FALSE ( #ct #p F )

LAM 'CurrentField ( #ct #p F #c )

FLASHPTR IfGetFieldType ( #ct #p F #TipoCampo )

#2= ( #ct #p F flag ) ( CampoActual=COMBOCHOOSE? )

NOT?SEMI

( #ct #p F )

R> ( #ct #p F prog )

FLASHPTR 002 0A5 ( #ct #p F $ ) ( Equivale al comando x->H )

"52133" ( #ct #p F $ "52133" ) ( dirección del BINT3 )

"B1133" ( #ct #p F $ "52133" "B1133" ) ( dirección de BINT2 )

FLASHPTR 00F 01A ( #ct #p F $' %3 ) ( Equivale al comando xSREPL )

DROP ( #ct #p F $' )

FLASHPTR 002 0A4 ( #ct #p F prog' ) ( Equivale al comando xH-> )

>R ( #ct #p F )

;

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Si hay algún campo en blanco, evita su finalizacion con ENTER u OK

\*\*\* y muestra el mensaje "Escribe todos los datos"

BINT16 #=casedrop ( F ) ( -> flag T // F )

:: ( )

FLASHPTR IfPutFieldsOnStack ( ob1 ob2...obn )

FLASHPTR IfGetNbFields ( ob1 ob2...obn #n )

{}N ( { ob1 ob2...obn } )

' xNOVAL ( { ob1 ob2...obn } xNOVAL )

SWAP ( xNOVAL { ob1 ob2...obn } )

FLASHPTR ListPos ( #pos/#0 )

#0=ITE

TrueTrue

:: "Escribe todos los datos" FlashWarning FalseTrue ;

( flag T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 T // F )

;

NULLNAME MH5\_6\_17\_25\_ArregloReal

::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia.

\* Convierte valor ext en valor int (será guardado en la pila virtual)

BINT5 #=casedrop ( C ) ( Vext -> Vint flag )

:: ( valor )

DUP ( valor valor )

' xNOVAL ( valor valor NOVAL )

EQUAL ( valor flag )

caseTRUE ( sale con TRUE )

( valor )

DUP ( valor valor )

TYPERARRY? ( valor flag )

NOT\_IT

:: # 203 ERROROUT ;

( valor )

TRUE ( valor T )

;

\* Cuando se inicia el IfMain y cuando el valor del campo cambia

\* Debe mostrar el grob del campo (usando el comando ˆIfSetGrob)

BINT6 #=casedrop ( C ) ( #c Vext -> flag T // #c Vext F )

:: ( #c valor )

DUP ( #c valor valor )

' xNOVAL ( #c valor valor xNOVAL )

EQUAL ( #c valor flag )

caseFALSE

( #c valor )

TRUESWAP\_ ( #c T valor )

ob>grobmini ( #c T grob )

3PICK3PICK ( #c T grob #c T )

NULLPAINT\_ ( #c T grob #c T grob )

FLASHPTR IfSetGrob ( #c T grob )

FLASHPTR IfSetGrob ( )

TrueTrue ( T T )

;

\* Cuando se presiona la tecla CHOOS en un campo CHOOSE o COMBOCHOOSE.

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

BINT17 #=casedrop ( C/F ) ( -> T // F )

:: ( )

Filer\_Home\_S1\_NoDir\_Arry ( RealArry T // F )

IT

FLASHPTR IfSetCurrentFieldValue

( )

TRUE ( T )

;

\* Cuando se presiona EDIT en un campo TEXTO, COMBOCHOOSE o CHECK.

\* Primero es llamado el mensaje del campo, luego el del formulario.

\* Si se maneja el mensaje del campo, ya no es llamado el mensaje del

\* formulario ni tampoco el código estándar.

\* Si se maneja el mensaje del formulario, ya no se llama al códig est

\* Para manejar a este mensaje debes dejar TRUE.

\* La entrada es el valor externo del campo actual.

\* Puedes iniciar la línea de edición con ese valor.

\* O también puedes editar el valor del campo actual con otro editor

\* y luego modificar el campo actual.

\* O también puedes editar el valor del campo actual con otro editor

\* y dejar el valor modificado en la pila seguido por TRUE

\* (ese valor será puesto en el campo actual de forma automática).

\* Si este mensaje es manejado, ya no se llamará al mensaje número 23.

BINT25 #=casedrop ( C/F ) ( Vext -> T // Vext T // Vext F )

:: ( ob )

EditaArregloRealEnMTRW ( RealArry T // F )

DROPTRUE ( RealArry T // T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

\* CONVIERTE ob A GROB PEQUEÑO USANDO MINIFUENTE

\* Intenta convertir a grob con FLASHPTR EQW3GROBmini

\* Si no es posible hace lo siguiente:

\* Si ob es un grob, no hace nada.

\* Si ob es una cadena, hace $>grobCR

\* Si ob es de otro tipo, hace FLASHPTR FSTR11 $>grobCR

NULLNAME ob>grobmini ( ob -> grob )

:: ( ob )

BEGIN

:: DUP ( ob ob )

FLASHPTR EQW3GROBmini ( ob ext grob #0 // ob ob #1/2 )

BINT0 #=casedrop

:: UNROT2DROP TRUE ; ( OBS: SALE CON grob T ) ( SALE DE BUCLE )

( ob ob #1/2 )

BINT1 #=case

:: GARBAGE DROPFALSE ; ( OBS: SALE CON ob F ) ( REPITE BUCLE )

( ob ob )

TYPEGROB? ( ob flag )

caseTRUE

( ob )

DUPTYPECSTR? ( ob flag )

?SKIP

FLASHPTR FSTR11

( $ )

$>grobCR ( grob )

TRUE ( grob T ) ( SALE DE BUCLE )

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( grob )

;

\* Si en la pila se encuentra xNOVAL, abre el MTRW.

\* Si hay otro objeto, lo edita en MTRW (de ser posible)

\* Retorna un arreglo real y TRUE o sólo FALSE

NULLNAME EditaArregloRealEnMTRW ( ob -> RealArry T // F )

::

RunSafeFlags

:: BEGIN

BINT91 ClrSysFlag ( ob ) ( 91 MTRW: no list of list )

DUP ( ob ob )

' xNOVAL ( ob ob xNOVAL )

EQUAL ( ob flag )

ITE

:: DROP ( )

FLASHPTR DoNewMatrixReal ( ob' T // F ) ( abre MTRW )

;

FLASHPTR DoOldMatrixReal ( ob' T // F ) ( edita ob en MTRW )

( ob' T // F )

ITE

:: DUPTYPELIST? ( ob' flag )

ITE

:: INNERDUP ( ob1...obn #f #f )

ZERO\_DO (DO)

ROLL ( ...obi )

INNERCOMP ( ... ob1'...obm' #c )

TYPEMATRIX\_ ( ... ob1'...obm' #c #2686 )

COMPN\_ ( ... 1DMATRIX )

ISTOP@ ( ... 1DMATRIX #f )

LOOP

TYPEMATRIX\_ ( 1DMATRIX1...1DMATRIXn #f #2686 )

COMPN\_ ( 2DMATRIX1 )

FALSE ( 2DMATRIX1 F )

;

TrueTrue ( ob' T T )

;

FalseTrue

UNTIL

( RealArry T // F )

;

;

\* Busca un ARREGLO en el FILER

\* Inicia en el directorio HOME

\* Se puede desplazar por todos lados

\* No permite selección múltiple

\* Solo retorna un arreglo real seguido de TRUE.

\* Si se cancela, sólo retorna FALSE.

NULLNAME Filer\_Home\_S1\_NoDir\_Arry ( -> ob T // F )

::

{ # 29E8 # 2A96 } ( Filer\_Tipos: ARRAY y Directorios )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* HAY 6 TECLAS DEL MENU

\* La 1º es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* Esta lista es para quitar la tecla +/- "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT28 ( Tecla\_Atajo: BINT28 = #1C = Tecla +/- "CHECK" )

}

\* Esta lista es para quitar la tecla ALPHA + ENTER "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 133 ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista no hace nada

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

}

\* Esta lista es para la tecla de menú F5 "CANCL"

{ # DF2C ( Nombre: # DF2C "CANCL" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT37 ( Acción: BINT37= Sale del filer dejando FALSE en la pila )

}

\* Esta lista es para la tecla de menú F6 "OK"

\* También es para asignarla a la tecla ENTER

{ # DF2D ( Nombre: # DF2D "OK" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT17 ( Acción: BINT17= Prog Pers que pone Ruta, Objeto y Nombre )

:: ( Ruta Objeto Nombre )

DROPSWAPDROP ( Objeto )

DUPTYPERRP? ( Objeto flag ) ( Evita finalizar con Directorio )

caseDrpBadKy ( )

DUP ( Objeto Objeto )

TYPERARRY? ( Objeto flag )

NOTcasedrop

:: "Escoge un arreglo real"

FlashWarning

; ( )

( RealArry )

' TakeOver ( RealArry TakeOver )

; ( Programa\_Extra: Sale del filer, deja en la pila el Objeto y TRUE )

# 33 ( Tecla\_Atajo: BINT51 = #33 = Tecla ENTER )

}

}

FLASHPTR FILER\_MANAGERTYPE ( ob T // F // :0:{} )

TRUE

EQUAL ( ob T // F )

;

Ejemplo 36 Ifmain

**Campo COMBOCHOOSE transformado en campo COMBOFILER que contiene a un arreglo real que cumpla una condición.**

**Manejando los mensajes 5, 6, 17 y 25 en el campo.**

**Manejando los mensajes 0 y 16 en el formulario.**

En este ejemplo hay un campo COMBOCHOOSE🡪COMBOFILER que contiene a un arreglo real que cumple la condición de que debe de tener a 2 columnas y el número de filas debe ser mayor o igual a 2. Además los elementos del arreglo deben ser mayores a 0.

En el campo COMBOCHOOSE🡪COMBOFILER se llaman a los mensajes 5, 6, 17 y 25.

Con el mensaje 5, podemos validar la línea de comandos al escribir un argumento desde el editor de texto.

Con el mensaje 6, podemos ver la formación como un grob. Ya no será necesario el uso del parámetro **#Decompile** en ese campo.

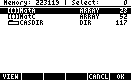
Con el mensaje 17, se podrá llamar al FILER cada vez que se presione la tecla CHOOS (F2).

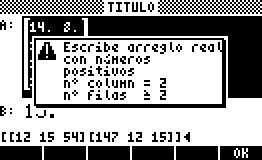
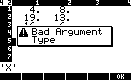
Con el mensaje 25, podemos editar la formación en el MTRW al presionar EDIT.

En el Message Handler del formulario se llaman a los mensajes 0 y 16.

Con el mensaje 0, podremos hacer que los campos COMBOCHOOSE funcionen correctamente.

Con el mensaje 16, podemos evitar la finalización del formulario (al presionar ENTER u OK), si hay un campo que esté vacío.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMaRACond ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

"B:" BINT0 BINT53 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: CAMPO COMBOCHOOSE->COMBOFILER. CONTIENE A UN ARREGLO REAL

' MH5\_6\_17\_25\_RACond ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT121 ( #w: 131-#x para 1 columna )

BINT41 ( #h: )

BINT2 ( #TipoDeCampo: COMBOCHOOSE )

{ BINT3 } ( TiposPermitidos: Arreglos reales y matrices simbólicas )

MINUSONE ( #Decompile: No es necesario, pues mensaje 6 es manejado )

"Escribe arreglo real 2 columnas" ( Ayuda )

MINUSONE ( ChooseData: No se necesita en campo COMCHOOSE->COMFILER )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

MINUSONE ( ValorReset: campo COMCHOOSE->COMFILER en blanco )

MINUSONE ( ValorInicial: campo COMCHOOSE->COMFILER en blanco )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT10 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT52 ( #y: )

BINT121 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"B: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona alguna tecla

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Permite que se puedan usar campos COMBOCHOOSE

BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

FALSE ( #ct #p F )

LAM 'CurrentField ( #ct #p F #c )

FLASHPTR IfGetFieldType ( #ct #p F #TipoCampo )

#2= ( #ct #p F flag ) ( CampoActual=COMBOCHOOSE? )

NOT?SEMI

( #ct #p F )

R> ( #ct #p F prog )

FLASHPTR 002 0A5 ( #ct #p F $ ) ( Equivale al comando x->H )

"52133" ( #ct #p F $ "52133" ) ( dirección del BINT3 )

"B1133" ( #ct #p F $ "52133" "B1133" ) ( dirección de BINT2 )

FLASHPTR 00F 01A ( #ct #p F $' %3 ) ( Equivale al comando xSREPL )

DROP ( #ct #p F $' )

FLASHPTR 002 0A4 ( #ct #p F prog' ) ( Equivale al comando xH-> )

>R ( #ct #p F )

;

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Si hay algún campo en blanco, evita su finalizacion con ENTER u OK

\*\*\* y muestra el mensaje "Escribe todos los datos"

BINT16 #=casedrop ( F ) ( -> flag T // F )

:: ( )

FLASHPTR IfPutFieldsOnStack ( ob1 ob2...obn )

FLASHPTR IfGetNbFields ( ob1 ob2...obn #n )

{}N ( { ob1 ob2...obn } )

' xNOVAL ( { ob1 ob2...obn } xNOVAL )

SWAP ( xNOVAL { ob1 ob2...obn } )

FLASHPTR ListPos ( #pos/#0 )

#0=ITE

TrueTrue

:: "Escribe todos los datos" FlashWarning FalseTrue ;

( flag T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 T // F )

;

NULLNAME MH5\_6\_17\_25\_RACond

::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia.

\* Convierte valor ext en valor int (será guardado en la pila virtual)

BINT5 #=casedrop ( C ) ( Vext -> Vint flag )

:: ( valor )

DUP ( valor valor )

' xNOVAL ( valor valor NOVAL )

EQUAL ( valor flag )

caseTRUE ( sale con TRUE )

( valor )

DUP ( valor valor )

TestRealArry ( valor flag )

NOT\_IT

:: # 203 ERROROUT ;

( valor )

TRUE ( valor T )

;

\* Cuando se inicia el IfMain y cuando el valor del campo cambia

\* Debe mostrar el grob del campo (usando el comando ˆIfSetGrob)

BINT6 #=casedrop ( C ) ( #c Vext -> flag T // #c Vext F )

:: ( #c valor )

DUP ( #c valor valor )

' xNOVAL ( #c valor valor xNOVAL )

EQUAL ( #c valor flag )

caseFALSE

( #c valor )

TRUESWAP\_ ( #c T valor )

ob>grobmini ( #c T grob )

3PICK3PICK ( #c T grob #c T )

NULLPAINT\_ ( #c T grob #c T grob )

FLASHPTR IfSetGrob ( #c T grob )

FLASHPTR IfSetGrob ( )

TrueTrue ( T T )

;

\* Cuando se presiona la tecla CHOOS en un campo CHOOSE o COMBOCHOOSE.

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

BINT17 #=casedrop ( C/F ) ( -> T // F )

:: ( )

Filer\_Home\_S1\_NoDir\_ArryCond ( RealArry T // F )

IT

FLASHPTR IfSetCurrentFieldValue

( )

TRUE ( T )

;

\* Cuando se presiona EDIT en un campo TEXTO, COMBOCHOOSE o CHECK.

\* Primero es llamado el mensaje del campo, luego el del formulario.

\* Si se maneja el mensaje del campo, ya no es llamado el mensaje del

\* formulario ni tampoco el código estándar.

\* Si se maneja el mensaje del formulario, ya no se llama al códig est

\* Para manejar a este mensaje debes dejar TRUE.

\* La entrada es el valor externo del campo actual.

\* Puedes iniciar la línea de edición con ese valor.

\* O también puedes editar el valor del campo actual con otro editor

\* y luego modificar el campo actual.

\* O también puedes editar el valor del campo actual con otro editor

\* y dejar el valor modificado en la pila seguido por TRUE

\* (ese valor será puesto en el campo actual de forma automática).

\* Si este mensaje es manejado, ya no se llamará al mensaje número 23.

BINT25 #=casedrop ( C/F ) ( Vext -> T // Vext T // Vext F )

:: ( ob )

BEGIN

:: ( ob )

EditaArregloRealEnMTRW ( RealArry T // F )

NOTcaseTRUE ( Sale con: T )

( ob )

DUP ( ob ob )

TestRealArry ( RealArry T // ob F )

ITE

TRUE

:: AlertaRACond FALSE ;

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( RealArry // Nada )

TRUE ( RealArry T // T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

\* CONVIERTE ob A GROB PEQUEÑO USANDO MINIFUENTE

\* Intenta convertir a grob con FLASHPTR EQW3GROBmini

\* Si no es posible hace lo siguiente:

\* Si ob es un grob, no hace nada.

\* Si ob es una cadena, hace $>grobCR

\* Si ob es de otro tipo, hace FLASHPTR FSTR11 $>grobCR

NULLNAME ob>grobmini ( ob -> grob )

:: ( ob )

BEGIN

:: DUP ( ob ob )

FLASHPTR EQW3GROBmini ( ob ext grob #0 // ob ob #1/2 )

BINT0 #=casedrop

:: UNROT2DROP TRUE ; ( OBS: SALE CON grob T ) ( SALE DE BUCLE )

( ob ob #1/2 )

BINT1 #=case

:: GARBAGE DROPFALSE ; ( OBS: SALE CON ob F ) ( REPITE BUCLE )

( ob ob )

TYPEGROB? ( ob flag )

caseTRUE

( ob )

DUPTYPECSTR? ( ob flag )

?SKIP

FLASHPTR FSTR11

( $ )

$>grobCR ( grob )

TRUE ( grob T ) ( SALE DE BUCLE )

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( grob )

;

\* Si en la pila se encuentra xNOVAL, abre el MTRW.

\* Si hay otro objeto, lo edita en MTRW (de ser posible)

\* Retorna un arreglo real y TRUE o sólo FALSE

NULLNAME EditaArregloRealEnMTRW ( ob -> RealArry T // F )

::

RunSafeFlags

:: BEGIN

BINT91 ClrSysFlag ( ob ) ( 91 MTRW: no list of list )

DUP ( ob ob )

' xNOVAL ( ob ob xNOVAL )

EQUAL ( ob flag )

ITE

:: DROP ( )

FLASHPTR DoNewMatrixReal ( ob' T // F ) ( abre MTRW )

;

FLASHPTR DoOldMatrixReal ( ob' T // F ) ( edita ob en MTRW )

( ob' T // F )

ITE

:: DUPTYPELIST? ( ob' flag )

ITE

:: INNERDUP ( ob1...obn #f #f )

ZERO\_DO (DO)

ROLL ( ...obi )

INNERCOMP ( ... ob1'...obm' #c )

TYPEMATRIX\_ ( ... ob1'...obm' #c #2686 )

COMPN\_ ( ... 1DMATRIX )

ISTOP@ ( ... 1DMATRIX #f )

LOOP

TYPEMATRIX\_ ( 1DMATRIX1...1DMATRIXn #f #2686 )

COMPN\_ ( 2DMATRIX1 )

FALSE ( 2DMATRIX1 F )

;

TrueTrue ( ob' T T )

;

FalseTrue

UNTIL

( RealArry T // F )

;

;

\* Busca un ARREGLO REAL en el FILER que cumpla el test: TestRealArry

\* Inicia en el directorio HOME

\* Se puede desplazar por todos lados

\* No permite selección múltiple

\* Solo retorna el arreglo real que cumpla condición seguido de TRUE

\* Si se cancela, sólo retorna FALSE.

NULLNAME Filer\_Home\_S1\_NoDir\_ArryCond ( -> ob T // F )

::

{ # 29E8 # 2A96 } ( Filer\_Tipos: ARRAY y Directorios )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* HAY 6 TECLAS DEL MENU

\* La 1º es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* Esta lista es para quitar la tecla +/- "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT28 ( Tecla\_Atajo: BINT28 = #1C = Tecla +/- "CHECK" )

}

\* Esta lista es para quitar la tecla ALPHA + ENTER "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 133 ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista no hace nada

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

}

\* Esta lista es para la tecla de menú F5 "CANCL"

{ # DF2C ( Nombre: # DF2C "CANCL" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT37 ( Acción: BINT37= Sale del filer dejando FALSE en la pila )

}

\* Esta lista es para la tecla de menú F6 "OK"

\* También es para asignarla a la tecla ENTER

{ # DF2D ( Nombre: # DF2D "OK" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT17 ( Acción: BINT17= Prog Pers que pone Ruta, Objeto y Nombre )

:: ( Ruta Objeto Nombre )

DROPSWAPDROP ( Objeto )

DUPTYPERRP? ( Objeto flag ) ( Evita finalizar con Directorio )

caseDrpBadKy ( )

DUP ( Objeto Objeto )

TestRealArry ( Objeto flag )

NOTcasedrop

AlertaRACond ( )

( Objeto )

' TakeOver ( Objeto TakeOver )

; ( Programa\_Extra: Sale del filer, deja en la pila el Objeto y TRUE )

# 33 ( Tecla\_Atajo: BINT51 = #33 = Tecla ENTER )

}

}

FLASHPTR FILER\_MANAGERTYPE ( ob T // F // :0:{} )

TRUE

EQUAL ( ob T // F )

;

\* Realiza un test al objeto del nivel uno de la pila

\* Retorna TRUE si en la pila hay un arreglo real con números positivos, de

\* 2 dimensiones con 2 columnas y con un número de filas mayor o igual a 2.

NULLNAME TestRealArry ( ob -> flag )

::

DUP TYPERARRY? ( ob flag )

NOTcase

DROPFALSE

( RealArry )

DUP ( RealArry RealArry )

FLASHPTR MDIMS ( [[%]] #filas #cols T // [%] #elem F )

NOTcase

2DROPFALSE

( [[%]] #filas #cols )

#2= ( [[%]] #filas flag )

SWAP ( [[%]] flag #filas )

BINT1 #> ( [[%]] flag flag' )

AND ( [[%]] flag'' )

NOTcase

DROPFALSE

( [[%]] ) ( arreglo real de 2 dimensiones en la pila )

FLASHPTR XEQARRY> ( %1...%n {%f %c} )

INCOMPDROP ( %1...%n %f %c )

%\* ( %1...%n %f·c )

COERCE ( %1...%n #f·c )

ONE\_DO (DO) %MIN LOOP

( %' )

%0> ( flag )

;

NULLNAME AlertaRACond ( -> )

::

"Escribe arreglo real con números positivos\0Anº column = 2\0Anº filas Š 2"

FlashWarning

;

Ejemplo 37 Ifmain

**Campo COMBOCHOOSE transformado en campo COMBOFILER que contiene a una función f(X)**

**Manejando los mensajes 5, 6, 17 y 25 en el campo.**

**Manejando los mensajes 0 y 16 en el formulario.**

En este ejemplo hay un campo COMBOCHOOSE🡪COMBOFILER que contiene a una función f(X) que cumple la condición de que debe contener sólo a la variable X o a ninguna variable.

En el campo COMBOCHOOSE🡪COMBOFILER se llaman a los mensajes 5, 6, 17 y 25.

Con el mensaje 5, podemos validar la línea de comandos al escribir un argumento desde el editor de texto.

Con el mensaje 6, podemos ver la formación como un grob. Ya no será necesario el uso del parámetro **#Decompile** en ese campo.

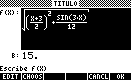
Con el mensaje 17, se podrá llamar al FILER cada vez que se presione la tecla CHOOS (F2).

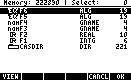
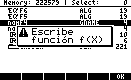
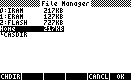
Con el mensaje 25, podemos editar la formación en el EQW al presionar EDIT.

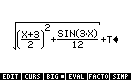
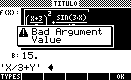
En el Message Handler del formulario se llaman a los mensajes 0 y 16.

Con el mensaje 0, podremos hacer que los campos COMBOCHOOSE funcionen correctamente.

Con el mensaje 16, podemos evitar la finalización del formulario (al presionar ENTER u OK), si hay un campo que esté vacío.



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME IfMaFuncCond ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"f(X):" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

" B:" BINT0 BINT53 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 )

\* CAMPO 0: CAMPO COMBOCHOOSE->COMBOFILER. CONTIENE A UN ARREGLO REAL

' MH5\_6\_17\_25\_FuncCond ( MH del campo )

BINT22 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT9 ( #y: 9 para fila 1/6 )

BINT109 ( #w: 131-#x para 1 columna )

BINT41 ( #h: )

BINT2 ( #TipoDeCampo: COMBOCHOOSE )

{ BINT0 BINT6 BINT9 # FF } ( TiposPermitidos: %, id, symb, Z )

MINUSONE ( #Decompile: No es necesario, pues mensaje 6 es manejado )

"Escribe f(X)" ( Ayuda )

MINUSONE ( ChooseData: No se necesita en campo COMCHOOSE->COMFILER )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

MINUSONE ( ValorReset: campo COMCHOOSE->COMFILER en blanco )

MINUSONE ( ValorInicial: campo COMCHOOSE->COMFILER en blanco )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT22 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT52 ( #y: )

BINT109 ( #w: 131-#x para 1 columna )

BINT8 ( #h: 8 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( #Decompile: 4=STD No1=SysF ) ( ) ( "$" 'id' 'u' )

"B: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: Es ignorado en IfMain )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona alguna tecla

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Permite que se puedan usar campos COMBOCHOOSE

BINT0 #=casedrop ( C/F ) ( #ct #p -> Acción T // #ct #p F )

:: ( #ct #p )

FALSE ( #ct #p F )

LAM 'CurrentField ( #ct #p F #c )

FLASHPTR IfGetFieldType ( #ct #p F #TipoCampo )

#2= ( #ct #p F flag ) ( CampoActual=COMBOCHOOSE? )

NOT?SEMI

( #ct #p F )

R> ( #ct #p F prog )

FLASHPTR 002 0A5 ( #ct #p F $ ) ( Equivale al comando x->H )

"52133" ( #ct #p F $ "52133" ) ( dirección del BINT3 )

"B1133" ( #ct #p F $ "52133" "B1133" ) ( dirección de BINT2 )

FLASHPTR 00F 01A ( #ct #p F $' %3 ) ( Equivale al comando xSREPL )

DROP ( #ct #p F $' )

FLASHPTR 002 0A4 ( #ct #p F prog' ) ( Equivale al comando xH-> )

>R ( #ct #p F )

;

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Si hay algún campo en blanco, evita su finalizacion con ENTER u OK

\*\*\* y muestra el mensaje "Escribe todos los datos"

BINT16 #=casedrop ( F ) ( -> flag T // F )

:: ( )

FLASHPTR IfPutFieldsOnStack ( ob1 ob2...obn )

FLASHPTR IfGetNbFields ( ob1 ob2...obn #n )

{}N ( { ob1 ob2...obn } )

' xNOVAL ( { ob1 ob2...obn } xNOVAL )

SWAP ( xNOVAL { ob1 ob2...obn } )

FLASHPTR ListPos ( #pos/#0 )

#0=ITE

TrueTrue

:: "Escribe todos los datos" FlashWarning FalseTrue ;

( flag T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del IfMain )

FLASHPTR IfMain ( ob1 ob2 T // F )

;

NULLNAME MH5\_6\_17\_25\_FuncCond

::

\* Cuando se inicia el IfMain y cuando el valor del campo cambia.

\* Convierte valor ext en valor int (será guardado en la pila virtual)

BINT5 #=casedrop ( C ) ( Vext -> Vint flag )

:: ( valor )

DUP ( valor valor )

' xNOVAL ( valor valor NOVAL )

EQUAL ( valor flag )

caseTRUE ( sale con TRUE )

( valor )

DUP ( valor valor )

TestFuncion ( valor flag )

NOT\_IT

:: # 203 ERROROUT ;

( valor )

TRUE ( valor T )

;

\* Cuando se inicia el IfMain y cuando el valor del campo cambia

\* Debe mostrar el grob del campo (usando el comando ˆIfSetGrob)

BINT6 #=casedrop ( C ) ( #c Vext -> flag T // #c Vext F )

:: ( #c valor )

DUP ( #c valor valor )

' xNOVAL ( #c valor valor xNOVAL )

EQUAL ( #c valor flag )

caseFALSE

( #c valor )

TRUESWAP\_ ( #c T valor )

ob>grobmini ( #c T grob )

3PICK3PICK ( #c T grob #c T )

NULLPAINT\_ ( #c T grob #c T grob )

FLASHPTR IfSetGrob ( #c T grob )

FLASHPTR IfSetGrob ( )

TrueTrue ( T T )

;

\* Cuando se presiona la tecla CHOOS en un campo CHOOSE o COMBOCHOOSE.

\* CAMPO y FORM: sólo tendrá efecto el message handler del campo.

BINT17 #=casedrop ( C/F ) ( -> T // F )

:: ( )

Filer\_Home\_S1\_NoDir\_FuncCond ( RealArry T // F )

IT

FLASHPTR IfSetCurrentFieldValue

( )

TRUE ( T )

;

\* Cuando se presiona EDIT en un campo TEXTO, COMBOCHOOSE o CHECK.

\* Primero es llamado el mensaje del campo, luego el del formulario.

\* Si se maneja el mensaje del campo, ya no es llamado el mensaje del

\* formulario ni tampoco el código estándar.

\* Si se maneja el mensaje del formulario, ya no se llama al códig est

\* Para manejar a este mensaje debes dejar TRUE.

\* La entrada es el valor externo del campo actual.

\* Puedes iniciar la línea de edición con ese valor.

\* O también puedes editar el valor del campo actual con otro editor

\* y luego modificar el campo actual.

\* O también puedes editar el valor del campo actual con otro editor

\* y dejar el valor modificado en la pila seguido por TRUE

\* (ese valor será puesto en el campo actual de forma automática).

\* Si este mensaje es manejado, ya no se llamará al mensaje número 23.

BINT25 #=casedrop ( C/F ) ( Vext -> T // Vext T // Vext F )

:: ( ob )

BEGIN

:: ( ob )

EditaEnEQW ( RealArry T // F )

NOTcaseTRUE ( Sale con: T )

( ob )

DUP ( ob ob )

TestFuncion ( RealArry T // ob F )

ITE

TRUE

:: AlertaFuncCond FALSE ;

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( RealArry // Nada )

TRUE ( RealArry T // T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

\* CONVIERTE ob A GROB PEQUEÑO USANDO MINIFUENTE

\* Intenta convertir a grob con FLASHPTR EQW3GROBmini

\* Si no es posible hace lo siguiente:

\* Si ob es un grob, no hace nada.

\* Si ob es una cadena, hace $>grobCR

\* Si ob es de otro tipo, hace FLASHPTR FSTR11 $>grobCR

NULLNAME ob>grobmini ( ob -> grob )

:: ( ob )

BEGIN

:: DUP ( ob ob )

FLASHPTR EQW3GROBmini ( ob ext grob #0 // ob ob #1/2 )

BINT0 #=casedrop

:: UNROT2DROP TRUE ; ( OBS: SALE CON grob T ) ( SALE DE BUCLE )

( ob ob #1/2 )

BINT1 #=case

:: GARBAGE DROPFALSE ; ( OBS: SALE CON ob F ) ( REPITE BUCLE )

( ob ob )

TYPEGROB? ( ob flag )

caseTRUE

( ob )

DUPTYPECSTR? ( ob flag )

?SKIP

FLASHPTR FSTR11

( $ )

$>grobCR ( grob )

TRUE ( grob T ) ( SALE DE BUCLE )

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( grob )

;

\* Si en la pila se encuentra xNOVAL, abre el EQW.

\* Si hay otro objeto, lo edita en EQW (de ser posible)

NULLNAME EditaEnEQW ( ob -> ob' T // F )

:: DUP ( ob ob )

' xNOVAL ( ob ob xNOVAL )

EQUAL ( ob flag )

ITE

:: DROP ( )

FLASHPTR EQW3 ( ob' T // F ) ( abre EQW )

;

FLASHPTR EQW3Edit ( ob' T // F ) ( edita ob en EQW )

;

\* Busca una función f(X) en el FILER que cumpla el test: TestFuncion

\* Inicia en el directorio HOME

\* Se puede desplazar por todos lados

\* No permite selección múltiple

\* Solo retorna una función que cumpla condición seguido de TRUE

\* Si se cancela, sólo retorna FALSE.

NULLNAME Filer\_Home\_S1\_NoDir\_FuncCond ( -> ob T // F )

::

{ # 2933 # 2E48 # 2AB8 # 2614 # 2A96 } ( Filer\_Tipos: %, id, symb, Z, Direct )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* HAY 6 TECLAS DEL MENU

\* La 1º es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* Esta lista es para quitar la tecla +/- "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT28 ( Tecla\_Atajo: BINT28 = #1C = Tecla +/- "CHECK" )

}

\* Esta lista es para quitar la tecla ALPHA + ENTER "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 133 ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista no hace nada

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

}

\* Esta lista es para la tecla de menú F5 "CANCL"

{ # DF2C ( Nombre: # DF2C "CANCL" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT37 ( Acción: BINT37= Sale del filer dejando FALSE en la pila )

}

\* Esta lista es para la tecla de menú F6 "OK"

\* También es para asignarla a la tecla ENTER

{ # DF2D ( Nombre: # DF2D "OK" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT17 ( Acción: BINT17= Prog Pers que pone Ruta, Objeto y Nombre )

:: ( Ruta Objeto Nombre )

DROPSWAPDROP ( Objeto )

DUPTYPERRP? ( Objeto flag ) ( Evita finalizar con Directorio )

caseDrpBadKy ( )

DUP ( Objeto Objeto )

TestFuncion ( Objeto flag )

NOTcasedrop

AlertaFuncCond ( )

( Objeto )

' TakeOver ( Objeto TakeOver )

; ( Programa\_Extra: Sale del filer, deja en la pila el Objeto y TRUE )

# 33 ( Tecla\_Atajo: BINT51 = #33 = Tecla ENTER )

}

}

FLASHPTR FILER\_MANAGERTYPE ( ob T // F // :0:{} )

TRUE

EQUAL ( ob T // F )

;

\* Realiza un test al objeto del nivel uno de la pila

\* Retorna TRUE si en la pila hay una función f(X)

\* Reales, Enteros, ids o symbs que contengan a la variable X

\* o a ninguna variable

NULLNAME TestFuncion ( ob -> flag )

::

DUPTYPEREAL?

case

DROPTRUE

( ob )

DUPTYPEZINT?

case

DROPTRUE

( ob )

DUPTYPESYMB?

OVER

TYPEIDNT?

OR ( ob flag )

NOTcase

DROPFALSE

( ob )

FLASHPTR LIDNText ( {vars} )

DUPNULL{}?

SWAP

{ ID X }

EQUAL

OR ( flag )

;

NULLNAME AlertaFuncCond ( -> )

::

"Escribe función f(X)"

FlashWarning

;

Capítulo 38  
Formularios de Entrada con DoInputForm

En el capítulo anterior vimos como crear formularios de entrada con el comando ^IfMain, el cual está presente a partir de la HP 49g.

En este capítulo veremos otra manera de crear formularios de entrada. Usaremos ahora el comando DoInputForm. Este comando ya estaba presente en la HP 48. Con el comando DoInputForm podemos hacer algunas cosas más fácilmente. Los argumentos son los mismos que los del comando ^IfMain, pero los message handlers son diferentes. Además hay muchas diferencias más, las cuales veremos en su totalidad en este capítulo.

Los argumentos del comando DoInputForm son los mismos que los del comando ^IfMain como se muestra en la tabla de abajo.

**Parámetro Descripción**

**etiqueta\_1**

. . . Definiciones de las etiquetas

**et**ique**ta\_n**

**campo\_1**

. . . Definiciones de los campos

**campo\_n**

**#etiquetas** Número de etiquetas

**#campos** Número de campos

**MessageHandlerIfMain**  Ver sección 38.4 abajo

**Titulo**  Título a ser mostrado en la parte alta de la pantalla.

38.1 Definiciones de Etiquetas

Cada definición de una etiqueta consiste de tres argumentos:

**Parametro Descripción**

**cuerpo\_etiqueta** Cadena

**#x\_posición** Coordenada X

**#y\_posición** Coordenada Y

**cuerpo\_etiqueta** es una cadena o un bint (no grob). Si pones una cadena, esta sera convertida a gro nb usando la minifuente. También puede ser un bint, en este caso se mostrará el mensaje de error (ver Apéndice E) correspondiente a ese bint.

La etiqueta sera mostrada en las coordenadas especificadas. Estas coordenadas son dos bints que representan las posiciones x e y de la etiqueta en la pantalla. La esquina superior izquierda tiene coordenadas (0, 0) y las coordenadas x e y se incrementan hacia la derecha y hacia abajo respectivamente.

38.2 Definiciones de Campos

Cada definición de un campo consiste de trece argumentos:

**Parametro Descripción**

**MessageHandlerCampo** Ver sección 38.4 abajo.

**#x\_posición** Coordenada X

**#y\_posición** Coordenada Y (normalmente coord Y de etiqueta menos 1)

**#w\_ancho** Ancho del campo.

**#h\_altura** Altura del campo (usualmente 8 ó 6).

**#TipoDeCampo** Tipo de campo, ver abajo los valores válidos.

**TiposPermitidos** Lista con los tipos de objetos válidos para este campo.

**Decompile** Ver abajo.

**"Ayuda"** Cadena de ayuda para este campo.

**ChooseData** Ver abajo.

**ChooseDecompile** Ver abajo.

**ValorReset** Valor reset de este campo.

**ValorInicial** Valor inicial de este campo.

Los **Message Handler de los campos** serán descritos abajo.

Las **posiciones #x** e **#y** especifican donde aparecerán los campos. . El contenido del campo será mostrado 1 píxel a la derecha de X y 1 píxel debajo de Y.

Las dimensiones **#w** y **#h** son también dos bints, los cuales especifican el tamaño del campo. Por ejemplo, campos que muestran objetos con la minifuente pueden tener una altura de 7. Campos que muestran objetos con fuente de tamaño 8 pueden tener altura 9. En campos **CHECKBOX #w** debe ser BINT6 y **#h** debe ser BINT9.

El parámetro **#TipoDeCampo** es un bint que define el tipo del campo. En DoInputForm hay 6 tipos de campos.

**Valor decimal Tipo de campo**

BINT1 BINT3 **TEXTO**: el usuario puede ingresar cualquier cosa.

BINT12 **CHOOSE**: el usuario debe seleccionar de una lista de valores válidos.

BINT20 **FILER**: el usuario puede escoger un objeto desde el filer.

BINT32 **CHECKBOX**: es una casilla de verificación que el usuario puede marcar o desmarcar.

BINT13 BINT15 **COMBOCHOOSE**: el usuario puede seleccionar de una lista de valores o ingresar otro. Combinación de CHOOSE+TEXTO.

BINT21 BINT23 **COMBOFILER**: el usuario puede ingresar cualquier cosa o

seleccionar un objeto a partir del filer. Combinación de FILER+TEXTO.

Los valores BINT3, BINT15 y BINT23 del parámetro #TipoDeCampo, corresponden a los campos **texto algebraico**, **combochoose algebraico** y **combofiler algebraico** respectivamente. En estos campos, al iniciar una línea de edición nueva, se activa automáticamente el modo algebraico y al texto ingresado se le adicionarán comillas simples de manera automática en los extremos (esto se cumplirá cuando la línea de edición no se inicie con algo que contenga a **[**, **«**, **"**, **#**, **:** o **{**).

El parámetro TiposPermitidos es usado en los campos **TEXTO**, **FILER**, **COMBOCHOOSE** y **COMBOFILER**. Es una lista de bints, que representan los tipos permitidos de los objetos que pueden ser ingresados en ese campo. Sólo son permitidos objetos de los tipos mostrados en la siguiente tabla. Son los mismos tipos que acepta el comando xINFORM de User RPL (excepto BINT255d), pero en lugar de números reales estos tipos se indican con bints.

Por ejemplo, si queremos que un campo acepte reales o complejos, su parámetro TiposPermitidos debe ser { BINT0 BINT1 }. Debemos tener presente que el comando DoInputForm no aceptaría la siguiente lista: { # 0 # 1 }.

|  |  |
| --- | --- |
| **BINT** | **TIPOS DE OBJETOS** |
| BINT0 | Reales |
| BINT1 | Complejos |
| BINT2 | Cadenas |
| BINT3 | Arreglos reales (no matrices simbólicas) |
| BINT4 | Arreglos no reales |
| BINT5 | Listas |
| BINT6 | Nombres globales |
| BINT8 | Programas |
| BINT9 | Simbólicos |
| BINT10 | Cadenas hexadecimales |
| BINT13 | Unidades |

Para los campos **CHOOSE** y **CHECKBOX** el parámetro TiposPermitidos debe ser MINUSONE. También puedes especificar MINUSONE para campos **TEXTO**, **FILER**, **COMBOCHOOSE** y **COMBOFILER**, dando como resultado que serán aceptados los objetos de todos los tipos existentes (no sólo los de la tabla mostrada).

El parámetro Decompile es un bint o un programa y especifica como serán mostrados en la pantalla los objetos ingresados en el campo. Su acción es parecida a la del parámetro Converter del browser 48. Puede ser:

a) Un PROGRAMA. Si el parámetro es un programa, su diagrama de pila debe ser

( ob 🡺 $ )

Donde ob es el objeto contenido en el campo.

Un valor muy útil para este parámetro puede ser el comando DO>STR.

b) Un BINT. Si el parámetro es un bint, puede ser cualquiera de estos:

• #1: No se hará ninguna descompilación al elemento.

Usar este bint cuando en el campo sólo pueden haber cadenas.

Las cadenas se mostrarán sin comillas dobles.

• #2: Los números se mostrarán en el formato numérico actual.

Los ids y los objetos unidad se mostrarán sin comillas.

Las cadenas se mostrarán con comillas dobles.

• #4: Los números se mostrarán en el formato numérico estándar.

Los ids y los objetos unidad se mostrarán sin comillas.

Las cadenas se mostrarán con comillas dobles.

También puedes agregar (sumando al anterior) uno de estos bints:

• #16: Cuando el ítem es una lista y quieres mostrar el primer elemento de esta lista.

• #32: Cuando el ítem es una lista y quieres mostrar el segundo elemento de esta lista.

• #8: Se mostrará sólo el primer carácter.

Por ejemplo, si en un campo siempre habrá un objeto lista de esta forma {$ ob}, puedes fijar el parámetro Decompile como 17 (16+1), lo cual significa que se tomará el primer elemento de la lista y esta cadena se mostrará directamente.

OBSERVACIONES:

También puedes especificar el parámetro Decompile como BINT1. En este caso, ninguna descompilación es hecha y sólo puedes usar cadenas y bints (se mostrará la cadena correspondiente al mensaje de error de dicho bint). En ambos casos, la cadena se mostrará sin las comillas dobles.

Cuando el parámetro Decompile es un bint, debe contener necesariamente como uno de sus sumandos a BINT2 o a BINT4, excepto cuando el objeto que se mostrará es una cadena o un bint.

El valor del parámetro Decompile no tiene ningun significado en un campo **CHECKBOX**.

No hay ningún valor para el parámetro Decompile que permita mostrar un objeto con la minifuente. Para mostrar el contenido de un campo con minifuente, puedes usar el message handler 7 o el 9.

Si usas el message handler 7, 9 o 10, puedes pasar por alto al parámetro Decompile.

En campos **CHOOSE**,el parámetro Decompile es usado en la búsqueda alfabética.

El parámetro "Ayuda" especifica la cadena que sera mostrada en la parte inferior de la pantalla cuando ese campo tenga el enfoque. Este parámetro también puede ser un bint (a diferencia del comando ^IfMain que sólo acepta una cadena).

El parámetro ChooseData es usado solamente en campos **CHOOSE**, **COMBOCHOOSE**, **FILER**, y **COMBOFILER**. Para otros campos puedes poner MINUSONE como el valor de este parámetro.

a) En campos **CHOOSE** y **COMBOCHOOSE**, ChooseData es la lista de valores que serán presentados al usuario para que este seleccione (DoInputForm usa el browser 48).

a.1) Cuando usas un Decompile igual a 1, puedes usar un ChooseData como este:

{ "cadena1" "cadena2" ... }e

a.2) En cambio, si usas un Decompile igual a 17, puedes usar un ChooseData asi:

{ { "cadena1" <cuerpo1> } { "cadena2" <cuerpo2> } {...} ... }

De esta manera, solo el primer objeto de cada lista sera mostrado, pero la lista completa sera retornada (como al usar el comando CHOOSE de User RPL).

b) En campos **FILER** y **COMBOFILER**, ChooseData contiene a los parámetros que usará el comando ^BrowseMem.1 (ver seccion 29.2) para mostrar objetos desde el filer. Puede ser:

b.1) Una lista con 4 de los 5 argumentos de ^BrowseMem.1:

{Ob2 flagTeclaCheck TiposPermitidos flagObjetoNombre}

b.2) Una lista con 2 de los 5 argumentos de ^BrowseMem.1:

{Ob2 flagTeclaCheck}

TiposPermitidos en el filer será igual a TiposPermitidos del campo.

flagObjetoNombre será TRUE (se retornarán objetos y no nombres).

b.3) **MINUSONE**

flagTeclaCheck será FALSE (no estará disponible la tecla CHECK en el filer).

TiposPermitidos en el filer será igual a TiposPermitidos del campo.

flagObjetoNombre será TRUE (se retornarán objetos y no nombres).

El parámetro ChooseDecompile es requerido solamente en campos **CHOOSE** y **COMBOCHOOSE**. Puede ser un programa o un bint. Para otros campos puedes poner MINUSONE como el valor de este parámetro. La misión de este parámetro es convertir cada valor de la lista Choosedata a cadena para que sea mostrado en la lista de valores que serán presentados al usuario para que este seleccione (DoInputForm usa el browser 48). Es decir, ChooseDecompile es el parámetro Converter del browser 48 (ver la seccion 35.3) que se abrirá al presionar la tecla CHOOS. Puede ser:

a) Un PROGRAMA. Si el parámetro es un programa, su diagrama de pila debe ser

( ob 🡺 $ )

b) Un BINT. Si el parámetro es un bint, puede ser cualquiera de estos:

• #1: No se hará ninguna descompilación al elemento.

Usar este bint cuando todos los elementos son cadenas.

Las cadenas se mostrarán sin comillas dobles.

• #2: Los números se mostrarán en el formato numérico actual.

Los ids se mostrarán sin comillas.

Las cadenas se mostrarán con comillas dobles.

• #4: Los números se mostrarán en el formato numérico estándar.

Los ids se mostrarán sin comillas.

Las cadenas se mostrarán con comillas dobles.

También puedes agregar (sumando al anterior) uno de estos bints:

• #16: Cuando el ítem es una lista y quieres mostrar el primer elemento de esta lista.

• #32: Cuando el ítem es una lista y quieres mostrar el segundo elemento de esta lista.

• #8: Se mostrará sólo el primer carácter.

Por ejemplo, si cada uno de los ítems son listas de esta forma {$ ob}, puedes fijar el parámetro Choosedecompile como 17 (16+1), lo cual significa que se tomará el primer elemento de la lista y esta cadena se mostrará directamente.

Los parámetros ValorReset y ValorInicial se refieren al valor del campo cuando es mostrado inicialmente y cuando es reseteado. Ambos deben ser objetos de algún tipo permitido para este campo.

Para un campo **CHOOSE** deben ser alguno de los elementos de la lista ChooseData.

Para un campo **CHECKBOX**, usa TRUE o FALSE.

Puedes dejar un campo **TEXTO**, **FILER**, **COMBOCHOOSE** o **COMBOFILER** en blanco, especificando MINUSONE como uno o ambos de estos parámetros.

38.3 Número de Etiquetas y de Campos

Estos son dos bints que representan el número de etiquetas y de campos del formulario de entrada. Observa que estos dos valores pueden ser diferentes, de manera que puedes tener etiquetas que sólo muestren algún tipo de información al usuario, o campos sin ninguna etiqueta asociada. Ambos números deben ser mayores que cero.

38.4 Message Handler

Como otras aplicaciones de la calculadora HP, los formularios de entrada también usan message handlers para permitir al programador un mayor control. Hay un message handler para cada campo, y también un message handler para el formulario de entrada. Los message handler son llamados cuando algo “interesante” ocurre en el campo o en el formulario, y durante el inicio del formulario de entrada. Como sucede con otros message handlers, el programa proporcionado es llamado con un número (un bint) en el nivel uno, y a veces otros parámetros.

Si el programa message handler maneja al mensaje correspondiente al bint proporcionado, entonces debería retornar lo que sea que pida ese mensaje (a veces nada).

Si el programa message handler no maneja al mensaje que corresponde al bint proporcionado, entonces el programa debe borrar ese bint de la pila y poner FALSE, dejando los argumentos restantes (si los hubiera) en su lugar.

Por lo tanto un programa message handler que no maneja ningún mensaje es simplemente DROPFALSE, el cual, como sabemos, puede ser puesto en la pila con el comando 'DROPFALSE.

En el cuerpo de un message handler, los comandos listados en la sección de referencia de abajo pueden ser usados para conseguir información desde el formulario de entrada o para modificar este. La sección 38.8.2 describirá cada uno de los message handler disponibles en DoInputForm.

Esta es una plantilla para un programa message handler con tres mensajes:

' :: BINT1 #=casedrop ( Mensaje 1 ) ( puede ser cualquier otro mensaje )

::

\* Código para el mensaje. No olvidar retornar TRUE.

;

BINT2 #=casedrop ( Mensaje 2 ) ( puede ser cualquier otro )

::

\* Código para el mensaje.

;

\* Y posiblemente más mensajes.

DROPFALSE ( indica que otros mensajes no serán llamados )

;

• Los message handler en DoInputForm van desde el 1 hasta el 56.

• En campos **CHOOSE** y **COMBOCHOOSE**, puedes tambien usar los mensajes 57 al 96 del browser 48 (serán llamados al mostrar las opciones al presionar la tecla CHOOS y también al mostrar los tipos permitidos al presionar la tecla TYPES).

38.5 El Título del Formulario de Entrada

El título del formulario de entrada sera mostrado en la parte superior de la pantalla. Puede ser una cadena o un bint (no grob).

* Si es una cadena esta será mostrada con minifuente. Esta cadena debe tener como máximo 32 caracteres (el ancho de la pantalla). Si tiene más caracteres, no se podrá ver el título.
* Si es un bint, será mostrado el mensaje de error correspondiente a ese bint.

38.6 Resultados del Formulario de Entrada

La pila de salida, si el usuario salió del formulario de entrada con la tecla ENTER es:

N+1: Valor\_Campo\_1

N: Valor\_Campo\_2

. . . . .

2: Valor\_Campo\_n

1: TRUE

Si CANCEL fue usado para salir del formulario de entrada, entonces sólo FALSE es retornado en la pila.

Si un campo está vacío, entonces MINUSONE es retornado como el valor de este campo.

38.7 NULLLAMs Usados por DoInputForm

El comando DoInputForm usa nombres locales.

En su entorno temporal tiene 13•NCampos+3•NEtiq+15 variables locales, de los cuales son listados los primeros 15 en esta página.

Además, también otro nombre local (LAM 'IF) es creado en un entorno temporal anterior al descrito.

**LAM Descripción Tipo**

1 Contador usado por CACHE n/a

2 Igual a 13•NCampos+16 #

3 Igual a 13•NCampos+3•NEtiq+16 #

4 Número del campo de interés al usar algunos comandos. #

5 Número del campo actual (aquel que tiene el enfoque) #

Para el primer campo, este valor es **uno**.

6 En campos algebraicos, es la asignación por defecto de la ob

tecla que inició la línea de edición más reciente.

7 Indice de la 1º tecla de menu en la página actual de menú. #

1 para la primera, 7 para la segunda, 13 para la tercera, etc.

8 Menú {},::

9 Lista con flags. Tiene NCampos+1 elementos {flag}

Los flags correspondientes a cada campo son los de las

posiciones 2,3, … , NC+1

Si todos los flags de las posiciones 2,3, … , NC+1 son T,

entonces el flag de la ubicación 1 es T.

Si entre los flags de las posiciones 2,3, … , NC+1 hay por lo

menos un F, entonces el flag de la ubicación 1 es F.

10 Grob de las etiquetas. Tiene tamaño 131x39. grob

11 Condición de salida del POL. flag

12 Título del formulario $,#

13 Message Handler del formulario ::

14 Número de campos #

15 Número de etiquetas #

**LAM Descripción Tipo**

'IF Es un flag. flag

38.8 Referencia

36.8.1 Comandos Generales

**Direcc. Nombre Descripción**

2C371 DoInputForm ( l1..ln f1..fm #n #m msg $ 🡺 ob1..obn T )

( l1..ln f1..fm #n #m msg $ 🡺 F )

l = $ #x #y

f = MsgH #x #y #w #h #TipoCampo TipoObjetos #dec

$ayuda ChData #ChDec ValorReset ValorInicial

Inicia un formulario de entrada con el generador antiguo.

0410B0 ROMPTR 0B0 041 ( #campo 🡺 )

Llama al mensaje 47 del campo indicado.

0BE002 ^ChangeFocus ( #campo 🡺 )

Hace que un nuevo campo reciba el enfoque.

Llama a los mensajes 19 y 20.

0970B0 ROMPTR 0B0 097 ( #campo 🡺 )

Llama a ^ChangeFocus.

Además invierte los píxeles del campo que deja el enfoque y del campo que recibe el enfoque.

01F0B0 ROMPTR 0B0 01F ( #campo 🡺 )

Resetea un campo.

Llama al mensaje 33.

01C0B0 ROMPTR 0B0 01C ( 🡺 )

Resetea el campo actual y muestra el cambio en pantalla.

Llama al mensaje 33.

01D0B0 ROMPTR 0B0 01D ( 🡺 )

Resetea todos los campos.

Llama al mensaje 34.

0A50B0 ROMPTR 0B0 0A5 ( 🡺 )

Fija los flags correspondientes a todos los campos como TRUE en la lista del LAM9.

Luego actualiza el primer flag de esa lista como TRUE

0A60B0 ROMPTR 0B0 0A6 ( 🡺 )

Fija los flags correspondientes a todos los campos como FALSE en la lista del LAM9.

Luego actualiza el primer flag de esa lista como FALSE

0A30B0 ROMPTR 0B0 0A3 ( #campo 🡺 )

Fija el flag correspondiente al campo como TRUE en la lista del LAM9.

Luego actualiza el primer flag de esa lista.

0A40B0 ROMPTR 0B0 0A4 ( #campo 🡺 )

Fija el flag correspondiente al campo como FALSE en la lista del LAM9.

Luego actualiza el primer flag de esa lista como FALSE.

0A10B0 ROMPTR 0B0 0A1 ( #campo 🡺 flag )

Retorna el valor **opuesto** del flag correspondiente al campo en la lista del LAM9.

0A20B0 ROMPTR 0B0 0A2 ( 🡺 flag )

Retorna el valor **opuesto** del primer flag de la lista del LAM9.

Hace 9GETLAM CARCOMP NOT

36.8.2 Menú en DoInputForm

**Direcc. Nombre Descripción**

0090B0 ROMPTR 0B0 009 ( 🡺 menú )

Relee el menú del formulario.

Llama a los mensajes 15 y 16.

0040B0 ROMPTR 0B0 004 ( 🡺 {} )

Retorna el menú por defecto (con sus doce elementos).

Antes ejecuta NoExitAction

Para tener el menú por defecto, escribe ' ROMPTR 0B0 004

0050B0 ˜IFMenuRow1 ( 🡺 {} )

Retorna una lista con los tres primeros elementos del menú por defecto (las tres primera teclas).

0060B0 ˜IFMenuRow2 ( 🡺 {} )

Retorna una lista con las seis últimos elementos del menú por defecto (la segunda página del menú).

0070B0 ROMPTR 0B0 007 ( 🡺 {} )

Retorna el menú por defecto (con doce elementos) del formulario cuando hay una línea de edición activa.

Antes ejecuta NoExitAction

Para tener el menú por defecto, escribe ' ROMPTR 0B0 007

0080B0 ROMPTR 0B0 008 ( 🡺 {} )

Retorna una lista con los ocho últimos elementos del menú por defecto del formulario cuando hay una línea de edición activa (CANCL, OK y la segunda página).

36.8.3 Visualización en Pantalla

**Direcc. Nombre Descripción**

00F0B0 ROMPTR 0B0 00F ( 🡺 )

Redibuja el título del formulario (si la línea de edición no es muy grande).

Llama al mensaje 1.

0100B0 ROMPTR 0B0 010 ( 🡺 grob131x7 )

Retorna el título del formulario como un grob.

Llama al mensaje 2.

011B0 ROMPTR 0B0 011 ( 🡺 $ )

Retorna el título del formulario como una cadena.

Llama al mensaje 3.

0B10B0 ROMPTR 0B0 0B1 ( 🡺 $ )

Retorna el título del formulario.

Si el título es un bint, retorna la cadena correspondiente.

No llama a ningún mensaje.

Hace 12GETLAM ?GetMsg\_

0170B0 ROMPTR 0B0 017 ( flag #campo 🡺 )

Muestra en pantalla la ayuda del campo.

Si flag es TRUE, se mostrará justo arriba de los menús.

Si flag es FALSE, se mostrará justo debajo del título.

Llama al mensaje 11.

00D0B0 ROMPTR 0B0 00D ( 🡺 )

Muestra en pantalla la ayuda del campo actual.

Se mostrará justo debajo del título.

Llama al mensaje 11.

0180B0 ROMPTR 0B0 018 ( #campo 🡺 grob )

Retorna la ayuda del campo como un grob.

Llama al mensaje 12.

0190B0 ROMPTR 0B0 019 ( #campo 🡺 $ )

Retorna la ayuda del campo como una cadena.

Llama al mensaje 13.

Si la ayuda es un bint, retorna la cadena correspondiente.

00C0B0 ROMPTR 0B0 00C ( 🡺 )

Redibuja la zona de la pantalla donde están los campos y las etiquetas (no título, ayuda, ni menú).

Llama al mensaje 4.

0990B0 ROMPTR 0B0 099 ( 🡺 grob )

Retorna el grob de etiquetas.

Su tamaño por defecto es 131x39.

Si ya está guardado en LAM10, sólo lo retorna.

Si aún no está guardado en LAM10, lo crea primero, luego lo guarda en LAM10 y luego lo retorna. Puedes evitar que el grob de etiquetas sea guardado usando el mensaje 5.

00E0B0 ROMPTR 0B0 00E ( 🡺 grob T )

Crea y retorna el grob de etiquetas. En este sólo están dibujadas las etiquetas. Su tamaño por defecto es 131x39. Llama a los mensajes 5 y 6.

Este grob corresponde a la parte de la pantalla ubicada debajo del título. Es decir, la posición (0,7) en la pantalla.

0120B0 ROMPTR 0B0 012 ( grob #etiq 🡺 grob' )

Dibuja una sóla etiqueta en el grob de etiquetas.

Llama al mensaje 6.

0BC002 ˆIFEDispField ( #campo 🡺 )

Dibuja un campo en la pantalla.

Llama a los mensajes 7, 9 y 10.

0440B0 ROMPTR 0B0 044 ( #campo 🡺 grobbxh )

Retorna un grob correspondiente al campo.

Llama al mensaje 9.

0450B0 ROMPTR 0B0 045 ( #campo 🡺 $ )

Retorna una cadena correspondiente al campo.

Usa el parámetro Decompile.

Llama al mensaje 10.

No usarlo en campos CHECK.

0750B0 ROMPTR 0B0 075 ( ob Decompile 🡺 $ )

Convierte ob en cadena usando Decompile.

Decompile es un programa o un bint.

0160B0 ROMPTR 0B0 016 ( #campo 🡺 )

Invierte los píxeles en un campo.

36.8.4 Teclas de Menú

**Direcc. Nombre Descripción**

0B5002 ˆDoKeyEdit ( 🡺 (cmd line) )

Para campos texto, combochoose o combofiler.

Edita el valor del campo actual.

Llama a los mensajes 23, 41 y otros.

Equivale a presionar la tecla de menú EDIT.

0B6002 FLASHPTR 002 0B6 ( 🡺 )

Para campos choose, filer combochoose o combofiler.

Muestra las opciones y permite al usuario escoger.

Llama a los mensajes 26 y/o 37.

Equivale a presionar la tecla de menú CHOOS.

0B7002 FLASHPTR 002 0B7 ( 🡺 )

Si el campo actual es un campo check, invierte su valor.

Llama al mensaje 27.

Equivale a presionar la tecla de menú CHK o la tecla +/-.

0B8002 FLASHPTR 002 0B8 ( 🡺 )

Permite escoger entre 2 opciones: "Reset value" (o "Delete value" cuando el valor reset del campo es MINUSONE) y "Reset all". Luego resetea el valor del campo actual o de todos los campos según la opción escogida. Para esto, fija el valor del campo como el valor del parámetro reset del campo.

Llama a los mensajes 31, 32, 33, 34.

Equivale a presionar la tecla de menú RESET.

0B9002 FLASHPTR 002 0B9 ( 🡺 )

Para campos texto, combochoose o combofiler.

Interrumpe temporalmente el formulario de entrada e inicia un entorno con el valor del campo actual en el nivel uno de la pila. Permite al usuario computar un nuevo valor. Si el usuario se retira con CANCL u ON no se actualizará el campo actual.

Llama al mensaje 30.

Equivale a presionar la tecla de menú CALC.

0BA002 FLASHPTR 002 0BA ( 🡺 (cmd line) )

( 🡺 )

Para campos texto, combochoose o combofiler.

Muestra un browser con todos los posibles tipos para el objeto del campo actual.

Si el usuario, presiona la tecla de menú NEW, una línea de comandos es abierta.

Llama a los mensajes 35 y/o 36.

Equivale a presionar la tecla de menú TYPES.

0AF002 ˆDoKeyCancel ( 🡺 )

La acción por defecto es finalizar el formulario dejando FALSE en la pila (fijando el lam 11 como TRUE).

Llama al mensaje 28.

Equivale a presionar la tecla CANCL o la tecla ON.

0B4002 ˆDoKeyOK ( 🡺 )

La acción por defecto es finalizar el formulario dejando los valores de los campos y TRUE en la pila (fijando el lam 11 como TRUE).

Llama al mensaje 29.

Equivale a presionar la tecla OK o la tecla ENTER.

38.8.5 Parámetros de los Campos

**Direcc. Nombre Descripción**

0BC0B0 ROMPTR 0B0 0BC ( #campo 🡺 MessageHandler )

Retorna el message handler de un campo.

0BD0B0 ROMPTR 0B0 0BD ( #campo 🡺 #posX )

Retorna la posición X de un campo.

0BE0B0 ROMPTR 0B0 0BE ( #campo 🡺 #posY )

Retorna la posición Y de un campo.

0BF0B0 ROMPTR 0B0 0BF ( #campo 🡺 #b )

Retorna el ancho de un campo.

0C00B0 ROMPTR 0B0 0C0 ( #campo 🡺 #h )

Retorna la altura de un campo.

0C10B0 ROMPTR 0B0 0C1 ( #campo 🡺 #TipoDeCampo )

Retorna el tipo de campo.

0C20B0 ROMPTR 0B0 0C2 ( #campo 🡺 {#TiposObj}/MINUSONE )

Retorna lista de los tipos permitidos en un campo.

0C30B0 ROMPTR 0B0 0C3 ( #campo 🡺 #/prog )

Retorna el parámetro Decompile de un campo.

0C40B0 ROMPTR 0B0 0C4 ( #campo 🡺 $ )

Retorna la ayuda de un campo.

Si la ayuda es un bint, retorna la cadena correspondiente.

No llama a ningún mensaje.

0C50B0 ROMPTR 0B0 0C5 ( #campo 🡺 {} )

Retorna el parámetro ChooseData de un campo.

0C60B0 ROMPTR 0B0 0C6 ( #campo 🡺 #/prog )

Retorna el parámetro ChooseDecompile de un campo.

0C70B0 ROMPTR 0B0 0C7 ( #campo 🡺 ob )

Retorna el valor Reset de un campo.

0C80B0 ˜gFldVal ( #campo 🡺 ob )

Retorna el valor de un campo.

0CC0B0 ROMPTR 0B0 0CC ( MessageHandler #campo 🡺 )

Cambia el message handler de un campo.

0CD0B0 ROMPTR 0B0 0CD ( #posX #campo 🡺 )

Cambia la posición X de un campo.

0CE0B0 ROMPTR 0B0 0CE ( #posY #campo 🡺 )

Cambia la posición Y de un campo.

0CF0B0 ROMPTR 0B0 0CF ( #b #campo 🡺 )

Cambia el ancho de un campo.

0D00B0 ROMPTR 0B0 0D0 ( #h #campo 🡺 )

Cambia la altura de un campo.

0D10B0 ROMPTR 0B0 0D1 ( #TipoDeCampo #campo 🡺 )

Cambia el tipo de campo.

0D20B0 ROMPTR 0B0 0D2 ( {#TiposObj}/MINUSONE #campo 🡺 )

Cambia lista de los tipos permitidos en un campo.

0D30B0 ROMPTR 0B0 0D3 ( #/prog #campo 🡺 )

Cambia el parámetro Decompile de un campo.

0D40B0 ROMPTR 0B0 0D4 ( #/$ #campo 🡺 )

Cambia la ayuda de un campo.

**Direcc. Nombre Descripción**

0D50B0 ROMPTR 0B0 0D5 ( {} #campo 🡺 )

Cambia el parámetro ChooseData de un campo.

0D60B0 ROMPTR 0B0 0D6 ( #/prog #campo 🡺 )

Cambia el parámetro ChooseDecompile de un campo.

0D70B0 ROMPTR 0B0 0D7 ( ob #campo 🡺 )

Cambia el valor Reset de un campo.

0D80B0 ˜sFldVal ( ob #campo 🡺 )

Cambia el valor de un campo.

0C90B0 ROMPTR 0B0 0C9 ( #campo 🡺 #X #Y )

Retorna las coordenadas de un campo.

0D90B0 ROMPTR 0B0 0D9 ( #X #Y #campo 🡺 )

Cambia las coordenadas de un campo.

0CA0B0 ROMPTR 0B0 0CA ( #campo 🡺 #b #h )

Retorna la base y la altura de un campo.

0DA0B0 ROMPTR 0B0 0DA ( #b #h #campo 🡺 )

Cambia la base y la altura de un campo.

0BB002 ^GetFieldVals ( 🡺 ob1 ob2 ... obn )

Retorna en la pila los valores de todos los campos.

0DB0B0 ROMPTR 0B0 0DB ( ob1 ob2 ... obn 🡺 )

Cambia los valores de todos los campos.

0890B0 ROMPTR 0B0 089 ( #campo 🡺 flag )

Retorna TRUE, si es un campo

TEXTO,

COMBOCHOOSE o

COMBO FILER,

TEXTO ALGEBRAICO,

COMBOCHOOSE ALGEBRAICO o

COMBOFILER ALGEBRAICO.

Es decir, cuando se permite línea de edición.

#TipoCampo: 1, 13, 21, 3, 15, 23.

08A0B0 ROMPTR 0B0 08A ( #campo 🡺 flag )

Retorna TRUE, si es un campo

TEXTO ALGEBRAICO,

COMBOCHOOSE ALGEBRAICO o

COMBOFILER ALGEBRAICO.

Es decir, cuando una línea de edición nueva es algebraica.

#TipoCampo: 3, 15, 23.

08B0B0 ROMPTR 0B0 08B ( #campo 🡺 flag )

Retorna TRUE, si es un campo

CHOOSE,

FILER,

COMBOCHOOSE,

COMBOFILER,

COMBOCHOOSE ALGEBRAICO o

COMBOFILER ALGEBRAICO.

Es decir, cuando se mostrará la tecla de menú CHOOS.

#TipoCampo: 12, 20, 13, 21, 15, 23.

**Direcc. Nombre Descripción**

08C0B0 ROMPTR 0B0 08C ( #campo 🡺 flag )

Retorna TRUE, si es un campo

CHOOSE,

COMBOCHOOSE o

COMBOCHOOSE ALGEBRAICO.

Es decir, cuando será posible escoger de una lista.

#TipoCampo: 12, 13, 21.

08D0B0 ROMPTR 0B0 08D ( #campo 🡺 flag )

Retorna TRUE, si es un campo

FILER,

COMBOFILER o

COMBOFILER ALGEBRAICO.

Es decir, cuando será posible escoger desde el filer.

#TipoCampo: 20, 21, 23.

08E0B0 ROMPTR 0B0 08E ( #campo 🡺 flag )

Retorna TRUE, si es un campo CHECKBOX.

#TipoCampo: 32.

38.8.6 Parámetros de las Etiquetas

**Direcc. Nombre Descripción**

0B20B0 ROMPTR 0B0 0B2 ( #etiq 🡺 # )

Retorna la ubicación de la coordenada Y de la etiqueta dentro del entorno temporal de variables locales de DoInputForm.

Para retornar la ubicación de la coordenada X de la etiqueta:

ROMPTR 0B0 0B2 #1+

Para retornar la ubicación del contenido de la etiqueta:

ROMPTR 0B0 0B2 #2+

0B30B0 ROMPTR 0B0 0B3 ( #etiq 🡺 $ )

Retorna el contenido de la etiqueta.

Si es un bint, retorna la cadena correspondiente.

0B40B0 ROMPTR 0B0 0B4 ( #etiq 🡺 #posX )

Retorna la posición X de una etiqueta.

0B50B0 ROMPTR 0B0 0B5 ( #etiq 🡺 #posY )

Retorna la posición Y de una etiqueta.

0B60B0 ROMPTR 0B0 0B6 ( #etiq 🡺 #posX #posY )

Retorna las coordenadas de una etiqueta.

0B70B0 ROMPTR 0B0 0B7 ( #/$ #etiq 🡺 )

Cambia el contenido de una etiqueta.

0B80B0 ROMPTR 0B0 0B8 ( #posX #etiq 🡺 )

Cambia la posición X de una etiqueta.

0B90B0 ROMPTR 0B0 0B9 ( #posY #etiq 🡺 )

Cambia la posición Y de una etiqueta.

0BA0B0 ROMPTR 0B0 0BA ( #posX #posY #etiq 🡺 )

Cambia las coordenadas de una etiqueta.

38.9 Mensajes para Campos y para Formulario

Primero mostramos un cuadro resumen y luego se explica en detalle a cada uno.

**#MH Lugar Descripción**

1 Form Dibuja el título.

2 Form Retorna el título como grob.

3 Form Retorna el título como cadena.

4 Form Dibuja en la región de campos y etiquetas.

5 Form Retorna el grob de etiquetas.

6 Form Dibuja una etiqueta en el grob de etiquetas.

7 Campo Dibuja el campo en la pantalla.

8 Campo Es llamado cuando se presiona una tecla.

9 Campo Retorna el campo como grob.

10 Campo Retorna el campo como cadena.

11 Campo Dibuja la ayuda del campo actual.

12 Campo Retorna la ayuda del campo actual como grob.

13 Campo Retorna la ayuda del campo actual como cadena.

14 Form Fija el campo que tendrá el enfoque al inicio y al resetear todos.

15 Form Provee el menú.

16 Form Cambia las tres últimas teclas de menú de la primera fila.

17 Form Es llamado al inicio.

18 Campo Es llamado cuando se presiona una tecla capaz de iniciar línea de edición.

19 Campo Es llamado cuando un campo está a punto de perder el enfoque.

20 Campo Es llamado justo cuando un campo ha recibido el enfoque.

21 Campo Es llamado después de que el campo ha perdido el enfoque.

22 Campo Es llamado justo después de que el campo ha recibido el enfoque.

23 Campo Es llamado al presionar la tecla EDIT.

24 Campo Es llamado al presionar la tecla ENTER u OK. Aún existe línea de edición.

25 Campo Es llamado al presionar la tecla ENTER u OK. Ya no existe línea de edición.

26 Campo Es llamado al presionar la tecla CHOOSE.

27 Campo Es llamado al presionar la tecla CHECK.

28 Form Es llamado al presionar la tecla CANCL u ON.

29 Form Es llamado al presionar la tecla ENTER u OK.

30 Campo Es llamado al presionar la tecla CALC.

31 Campo Es llamado al presionar la tecla RESET o DEL. Muestra Browser.

32 Campo Es llamado al presionar la tecla RESET o DEL. Retorna lista.

33 Campo Es llamado cuando se van a resetear un campo.

34 Campo Es llamado cuando se van a resetear todos los campos.

35 Campo Es llamado al presionar la tecla TYPES. Muestra Browser.

36 Campo Es llamado al presionar la tecla TYPES. Retorna lista.

37 Campo Es llamado al presionar la tecla CHOOSE. Retorna objeto escogido.

38 Campo Es llamado al presionar la tecla +/-.

39 Campo Es llamado al presionar una tecla capaz de producir búsqueda alfabética.

40 Campo Es llamado al presionar la tecla CHOOS o +/-.

41 Campo Es llamado al presionar la tecla EDIT o CALC.

42 Campo Es llamado al iniciarse una nueva línea de edición. Provee menú.

43 Campo Es llamado al iniciarse una nueva línea de edición. Provee parte del menú.

44 Campo Es llamado al querer confirmar la línea de edición con ENTER u OK.

45 Campo Es llamado al querer confirmar la línea de edición con ENTER, OK o CALC.

Verifica que el objeto sea de un tipo válido.

46 Campo Es llamado al querer confirmar la línea de edición con ENTER, OK o CALC.

Verifica que el objeto tenga un valor válido.

47 Campo Es llamado inmediatamente después de que un campo ha cambiado su valor.

48 Campo Es llamado al presionar la tecla ABAJO.

49 Campo Es llamado al presionar la tecla ARRIBA.

50 Campo Es llamado al presionar la tecla ShiftDerecho+ABAJO.

51 Campo Es llamado al presionar la tecla ShiftDerecho+ARRIBA.

52 Campo Es llamado al presionar la tecla DERECHA.

53 Campo Es llamado al presionar la tecla IZQUIERDA.

54 Campo Es llamado al presionar la tecla ENTER u OK y se confirmó línea de edición.

55 Campo Es llamado para decidir si el campo actual conserva el enfoque

después de resetear todos los campos

56 Form Es llamado cuando el formulario está a punto de finalizar.

38.9.1 Mensaje 1 (formulario)

Este mensaje es llamado durante el inicio del formulario de entrada para dibujar el título en la pantalla.

La acción por defecto es dibujar un grob de tamaño 131x7 en HARDBUFF, sólo cuando la línea de edición no ocupe toda la pantalla.

Recuerda que para llamar al título del formulario debes usar el comando **12GETLAM**.

Si este mensaje es manejado, ya no se llamarán a los mensajes 2 ni 3. Excepto cuando se usa el comando ROMPTR 0B0 010 (que llama al mensaje 2 y luego al mensaje 3) o el comando ROMPTR 0B0 011 (que llama al mensaje 3).

**Entrada** 1: Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.2 Mensaje 2 (formulario)

Este mensaje es llamado durante el inicio del formulario de entrada para fijar el grob que será mostrado como título del formulario.

La acción por defecto es retornar un grob de tamaño 131x7 que será el título.

Recuerda que para llamar al título del formulario debes usar el comando **12GETLAM**.

Si este mensaje es manejado, ya no será llamado el mensaje 3. Excepto cuando se usa el comando ROMPTR 0B0 011 en el mensaje 2, pues ese comando si llama al mensaje 3.

**Entrada** 1: Nada

**Salida (con efecto)** 2: grob

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.3 Mensaje 3 (formulario)

Este mensaje es llamado durante el inicio del formulario de entrada para fijar la cadena que será mostrada como título del formulario.

La acción por defecto es retornar una cadena a partir del parámetro Título.

Recuerda que para llamar al título del formulario debes usar el comando **12GETLAM**.

Si el mensaje 2 es manejado, este mensaje ya no será llamado. Excepto cuando se usa el comando ROMPTR 0B0 011 en el mensaje 2, pues ese comando si llama al mensaje 3.

**Entrada** 1: Nada

**Salida (con efecto)** 2: $

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.4 Mensaje 4 (formulario)

Este mensaje es llamado para dibujar la pantalla en la región de los campos y las etiquetas.

Las dos acciones por defecto son:

a) Si hay una línea de edición con más de un renglón sólo dibuja la ayuda en la parte superior, debajo del título.

b) De lo contrario, dibuja todos los campos y las etiquetas.

Si este mensaje es manejado, ya no se llamarán a los mensajes 5 ni 6. Excepto cuando se usa el comando ROMPTR 0B0 099 (que llama al mensaje 5 y luego al mensaje 6) o el comando ROMPTR 0B0 00E (que llama al mensaje 5 y luego al mensaje 6) o el comando ROMPTR 0B0 012 (que llama al mensaje 6).

**Entrada** 1: Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación un message handler que imita a la acción por defecto:

' :: BINT4 #=casedrop ( F ) ( -> T // F ) ( Dibuja zona de etiquetas y campos )

:: ( )

EDITPARTS ( # )

BINT1

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

ROMPTR 0B0 099 ( grob ) ( es el grob de etiquetas )

HARDBUFF ( grob HARDBUFF )

BINT0

BINT7 ( grob HARDBUFF #0 #7 )

GROB! ( )

HARDBUFF ( HARDBUFF )

BINT0

BINT46

BINT131 ( HARDBUFF 0 46 131 )

IsBigApple\_

ITE

BINT72

BINT56

GetFontHeight ( grob 0 46 131 # #Hf ) ( 6,7,8 )

#- ( grob 0 46 131 # )

GROB!ZERO ( grob' )

DROP ( )

14GETLAM ( #NC )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

DROPFALSE

;

38.9.5 Mensaje 5 (formulario)

Este mensaje es llamado durante el inicio del formulario de entrada para fijar el grob que será mostrado para las etiquetas.

La acción por defecto es dibujar las etiquetas en un grob de tamaño **131x39**. Ese es el tamaño del grob de etiquetas para la pantalla de la HP49G que tiene una altura de 64 pixeles.

Pero si quieres aprovechar el mayor tamaño de la HP50G (de altura 80 pixeles), puedes usar este mensaje para mostrar más etiquetas y campos, de manera que el grob de etiquetas ahora tendrá un tamaño de **131x55**.

Si en la salida con efecto se pone TRUE en el nivel 2, entonces el grob será colocado en el **lam 10** y este mensaje sólo será llamado al inicio.

Si en la salida con efecto se pone FALSE en el nivel 2, entonces el grob no será colocado en el **lam 10** y este mensaje también será llamado cada vez que se redibuja la pantalla.

Si este mensaje es manejado, ya no será llamado el mensaje 6. Excepto cuando se usa el comando ROMPTR 0B0 012, pues ese comando si llama al mensaje 6.

**Entrada** 1: Nada

**Salida (con efecto)** 3: grob

2: TRUE o FALSE

1: TRUE

**Salida (sin efecto)** 1: FALSE

|  |  |
| --- | --- |
| Grob de etiquetas por defecto  Tiene una altura de 39 pixeles | Con el mensaje 5 se puede aumentar la altura del grob de etiquetas hasta 55 pixeles |
|  |  |

A continuación un message handler que imita a la acción por defecto:

' :: BINT5 #=casedrop ( F ) ( -> GROB' T/F T // F )

:: ( )

BINT39 ( 39 )

BINT131 ( 39 131 )

MAKEGROB ( grob131x39 )

15GETLAM ( grob131x39 #NEtiq )

#1+\_ONE\_DO (DO)

INDEX@ ( grob131x39 #etiq )

ROMPTR 0B0 012 ( grob131x39' ) ( dibuja etiq en grob de etiq )

LOOP

TrueTrue ( grob131x39' T T )

;

DROPFALSE

;

Con el mensaje 5, se pueden dibujar en el grob de etiquetas a lineas, rectángulos, círculos y grobs. A continuación un message handler que dibuja 2 rectángulos en el grob de etiquetas.

' :: BINT5 #=casedrop ( F ) ( -> grob flag T // F ) ( grob de etiquetas )

:: ( )

BINT39 ( 39 )

BINT131 ( 39 131 )

MAKEGROB ( grob131x39 )

15GETLAM ( grob131x39 #NEtiq )

#1+\_ONE\_DO (DO)

INDEX@ ( grob131x39 #etiq )

ROMPTR 0B0 012 ( grob131x39' ) ( dibuja etiq en grob de etiq )

LOOP

0 0 65 29 LBoxB ( grob131x39' )

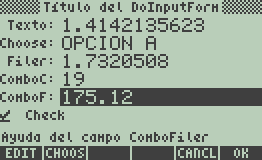
66 0 131 29 LBoxB ( grob131x39' )

TrueTrue ( grob131x39' T T )

;

DROPFALSE

;

A continuación un message handler que permite tener un grob de etiquetas más alto para aprovechar la pantalla de la HP50G. Se ha aumentado la altura del grob de etiquetas desde 39 hasta 55 pixeles.

Aquí se usan los mensajes 4 y 5.

Como en el mensaje 4 se usa el comando ROMPTR 0B0 099, entonces el mensaje 5 será llamado.

Como en el mensaje 5 se usa el comando ROMPTR 0B0 012, entonces el mensaje 6 será llamado.

Características al usar este código:

* Si se emplea un grob de etiquetas (de altura 55)
* Las etiquetas son dibujadas en el grob de etiquetas.
* Las etiquetas pueden dibujarse en toda la pantalla.
* Una etiqueta no puede definirse como un grob.
* Las etiquetas se dibujarán de nuevo, cada vez que se redibuje la pantalla.
* Líneas, rectángulos, círculos y grobs se pueden dibujar sobre el grob de etiquetas.
* El mensaje 6 si será llamado.

\* Message handler del formulario

' ::

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Si se emplea un grob de etiquetas (de altura 55)

\*\*\* Las etiquetas son dibujadas en el grob de etiquetas.

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla.

\*\*\* Una etiqueta no puede definirse como un grob.

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectáng círc grobs se pueden dibujar en el grob de etiquet

\*\*\* El mensaje 6 si será llamado (con el comando ROMPTR 0B0 012)

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

ROMPTR 0B0 099 ( grob ) ( es el grob de etiquetas )

HARDBUFF ( grob HARDBUFF )

BINT0

BINT7 ( grob HARDBUFF 0 7 )

GROB! ( ) ( Reempl grob1 en grob2 )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Crea el grob de etiquetas. Dibuja las etiquetas en ese grob.

\* Retorna el grob de etiquetas.

BINT5 #=casedrop ( F ) ( -> grob flag T // F )

:: ( )

BINT55 ( 55 )

BINT131 ( 55 131 )

MAKEGROB ( grob131x55 ) ( es el grob de etiquetas )

15GETLAM ( grob131x55 #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( grob131x55 #etiq )

ROMPTR 0B0 012 ( grob131x55' ) ( dibuja etiq en grob de etiq )

LOOP

( grob131x55' )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* en el grob de etiquetas. Previamente restar #7 a #Y

FalseTrue ( grob131x55' F T )

;

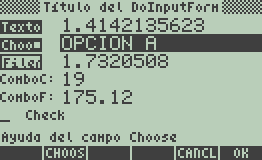
\* Fin del Message handler del formulario:

DROPFALSE

;

Aquí se usa solamente el mensaje 4 para dibujar campos y etiquetas.

Características al usar este código:

* Ya no se emplea un grob de etiquetas
* Los campos y etiquetas son dibujados directamente en la pantalla.
* Las etiquetas pueden dibujarse en toda la pantalla.
* Una etiqueta también puede definirse como un grob.
* Las etiquetas se dibujarán de nuevo, cada vez que se redibuje la pantalla.
* Líneas, rectángulos, círculos y grobs se pueden dibujar directamente en la pantalla.
* Los mensajes 5 y 6 no serán llamados.

\* Message handler del formulario

' ::

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT63 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

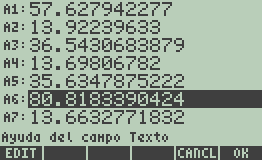
DROPFALSE

;

Aquí se usa solamente el mensaje 4 para dibujar campos y etiquetas y el mensaje 1 para no dibujar el título.

Características al usar este código:

* Ya no se emplea un grob de etiquetas
* Los campos y etiquetas son dibujados directamente en la pantalla.
* Las etiquetas pueden dibujarse en toda la pantalla.
* Una etiqueta también puede definirse como un grob.
* Las etiquetas se dibujarán de nuevo, cada vez que se redibuje la pantalla.
* Líneas, rectángulos, círculos y grobs se pueden dibujar directamente en la pantalla.
* Los mensajes 5 y 6 no serán llamados.
* El titulo del formulario ya no será dibujado, para que la región de etiquetas y campos tenga aun una mayor altura.



\* Message handler del formulario

' ::

\* Dibuja el título, el cual es un grob de tamaño 131x7

\* Sólo cuando la línea de edición no ocupe toda la pantalla

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se dibujará el título en la pantalla

\*\*\* Esto permite que camp y etiq puedan dibujarse en la zona del título

BINT1 #=casedrop ( -> T // F )

TRUE

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT0 ( HARDBUFF 0 0 )

BINT131 ( HARDBUFF 0 0 131 )

BINT63 ( HARDBUFF 0 0 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

38.9.6 Mensaje 6 (formulario)

Este mensaje es llamado durante el inicio del formulario de entrada (para cada etiqueta) para modificar el grob de etiquetas (dibujando en este, la etiqueta correspondiente). También es llamado si se redibuja el grob de etiquetas.

Como el grob de etiquetas se colocará debajo del título, deberás poner el grob en la posición (#x,#y-7) del grob de etiquetas.

Con este mensaje también puedes dibujar la etiqueta usando fuente normal o usando estilos tales como negrita, cursiva, subrayado o inversa; o cualquier grob.

Si el mensaje 5 es manejado, este mensaje ya no será llamado. Excepto cuando se usa el comando ROMPTR 0B0 012 en el mensaje 5, pues ese comando si llama al mensaje 6.

**Entrada** 2: grob

1: #etiqueta

**Salida (con efecto)** 2: grob'

1: TRUE

**Salida (sin efecto)** 3: grob

2: #etiqueta

1: FALSE

A continuación un message handler que imita a la acción por defecto:

' :: BINT6 #=casedrop ( F ) ( GROB #et -> GROB' T // GROB #et F )

:: ( GROB #et )

DUP ( GROB #et #et )

ROMPTR 0B0 0B3 ( GROB #et $etiq )

$>grob ( GROB #et grob )

3PICK ( GROB #et grob GROB )

ROT ( GROB grob GROB #et )

ROMPTR 0B0 0B6 ( GROB grob GROB #X #Y )

#7-\_ ( GROB grob GROB #X #Y-7 )

GROB! ( GROB' )

TRUE ( GROB' T )

;

DROPFALSE

;



Con el siguiente MH, las etiquetas también podrán ser grobs (además de cadenas y bints):

' :: BINT6 #=casedrop ( F ) ( GROB #et -> GROB' T // GROB #et F )

:: ( GROB #et )

DUP ( GROB #et #et )

ROMPTR 0B0 0B3 ( GROB #et $etiq/grob )

DUPTYPEGROB? ( GROB #et $etiq/grob flag )

NOT\_IT

$>grob ( GROB #et grob )

3PICK ( GROB #et grob GROB )

ROT ( GROB grob GROB #et )

ROMPTR 0B0 0B6 ( GROB grob GROB #X #Y )

#7-\_ ( GROB grob GROB #X #Y-7 )

GROB! ( GROB' )

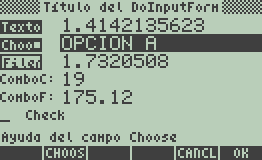
TRUE ( GROB' T )

;

DROPFALSE

;

Aquí se usan los mensajes 4, 5 y 6.

Con el siguiente message handler en el formulario también se pueden dibujar los campos y las etiquetas en un grob de etiquetas ampliado a una altura de 55 píxeles.

Además se pueden usar grobs en las etiquetas.

Como en el mensaje 4 se usa el comando ROMPTR 0B0 099, entonces el mensaje 5 será llamado.

Como en el mensaje 5 se usa el comando ROMPTR 0B0 012, entonces el mensaje 6 será llamado.

' :: BINT4 #=casedrop ( F ) ( -> T // F ) ( Dibuja zona de etiquetas y campos )

:: ( )

EDITPARTS ( # )

BINT1

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

ROMPTR 0B0 099 ( grob ) ( es el grob de etiquetas )

HARDBUFF ( grob HARDBUFF )

BINT0

BINT7 ( grob HARDBUFF #0 #7 )

GROB! ( )

14GETLAM ( #NC )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

BINT5 #=casedrop ( F ) ( -> grob flag T // F ) ( grob de etiquetas )

:: ( )

BINT55 ( 55 )

BINT131 ( 55 131 )

MAKEGROB ( grob131x55 )

15GETLAM ( grob131x55 #NEtiq )

#1+\_ONE\_DO (DO)

INDEX@ ( grob131x55 #etiq )

ROMPTR 0B0 012 ( grob131x55' ) ( dibuja etiq en grob de etiq )

LOOP

( grob131x55' )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* en el grob de etiquetas

TrueTrue ( grob131x55' T T )

;

BINT6 #=casedrop ( F ) ( GROB #et -> GROB' T // GROB #et F )

:: ( GROB #et )

DUP ( GROB #et #et )

ROMPTR 0B0 0B3 ( GROB #et $etiq/grob )

DUPTYPEGROB? ( GROB #et $etiq/grob flag )

NOT\_IT

$>grob ( GROB #et grob )

3PICK ( GROB #et grob GROB )

ROT ( GROB grob GROB #et )

ROMPTR 0B0 0B6 ( GROB grob GROB #X #Y )

#7-\_ ( GROB grob GROB #X #Y-7 )

GROB! ( GROB' )

TRUE ( GROB' T )

;

DROPFALSE

;

38.9.7 Mensaje 7 (campo)

Este mensaje es llamado cuando se va a dibujar un campo y debe dibujar el valor del campo en la pantalla. Si el campo es el campo actual, entonces es dibujado con los pixeles invertidos.

Este mensaje es llamado por el comando ^IFEDispField.

En el **lam 4** se encuentra el índice del campo que se va a dibujar.

Con este mensaje puedes pasar por alto al parámetro Decompile.

Si este mensaje es manejado, ya no se llamarán a los mensajes 9 ni 10. Excepto cuando se usa el comando ROMPTR 0B0 044 (que llama al mensaje 9 y luego al mensaje 10) o el comando ROMPTR 0B0 045 (que llama al mensaje 10).

**Entrada** 1: Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación un message handler que imita a la acción por defecto:

' :: BINT7 #=casedrop ( C ) ( -> T // F )

:: ( )

4GETLAM ( #c )

DUP ( #c #c )

ROMPTR 0B0 044 ( #c grob ) ( retorna el campo como grob )

OVER ( #c grob #c )

5GETLAM ( #c grob #c #ca )

#= ( #c grob flag )

IT

INVGROB

( #c grob )

HARDBUFF ( #c grob GROB )

ROT ( grob GROB #c )

ROMPTR 0B0 0C9 ( grob GROB #X #Y )

GROB! ( )

TRUE ( T )

;

DROPFALSE

;

38.9.8 Mensaje 8 (campo)

Este mensaje es llamado cuando se presiona alguna tecla.

Si uno escoge la salida con efecto, entonces luego se ejecutará el programa colocado en la pila.

Si uno escoge la salida sin efecto, entonces luego se ejecutará el código normal correspondiente a la tecla presionada.

**Entrada** 2: #CódigoDeTecla

1: #PlanoDeTecla

**Salida (con efecto)** 3: ::Programa

2: TRUE

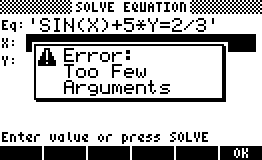
1: TRUE

**Salida (sin efecto)** 3: #CódigoDeTecla

2: #PlanoDeTecla

1: FALSE

Cuando se usa un formulario de entrada hecho con DoInputForm y no hay activa una línea de edición en un campo TEXTO, COMBOCHOOSE o COMBOFILER, la tecla STO realiza operaciones con la pila que a veces no son deseadas. Para quitar esta asignación de tecla puedes usar el siguiente message handler en cada campo TEXTO, COMBOCHOOSE o COMBOFILER.



' :: BINT8 #=casedrop ( C ) ( #ct #p -> accion T T // #ct #p F )

:: ( #ct #p )

EditLExists? ( #ct #p flag )

caseFALSE ( SALE CON: #ct #p flag )

( #ct #p )

OVER BINT12 #= ( #ct #p flag )

OVER #1= ( #ct #p flag flag' )

ANDcase

:: ( #ct #p )

2DROP ( )

'DoBadKey ( DoBadKey )

TrueTrue ( DoBadKey T T )

;

( #ct #p )

FALSE ( #ct #p F )

;

DROPFALSE

;

38.9.9 Mensaje 9 (campo)

Este mensaje es llamado cuando se va a dibujar un campo y debe retornar el valor del campo convertido en grob.

En el nivel 1 de la pila se encuentra el índice del campo que se va a dibujar.

Con este mensaje puedes pasar por alto al parámetro Decompile.

Con este mensaje puedes retornar un campo en minifuente, estilos negrita, cursiva, subrayado, inversa, etc.

Si este mensaje es manejado, ya no será llamado el mensaje 10. Excepto cuando se usa el comando ROMPTR 0B0 045 en el mensaje 9, pues ese comando si llama al mensaje 10.

**Entrada** 1: #Campo

**Salida (con efecto)** 3: #Campo

2: grob

1: TRUE

**Salida (sin efecto)** 2: #Campo

1: FALSE

Aquí presentamos una plantilla que usa el mensaje 9 para retornar un grob correspondiente a un campo (que no sea CHECK) que respete su base y su altura y coloque el contenido del campo en un lugar correcto del grob. En este ejemplo se muestra el campo con minifuente.

En este código se ha ignorado al parámetro Decompile.

Además el mensaje 10 no será llamado.

' :: BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h )

SWAP ( #c #h #b )

MAKEGROB ( #c grobbxh )

OVER ( #c grobbxh #c )

ROMPTR gFldVal ( #c grobbxh valor )

\* Lo siguiente debe convertir a grob: ( valor -> grob )

\* Puedes cambiar por otro código

DO>STR

$>grob

\* -----------------------------------------------------

( #c grobbxh grob )

OVER ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' )

TRUE ( #c grobbxh' T )

;

DROPFALSE

;

38.9.10 Mensaje 10 (campo)

Este mensaje es llamado cuando se va a dibujar un campo y debe retornar el valor del campo convertido en cadena.

Este mensaje no es llamado para campos CHECK.

En el nivel 1 de la pila se encuentra el índice del campo que se va a dibujar.

Con este mensaje puedes pasar por alto al parámetro Decompile.

Con este mensaje puedes retornar un campo negrita, cursiva, subrayado o inversa.

Si el mensaje 9 es manejado, este mensaje ya no será llamado. Excepto cuando se usa el comando ROMPTR 0B0 045 en el mensaje 9, pues ese comando si llama al mensaje 10.

**Entrada** 1: #Campo

**Salida (con efecto)** 3: #Campo

2: $

1: TRUE

**Salida (sin efecto)** 2: #Campo

1: FALSE

A continuación un message handler que imita a la acción por defecto:

' :: BINT10 #=casedrop ( C ) ( #c -> #c $ T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR gFldVal ( #c valor )

OVER ( #c valor #c )

ROMPTR 0B0 0C3 ( #c valor Decompile )

ROMPTR 0B0 075 ( #c $ )

TRUE ( #c $ T )

;

DROPFALSE

;

38.9.11 Mensaje 11 (campo)

Este mensaje es llamado cuando se va a mostrar la ayuda del campo actual y debe dibujar la ayuda del campo actual en la pantalla.

En el nivel 1 de la pila está disponible un flag.

Es TRUE cuando la ayuda debe dibujarse en la parte inferior.

Es FALSE cuando la ayuda debe dibujarse en la parte superior debido a la existencia de una línea de edición con más de un renglón.

Si este mensaje es manejado, ya no se llamarán a los mensajes 12 ni 13. Excepto cuando se usa el comando ROMPTR 0B0 018 (que llama al mensaje 12 y luego 13) o el comando ROMPTR 0B0 019 (que llama al mensaje 13).

**Entrada** 1: flag

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 2: flag

1: FALSE

38.9.12 Mensaje 12 (campo)

Este mensaje es llamado cuando se va a mostrar la ayuda del campo actual y debe retornar la ayuda del campo actual como un grob.

Si este mensaje es llamado, ya no será llamado el mensaje 13. Excepto cuando se usa el comando ROMPTR 0B0 019 en el mensaje 12, pues ese comando si llama al mensaje 13.

**Entrada** Nada

**Salida (con efecto)** 2: grob

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.13 Mensaje 13 (campo)

Este mensaje es llamado cuando se va a mostrar la ayuda del campo actual y debe retornar la ayuda del campo actual como una cadena.

Si el mensaje 12 es manejado, este mensaje ya no será llamado. Excepto cuando se usa el comando ROMPTR 0B0 019 en el mensaje 12, pues ese comando si llama al mensaje 13.

**Entrada** Nada

**Salida (con efecto)** 2: $

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.14 Mensaje 14 (formulario)

Este mensaje es llamado durante el inicio del formulario de entrada para fijar el número del campo que primero tendrá el enfoque.

También es llamado cuando se resetean todos los campos, y se decide que el campo actual ya no debe seguir con el enfoque; para fijar el índice del nuevo campo.

El valor por defecto para el campo inicial es #1.

**Entrada** Nada

**Salida (con efecto)** 2: #campo

1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación un message handler que le da el enfoque inicial al campo 2:

' :: BINT14 #=casedrop ( F ) ( -> #c T // F ) ( Fija camp con enfoque inic )

:: ( )

BINT2 ( #c )

TRUE ( #c T )

;

DROPFALSE

;

A continuación un message handler que le da el enfoque inicial al último campo:

' :: BINT14 #=casedrop ( F ) ( -> #c T // F ) ( Fija camp con enfoque inic )

:: ( )

14GETLAM ( #c ) ( 14LAM contiene al nº de campos )

TRUE ( #c T )

;

DROPFALSE

;

A continuación un message handler que le da el enfoque inicial al primer campo que encuentre que contenga a un número real mayor a 75:

' :: BINT14 #=casedrop ( F ) ( -> #c T // F ) ( Fija camp con enfoque inic )

:: ( )

BINT1 ( #1 )

14GETLAM ( #1 #nc ) ( 14LAM contiene al nº de campos )

#1+\_ONE\_DO (DO)

INDEX@ ( #1 #c )

ROMPTR gFldVal ( #1 Valor )

DUPTYPEREAL? ( #1 Valor flag )

ITE

:: % 75. %> ( #1 flag )

NOT?SEMI

( #1 )

ExitAtLOOP ( #1 )

DROP ( )

INDEX@ ( #c )

;

DROP

LOOP

TRUE ( #c T )

;

DROPFALSE

;

38.9.15 Mensaje 15 (formulario)

Este mensaje es llamado antes del inicio del formulario de entrada, y puede ser usado para proveer un menú. El menu debe estar en el formato descrito en la sección 39.1.

El menú por defecto es una lista con 12 elementos.

**Entrada** Nada

**Salida (con efecto)** 2: { } o programa (menú)

1: TRUE

**Salida (sin efecto)** 1: FALSE

Si el message handler del formulario de entrada es el siguiente, entonces estará disponible una tecla de menú para ver un texto de ayuda.

' :: BINT15 #=casedrop ( F ) ( -> menú T // F )

:: ( )

{ "HELP" :: TakeOver FALSE "Texto de ayuda" ViewStrObject DROP ; }

( {} )

BINT4 ( {} #4 ) ( 4 para la cuarta tecla )

ROMPTR 0B0 004 ( {} #4 {}menú ) ( menú por defecto )

PUTLIST ( {}menú )

' NoExitAction ( {}menú NoExitAction )

SWAP ( NoExitAction {}menú )

BINT2 ( NoExitAction {}menú #2 )

::N ( ::menú )

TRUE ( ::menú' T )

;

DROPFALSE

;

38.9.16 Mensaje 16 (formulario)

Este mensaje es llamado antes del inicio del formulario de entrada, y puede ser usado para cambiar las 3 últimas teclas de la primera fila del menú. Si este mensaje es manejado, se debe de retornar una lista con tres sublistas, cada una de estas será la definición de la tecla.

Si el mensaje 15 es manejado, este mensaje ya no será llamado.

**Entrada** Nada

**Salida (con efecto)** 3: { } (lista con tres elementos)

2: TRUE

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.17 Mensaje 17 (formulario)

Este mensaje es llamado al inicio del formulario de entrada. Cuando ya estan fijados los valores de los lams, pero antes de que se inicie el POL.

Puedes hacer cualquier cosa aquí. No hay diferencia si retornas TRUE o FALSE.

**Entrada** Nada

**Salida** 1: TRUE o FALSE

38.9.18 Mensaje 18 (campo)

Este mensaje es llamado cuando se presiona una tecla capaz de iniciar una línea de edición en un campo TEXTO, COMBOCHOOSE o COMBOFILER.

Generalmente este mensaje es llamado en campos algebraicos (TEXTO ALGEBRAICO, COMBOCHOOSE ALGEBRAICO o COMBOFILER ALGEBRAICO).

Su acción por defecto es agregarle comillas simples a una línea de edición nueva e iniciar el modo de entrada algebraica (esto se realizará cuando la línea de edición no se inicie con algo que contenga a [, «, ", #, : o {)

**Entrada** 2: #CódigoDeTecla

1: #PlanoDeTecla

**Salida (con efecto)** 3: ::Programa

2: TRUE

1: TRUE

**Salida (sin efecto)** 3: #CódigoDeTecla

2: #PlanoDeTecla

1: FALSE

El siguiente message handler imita a la acción por defecto:

' :: BINT18 #=casedrop ( C ) ( #ct #p -> acción T T // #ct #p F )

:: ( #ct #p )

EditLExists? ( #ct #p flag )

caseFALSE ( SALE CON: #ct #p F )

( #ct #p )

5GETLAM ( #ct #p #campo )

ROMPTR 0B0 08A ( #ct #p flag ) ( TRUE si es algebraico )

NOTcaseFALSE ( SALE CON: #ct #p F )

( #ct #p )

Key>U/SKeyOb ( ob )

6PUTLAM ( )

' ROMPTR 0B0 04C ( romptr )

TrueTrue ( romptr T T )

;

DROPFALSE

;

38.9.19 Mensaje 19 (campo)

Este mensaje es llamado cuando un campo está a punto de perder el enfoque.

Puedes hacer cualquier cosa aquí. No hay diferencia si retornas TRUE o FALSE.

El **lam 5** todavía contiene al índice del campo que está a punto de perder el enfoque.

**Entrada** Nada

**Salida** 1: TRUE o FALSE

38.9.20 Mensaje 20 (campo)

Este mensaje es llamado justo cuando un campo ha recibido el enfoque.

Puedes hacer cualquier cosa aquí. No hay diferencia si retornas TRUE o FALSE.

El **lam 5** ya contiene al índice del campo que ha recibido reciéntemente el enfoque.

**Entrada** Nada

**Salida** 1: TRUE o FALSE

38.9.21 Mensaje 21 (campo)

Este mensaje es llamado después de que el campo ha perdido el enfoque y su misión es invertir los píxeles del campo.

Cuando este mensaje es llamado, el índice del campo lo puedes encontrar en 4LAM (ya no está en 5LAM).

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

El siguiente message handler imita a la acción por defecto:

' :: BINT21 #=casedrop ( C ) ( -> T // F )

:: ( )

4GETLAM ( #c )

ROMPTR 0B0 016 ( ) ( invierte píxeles de campo )

TRUE ( T )

;

DROPFALSE

;

38.9.22 Mensaje 22 (campo)

Este mensaje es llamado justo después de que el campo ha recibido el enfoque y su misión es invertir los píxeles del campo.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

El siguiente message handler imita a la acción por defecto:

' :: BINT22 #=casedrop ( C ) ( -> T // F )

:: ( )

5GETLAM ( #c )

ROMPTR 0B0 016 ( ) ( invierte píxeles de campo )

TRUE ( T )

;

DROPFALSE

;

38.9.23 Mensaje 23 (campo)

Este mensaje es llamado al presionar la tecla EDIT y su misión es editar el valor del campo.

La acción por defecto es iniciar la línea de edición con el valor actual del campo.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.24 Mensaje 24 (campo)

Este mensaje es llamado cuando se quiere confirmar la línea de edición con ENTER u OK. Cuando es llamado, todavía existe una línea de edición, la cual puedes llamar con el comando RCL\_CMD o con RCL\_CMD2.

Puedes hacer cualquier cosa aquí. No hay diferencia si retornas TRUE o FALSE.

**Entrada** Nada

**Salida** 1: TRUE o FALSE

38.9.25 Mensaje 25 (campo)

Este mensaje es llamado cuando se quiere confirmar la línea de edición con ENTER u OK. Cuando es llamado, ya no existe línea de edición y la cadena de la línea de edición está descompilada en la pila (si son varios objetos, se encuentran agrupados en una lista). Este mensaje es llamado antes de verificar si el objeto que se quiere asiganar al campo es de un tipo correcto y de un valor válido (mensajes 45 y 46).

Puedes hacer cualquier cosa aquí. No hay diferencia si retornas TRUE o FALSE.

**Entrada** Nada

**Salida** 1: TRUE o FALSE

38.9.26 Mensaje 26 (campo)

Este mensaje es llamado cuando es presionada la tecla CHOOSE en un campo CHOOSE, FILER, COMBOCHOOSE o COMBOFILER.

Su misión es presentar las opciones al usuario y si una opción es escogida, guardar el nuevo valor del campo y mostrarlo en la pantalla.

Si manejas este mensaje, podrías pasar por alto al mensaje 37.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación un message handler que imita a la acción por defecto.

' :: BINT26 #=casedrop ( C ) ( -> T // F )

:: ( )

4GETLAM ( #c )

ROMPTR 0B0 030 ( ob T // F // tag )

' TRUE

EQUAL ( ob T // F )

NOTcaseTRUE ( SALE CON: T )

( ob )

DUPTYPEMATRIX?\_ ( ob flag )

IT

:: ( MATRIX )

"¿Convertir matriz a arreglo?" ( MATRIX $ )

AskQuestion ( MATRIX flag )

case

\* Intenta convertir a arreglo llamando a ^MATRIX2ARRAY

FLASHPTR 006 597 ( MATRIX/ARRAY )

;

( ob' )

4GETLAM ( ob' #c )

DUPUNROT ( #c ob' #c )

ROMPTR sFldVal ( #c ) ( guarda ob' en el campo #c )

DUP ( #c #c )

ROMPTR 0B0 041 ( #c ) ( ejecuta el mensaje 47 )

FLASHPTR IFEDispField ( ) ( dibuja el campo en la pantalla )

TRUE ( T )

;

DROPFALSE

;

38.9.27 Mensaje 27 (campo)

Este mensaje es llamado cuando es presionada la tecla CHECK o la tecla +/- en un campo check. Su misión es cambiar el valor del campo check.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación un message handler que imita a la acción por defecto:

' :: BINT27 #=casedrop ( C ) ( -> T // F )

:: ( )

4GETLAM ( #c )

DUP ( #c #c )

ROMPTR gFldVal ( #c flag )

NOT ( #c flag' )

OVER ( #c flag' #c )

ROMPTR sFldVal ( #c )

ROMPTR 0B0 041 ( ) ( ejecuta el mensaje 47 )

TRUE ( T )

;

DROPFALSE

;

38.9.28 Mensaje 28 (formulario)

Este mensaje es llamado cuando es presionada la tecla CANCL o la tecla ON.

El programador puede evitar que el formulario sea finalizado si hay una entrada inválida, por ejemplo.

**Entrada** Nada

**Salida (con efecto)** 2: TRUE o FALSE (TRUE finalizará el POL, FALSE no)

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.29 Mensaje 29 (formulario)

Este mensaje es llamado cuando es presionada la tecla ENTER o la tecla OK.

El programador puede evitar que el formulario sea finalizado si hay una entrada inválida, por ejemplo.

**Entrada** Nada

**Salida (con efecto)** 2: TRUE o FALSE (TRUE finalizará el POL, FALSE no)

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.30 Mensaje 30 (campo)

Este mensaje es llamado cuando es presionada la tecla de menú CALC en campos TEXTO, COMBOCHOOSE o COMBOFILER.

La acción por defecto es abrir un entorno donde podremos manipular la pila y colocar un objeto en el nivel 1 para que sea el nuevo valor del campo. Si este objeto es de un tipo válido y de un valor válido, entonces es guardado como el nuevo valor del campo. De lo contario, muestra una alerta en la pantalla.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.31 Mensaje 31 (campo)

Este mensaje es llamado cuando se presiona la tecla de menú RESET o la tecla DEL (ShiftIzq+DROP).

Su acción por defecto es mostrar un browser con dos opciones "Reset value" (o "Delete value" cuando el valor reset del campo es MINUSONE) y "Reset value". Luego procede de acuerdo a la opción escogida.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación un message handler que imita a la acción por defecto:

' :: BINT31 #=casedrop ( C ) ( -> T // F )

:: ( )

NULL$ ( $ #c )

4GETLAM ( $ #c )

ROMPTR 0B0 01B ( $ {{}} )

FLASHPTR RunChooseSimple ( ob2 T // F ) ( 2º elemento del ítem )

ITE

EVAL

SetDAsNoCh

( )

TRUE ( T )

;

DROPFALSE

;

38.9.32 Mensaje 32 (campo)

Este mensaje es llamado cuando se presiona la tecla de menú RESET o la tecla DEL (ShiftIzq+DROP).

Su acción por defecto es retornar una lista de ítems que serán mostrados al usuario con el browser.

Este mensaje debe retornar una lista de listas.

Cada una de las sublistas debe tener la siguiente forma: { #/$ ob }

#/$ Es lo que se mostrará en el browser

ob Es el objeto que será evaluado, si uno escoge esa opción.

**Entrada** Nada

**Salida (con efecto)** 1: {{}}

1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación un message handler que imita a la acción por defecto:

' :: BINT32 #=casedrop ( C ) ( -> {{}} T // F )

:: ( )

4GETLAM ( #c )

ROMPTR 0B0 0C7 ( reset )

MINUSONE ( reset MINUSONE )

EQUAL ( flag )

ITE

{ # B904 ROMPTR 0B0 01C } ( "Delete value" )

{ # B903 ROMPTR 0B0 01C } ( "Reset value" )

{ # B905 ROMPTR 0B0 01D } ( "Reset all" )

TWO{}N ( {{# romptr},{# romptr}} )

TRUE ( {{# romptr},{# romptr}} T )

;

DROPFALSE

;

38.9.33 Mensaje 33 (campo)

Este mensaje es llamado cuando a un campo se le va a asignar su valor reset, debido a la elección de la opción "Reset value" o "Delete value" en un browser.

También es llamado una vez por cada campo cuando se escoge la opción "Reset All".

También es llamado cuando se confirma una línea de edición vacía con ENTER u ON.

Su misión es colocar el valor reset como nuevo valor del campo.

El índice del campo actual está en **5LAM**.

El índice del campo que se está reseteando está en **4LAM**.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación un message handler que imita a la acción por defecto:

' :: BINT33 #=casedrop ( C ) ( -> T // F )

:: ( )

4GETLAM ( #c )

DUP ( #c #c )

DUP ( #c #c #c )

ROMPTR 0B0 0C7 ( #c #c reset )

SWAP ( #c reset #c )

ROMPTR sFldVal ( #c ) ( guarda valor en el campo )

ROMPTR 0B0 041 ( #c ) ( ejecuta el mensaje 47 )

TRUE ( T )

;

DROPFALSE

;

38.9.34 Mensaje 34 (campo)

Este mensaje es llamado cuando todos los campos se van a resetear, debido a la elección de la opción "Reset All".

Su acción por defecto es resetear cada uno de los campos, y luego preguntar al mensaje 55 si el campo actual merece seguir con el enfoque. Si no lo merece, pregunta al mensaje 14, cual será el nuevo campo.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.35 Mensaje 35 (campo)

Este mensaje es llamado cuando es presionada la tecla de menú TYPES (F3) en campos TEXTO, COMBOCHOOSE o COMBOFILER.

Este mensaje también es llamado cuando después de ingresar un valor de tipo incorrecto en un campo, aparece una alerta y el usuario presiona la tecla de menú TYPES (F1).

La acción por defecto es mostrar en un browser los tipos permitidos para este campo y abrir una línea de edición nueva si uno escoge una de estas opciones con la tecla NEW.

Si manejas este mensaje puedes pasar por alto al mensaje 36.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.36 Mensaje 36 (campo)

Este mensaje es llamado cuando es presionada la tecla de menú TYPES (F3) en campos TEXTO, COMBOCHOOSE o COMBOFILER.

Este mensaje también es llamado cuando después de ingresar un valor de tipo incorrecto en un campo, aparece una alerta y el usuario presiona la tecla de menú TYPES (F1).

Este mensaje debe retornar una lista de listas.

Cada una de las sublistas debe tener la forma: { #/$ :: #pos FlagALG $EditLine ; }

#/$ Es lo que se mostrará en el browser

#pos Es la posición del cursor en la nueva línea de edición.

FlagALG Si es TRUE, se activará el modo algebraico.

$EditLine Es la cadena de la nueva línea de edición que se abrirá.

Con este mensaje puedes tener más TiposPermitidos en el formulario DoInputForm (colocar los BINTS correspondientes en la lista TiposPermitidos del campo.

**Entrada** Nada

**Salida (con efecto)** 2: {{}}

1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación un message handler que imita a la acción por defecto:

' :: BINT36 #=casedrop ( C ) ( -> {{}} T // F )

:: ( )

4GETLAM ( #c )

ROMPTR 0B0 0C2 ( {#TiposObj}/MINUSONE )

DUP MINUSONE EQUAL ( {#TiposObj}/MINUSONE flag )

casedrop

:: { { # B908 :: BINT1 FALSE NULL$ ; } } TRUE ;

( {#TiposObj} )

DUPLENCOMP ( {#tipos} #n )

2NULLLAM{}\_ ( {#tipos} #n {NULLLAM NULLLAM} )

BIND ( )

1GETLAM ( #n )

#1+\_ONE\_DO

2GETLAM ( {#tipos} )

INDEX@ ( {#tipos} #i )

NTHCOMPDROP ( #tipo )

{ BINT0

{ # B909 :: BINT1 FALSE NULL$ ; }

BINT1

{ # B90A :: BINT2 FALSE "()" ; }

BINT2

{ # B90B :: BINT2 FALSE "\"\"" ; }

BINT3

{ # B90C :: BINT2 FALSE "[]" ; }

BINT4

{ # B90D :: BINT3 FALSE "[()]" ; }

BINT5

{ # B90E :: BINT2 FALSE "{}" ; }

BINT6

{ # B90F :: BINT1 FALSE NULL$ ; }

BINT8

{ # B910 :: BINT3 FALSE "" "" ; }

BINT9

{ # B911 :: BINT2 TRUE "''" ; }

BINT10

{ # B912 :: BINT2 FALSE "#" ; }

BINT13

{ # B913 :: BINT1 FALSE "\_" ; } }

EQLookup ( ... {}nextob T )

DROP ( ... {}nextob )

LOOP

1GETABND ( ... #n )

{}N ( {{}} )

TRUE ( {{}} T )

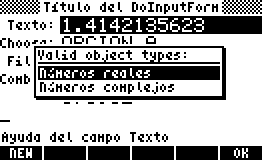
;

DROPFALSE

;

A continuación un message handler que podéis usar cuando los tipos permitidos del campo sean números reales y complejos.

Es decir, cuando el parámetro TiposPermitidos sea { BINT0 BINT1 }



' :: BINT36 #=casedrop ( C ) ( -> {{}} T // F )

:: ( )

{ { "Números reales" :: BINT1 FALSE NULL$ ; }

{ "Números complejos" :: BINT2 FALSE "()" ; }

} ( {{}} )

TRUE ( {{}} T )

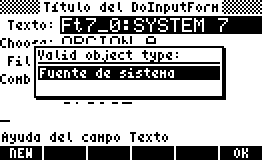
;

DROPFALSE

;

A continuación un message handler que te permite poner objetos de otros tipos en un campo (tipos diferentes a los de la tabla de la sección 38.2).

Por ejemplo, si quieres que en un campo texto, el unico objeto permitido sea una fuente normal, entonces el parámetro TiposPermitidos debe ser { BINT30 } y el message handler del campo debe ser:



' :: BINT36 #=casedrop ( C ) ( -> {{}} T // F )

:: ( )

{ { "Fuente de sistema" :: BINT1 FALSE NULL$ ; } } ( {{}} )

TRUE ( {{}} T )

;

DROPFALSE

;

38.9.37 Mensaje 37 (campo)

Este mensaje es llamado cuando es presionada la tecla CHOOSE en un campo CHOOSE, FILER, COMBOCHOOSE o COMBOFILER. Su misión es presentar las opciones al usuario y si una opción es escogida, retornarla en la pila seguida de TRUE y TRUE. Si el usuario no escoge ninguna opción, en la pila debe retornarse FALSE y TRUE.

**Entrada** Nada

**Salida (con efecto)** 3: objeto

2: TRUE

1: TRUE

**Salida (con efecto)** 2: FALSE

1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación un message handler que imita a la acción por defecto y que puede servir como plantilla.

' :: BINT37 #=casedrop ( C ) ( -> ob T T // F T // F )

:: ( )

4GETLAM ( #c )

DUP ( #c #c )

ROMPTR 0B0 08D ( #c flag ) ( TRUE si es filer o combofiler )

case

:: ( #c )

FALSE ( #c F )

SWAP ( F #c )

\* Retorna los parámetros de ^BrowseMem.1 a partir de ChooseData

\* Puedes cambiar este comando por otro código

ROMPTR 0B0 078 ( F ob2 flagCheck {}tipos flagObjNomb )

SWAP

TOTEMPOB

SWAP ( F ob2 flagCheck {}tipos flagObjNomb )

FLASHPTR BrowseMem.1 ( ob T // F // tag )

SetDAsNoCh ( ob T // F // tag )

TRUE ( ob T T // F T // tag T )

;

( #c )

\* Retorna lista de ítems e índice inicial para browser 48 a partir de

\* ChooseData. Puedes cambiar este comando por otro código

ROMPTR 0B0 02B ( {}items #índice )

OVER ( {}items #índice {}items )

NULL{} ( {}items #índice {}items {} )

EQUAL ( {}items #índice flag )

case2drop

:: ( )

DoBadKey ( )

FalseTrue ( F T )

;

( {}items #índice )

DUP#0= ( {}items #índice flag )

IT

DROPONE

( {}items #índice' )

4GETLAM ( {}items #índice' #c )

ROMPTR 0B0 0BC ( {}items #índice' MH ) ( MH del campo )

NULL$ ( {}items #índice' MH $ ) ( Título del Browser 48 )

4GETLAM ( {}items #índice' MH $ #c )

ROMPTR 0B0 0C6 ( {}items #índice' MH $ ChooseDecomp ) ( Converter )

5ROLL

5ROLL ( MH $ ChooseDecomp {}items #índice' )

ROMPTR Choose ( ob T // F )

SetDAsNoCh ( ob T // F )

TRUE ( ob T T // F T )

;

DROPFALSE

;

A continuación un message handler plantilla que puede usarse sólo en campos filer y combofiler.

' :: BINT37 #=casedrop ( C ) ( -> ob T T // F T // F )

:: ( )

FALSE ( F )

4GETLAM ( F #c )

\* Retorna los parámetros de ^BrowseMem.1 a partir de ChooseData

\* Puedes cambiar este comando por otro código

ROMPTR 0B0 078 ( F ob2 flagCheck {}tipos flagObjNomb )

SWAP

TOTEMPOB

SWAP ( F ob2 flagCheck {}tipos flagObjNomb )

FLASHPTR BrowseMem.1 ( ob T // F // tag )

SetDAsNoCh ( ob T // F // tag )

TRUE ( ob T T // F T // tag T )

;

DROPFALSE

;

A continuación un message handler plantilla que puede usarse sólo en campos choose y combochoose.

' :: BINT37 #=casedrop ( C ) ( -> ob T T // F T // F )

:: ( )

4GETLAM ( #c )

\* Retorna lista de ítems e índice inicial para browser 48 a partir de

\* ChooseData. Puedes cambiar este comando por otro código

ROMPTR 0B0 02B ( {}items #índice )

OVER ( {}items #índice {}items )

NULL{} ( {}items #índice {}items {} )

EQUAL ( {}items #índice flag )

case2drop

:: ( )

DoBadKey ( )

FalseTrue ( F T )

;

( {}items #índice )

DUP#0= ( {}items #índice flag )

IT

DROPONE

( {}items #índice' )

4GETLAM ( {}items #índice' #c )

ROMPTR 0B0 0BC ( {}items #índice' MH ) ( MH del campo )

NULL$ ( {}items #índice' MH $ ) ( Título del Browser 48 )

4GETLAM ( {}items #índice' MH $ #c )

ROMPTR 0B0 0C6 ( {}items #índice' MH $ ChooseDecomp ) ( Converter )

5ROLL

5ROLL ( MH $ ChooseDecomp {}items #índice' )

ROMPTR Choose ( ob T // F )

SetDAsNoCh ( ob T // F )

TRUE ( ob T T // F T )

;

DROPFALSE

;

38.9.38 Mensaje 38 (campo)

Este mensaje es llamado cuando es presionada la tecla la tecla +/- en un campo CHOOSE o COMBOCHOOSE. Su misión es cambiar el valor del campo y mostrar el nuevo valor del campo en la pantalla.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación un message handler que imita a la acción por defecto y se puede usar como plantilla.

' :: BINT38 #=casedrop ( C ) ( -> T // F )

:: ( )

SetDAsNoCh ( )

4GETLAM ( #c )

DUP ( #c #c )

ROMPTR 0B0 02B ( #c {}items #índice )

OVER ( #c {}items #índice {}ITEMS )

NULL{} ( #c {}items #índice {}ITEMS {} )

EQUAL ( #c {}items #índice flag )

case

3DROPTRUE\_ ( sale con TRUE )

( #c {}items #índice flag )

#1+ ( #c {}items #índice+1 )

2DUP ( #c {}items #índice+1 {}items #índice+1 )

SWAP ( #c {}items #índice+1 #índice+1 {}items )

LENCOMP ( #c {}items #índice+1 #índice+1 #n )

#> ( #c {}items #índice+1 flag )

IT

DROPONE

( #c {}items #índice+1/#1 )

NTHCOMPDROP ( #c obi+1/ob1 )

OVER ( #c obi+1/ob1 #c )

ROMPTR sFldVal ( #c )

DUP ( #c #c )

ROMPTR 0B0 041 ( #c ) ( ejecuta el mensaje 47 )

FLASHPTR IFEDispField ( )

TRUE ( T )

;

DROPFALSE

;

38.9.39 Mensaje 39 (campo)

Este mensaje es llamado cuando son presionadas algunas teclas (capaces de producir búsqueda alfabética) sólo en un campo choose.

Su misión es buscar un elemento de ChooseData cuyo primer carácter coincida con el carácter correspondiente a la tecla que se ha presionado.

En caso de encontrar un elemento, se deberá fijar este elemento como el nuevo valor del campo, y mostrarlo en la pantalla.

**Entrada** 1: chr

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 2: chr

1: FALSE

38.9.40 Mensaje 40 (campo)

Este mensaje es llamado cuando es presionada la tecla CHOOSE o la tecla +/- en un campo CHOOSE o COMBOCHOOSE. Su misión es retornar en la pila la lista de ítems y el índice inicial que usará el browser 48 para presentar las opciones al usuario.

Las acciones por defecto son:

a) Retornar el parámetro ChooseData del campo y el índice será la posición en la lista del objeto del campo.

b) Si el objeto del campo no se encuentra en la lista, entonces este objeto es agregado al inicio de la lista y debes retornar como índice a cero.

**Entrada** Nada

**Salida (con efecto)** 3: {}Items

2: #índice

1: TRUE

**Salida (sin efecto)** 1: FALSE

A continuación un message handler que imita a la acción por defecto y se puede usar como plantilla.

' :: BINT40 #=casedrop ( C ) ( -> {}items #índice T // F )

:: ( )

4GETLAM ( #c )

ROMPTR 0B0 0C5 ( ChooseData )

4GETLAM ( ChooseData #c )

ROMPTR gFldVal ( ChooseData valor )

DUP ( ChooseData valor valor )

MINUSONE ( ChooseData valor valor MINUSONE )

EQUALcase

:: DROPZERO TRUE ; ( SALE CON: ChooseData #0 T )

( ChooseData valor )

2DUP ( ChooseData valor ChooseData valor )

EQUALPOSCOMP ( ChooseData valor #i/#0 )

DUP#0<> ( ChooseData valor #i/#0 flag ) ( #i T // #0 F )

case

SWAPDROPTRUE ( SALE CON: Choosedata #i T )

( ChooseData valor #0 )

DROP ( Choosedata valor )

>HCOMP ( Choosedata' )

BINT1 ( Choosedata' #1 )

TRUE ( Choosedata' #1 T )

;

DROPFALSE

;

38.9.41 Mensaje 41 (campo)

Este mensaje es llamado cuando es presionada la tecla EDIT o la tecla CALC en un campo TEXTO, COMBOCHOOSE o COMBOFILER. Su misión es retornar el objeto que será editado o computado.

Si escoges la primera salida con efecto (valor T T), ese valor será editado o computado.

Si escoges la segunda salida con efecto (F T), la línea de edición se iniciará vacía al presionar EDIT, o el valor del campo no estará en el nivel 1 de la pila al presionar CALC.

Las acciones por defecto son:

a) Si el valor actual del campo no es MINUSONE, retornar el valor del campo, TRUE y TRUE.

b) Si el valor actual del campo es MINUSONE, retornar FALSE y TRUE.

**Entrada** Nada

**Salida (con efecto)** 3: valor

2: TRUE

1: TRUE

**Salida (con efecto)** 2: FALSE

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.42 Mensaje 42 (campo)

Este mensaje es llamado cada vez que se inicia una línea de edición. Su misión es retornar un menú que esté presente mientras la línea de edición esté activa.

Si este mensaje es manejado, ya no se llamará al mensaje 43.

La acción por defecto es retornar un menú con 12 elementos (ROMPTR 0B0 007) con las teclas:



**Entrada** Nada

**Salida (con efecto)** 2: { } o programa (menú)

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.43 Mensaje 43 (campo)

Este mensaje es llamado cada vez que se inicia una línea de edición. Su misión es retornar una lista con las primeras cuatro teclas de menú que estarán disponibles mientras la línea de edición esté activa.

Si este mensaje es manejado, se debe de retornar una lista con cuatro sublistas, cada una de estas será la definición de la tecla.

Si el mensaje 42 es manejado, este mensaje ya no será llamado.

**Entrada** Nada

**Salida (con efecto)** 3: { } (lista con cuatro elementos)

2: TRUE

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.44 Mensaje 44 (campo)

Este mensaje es llamado cuando se quiere confirmar la línea de edición con ENTER u OK.

Su misión es verificar que haya una línea de edición válida, convertir reales a enteros, matrices a arreglos, verificar que el objeto de la línea de edición sea de tipo válido y de valor válido, asignar valor al campo, mostrar los cambios, decidir cual será el nuevo campo con el enfoque, cambiar el enfoque, etc.

**Entrada** Nada

**Salida (con efecto)** 1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.45 Mensaje 45 (campo)

Este mensaje es llamado cuando se quiere confirmar la línea de edición con ENTER u OK o cuando se quiere asignar un valor al campo desde la pila (luego de presionar la tecla CALC).

Su misión es verificar que el objeto sea de un tipo válido.

Si retornas TRUE ob TRUE, el objeto será considerado de un tipo válido.

Si retornas FALSE ob TRUE, el objeto no se asignará al campo y se mostrará el mensaje “Invalid object type”.

Si retornas la salida sin efecto, se ejecutará el código por defecto para decidir si valor es de un tipo correcto.

valor es lo que se trata de introducir como valor del campo.

ob' es un objeto cualquiera.

**Entrada** 2: valor

1: valor

**Salida (con efecto)** 3: TRUE o FALSE

2: ob'

1: TRUE

**Salida (sin efecto)** 3: valor

2: valor

1: FALSE

38.9.46 Mensaje 46 (campo)

Este mensaje es llamado cuando se quiere confirmar la línea de edición con ENTER u OK o cuando se quiere asignar un valor al campo desde la pila (luego de presionar la tecla CALC).

Su misión es verificar que el objeto tenga un valor válido.

Por ejemplo, puedes verificar que un número real sea entero y positivo y/o se encuentre en un intervalo especificado.

Este mensaje es llamado después del mensaje 45.

Si retornas TRUE TRUE, el objeto será asignado al campo (similar a la salida sin efecto).

Si retornas FALSE TRUE, el objeto no se asignará al campo y se mostrará el mensaje “Invalid object value”.

Entrada 1: valor

Salida (con efecto) 2: TRUE o FALSE

1: TRUE

Salida (sin efecto) 2: valor

1: FALSE

38.9.47 Mensaje 47 (campo)

Este mensaje es llamado inmediatamente después de que un campo ha cambiado su valor.

Puedes hacer cualquier cosa aquí. No hay diferencia si retornas TRUE o FALSE.

**Entrada** Nada

**Salida** 1: TRUE o FALSE

38.9.48 Mensaje 48 (campo)

Este mensaje es llamado cuando se presiona la tecla ABAJO y su misión es decidir cual será el campo que recibirá el enfoque (no cambia el enfoque).

**Entrada** Nada

**Salida (con efecto)** 2: #campo

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.49 Mensaje 49 (campo)

Este mensaje es llamado cuando se presiona la tecla ARRIBA y su misión es decidir cual será el campo que recibirá el enfoque (no cambia el enfoque).

**Entrada** Nada

**Salida (con efecto)** 2: #campo

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.50 Mensaje 50 (campo)

Este mensaje es llamado cuando se presiona la tecla ShiftDerecho+ABAJO y su misión es decidir cual será el campo que recibirá el enfoque (no cambia el enfoque).

**Entrada** Nada

**Salida (con efecto)** 2: #campo

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.51 Mensaje 51 (campo)

Este mensaje es llamado cuando se presiona la tecla ShiftDerecho+ARRIBA y su misión es decidir cual será el campo que recibirá el enfoque (no cambia el enfoque).

**Entrada** Nada

**Salida (con efecto)** 2: #campo

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.52 Mensaje 52 (campo)

Este mensaje es llamado cuando se presiona la tecla DERECHA y su misión es decidir cual será el campo que recibirá el enfoque (no cambia el enfoque).

**Entrada** Nada

**Salida (con efecto)** 2: #campo

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.53 Mensaje 53 (campo)

Este mensaje es llamado cuando se presiona la tecla IZQUIERDA y su misión es decidir cual será el campo que recibirá el enfoque (no cambia el enfoque).

**Entrada** Nada

**Salida (con efecto)** 2: #campo

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.54 Mensaje 54 (campo)

Este mensaje es llamado cuando se presiona la tecla OK o ENTER cuando hay una línea de edición activa y el objeto que se estaba editando (de tipo y valor válido) ha pasado a ser el nuevo valor del campo. Su misión es decidir cual será el campo que a continuación recibirá el enfoque (no cambia el enfoque).

**Entrada** Nada

**Salida (con efecto)** 2: #campo

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.55 Mensaje 55 (campo)

Este mensaje es llamado cuando se intenta que el campo tenga el enfoque.

Su misión es decidir si el campo merece o no merece tener el enfoque.

Si colocas TRUE TRUE, el campo si tendrá el enfoque.

Si colocas FALSE TRUE, el campo no tendrá el enfoque.

Si escoges la salida sin efecto, el campo si tendrá el enfoque.

Este mensaje también es llamado cuando se resetean todos los campos y se decidirá si el campo actual debe continuar o no con el enfoque.

Cuando decides que el campo actual no conservará el enfoque, entonces el enfoque pasará después al primer campo, excepto si decides que sea hacia otro campo con el mensaje 14.

**Entrada** Nada

**Salida (con efecto)** 2: flag

1: TRUE

**Salida (sin efecto)** 1: FALSE

38.9.56 Mensaje 56 (formulario)

Este mensaje es llamado cuando el formulario está a punto de finalizar debido a la opresión de la tecla ENTER o la tecla OK. Es llamado antes de que el POL haya finalizado.

Puedes hacer cualquier cosa aquí. No hay diferencia si retornas TRUE o FALSE.

**Entrada** Nada

**Salida** 1: TRUE o FALSE

A continuación algunos messages handlers de ^IfMain y su equivalente en DoInputForm.

|  |  |  |
| --- | --- | --- |
| Mensaje ^IfMain | Mensaje DoInputForm | Descripción |
| 0 (c y f)  #ct #p 🡪 acción T | 8 (c)  #ct #p 🡪 acción T T | Asigna una acción a la tecla presionada |
| 1 (c)  #c 🡪 #c' flag | 19 (c)  🡪 flag | Cuando un campo está a punto de perder el enfoque |
| 3 (c)  🡪 flag | 20 (c)  🡪 flag | Cuando un campo acaba de recibir el enfoque |
|  |  |  |
| 7 (f)  #0 🡪 #c T | 14 (f)  🡪 #c T | Al inicio del formulario para decidir que campo tiene primero el enfoque |
| 11 (f)  {} 🡪 {} {}' T | 15 (f)  🡪 menú T | Menú del formulario |
| 12 (f)  🡪 {} T | 16 (f)  🡪 {} T T | Tres últimas teclas de la primera página del menú |
| 14 (f)  🡪 T/F T | 28 (f)  🡪 T/F T | Al presionar CANCL u ON |
| 16 (f)  🡪 T/F T | 29 (f)  🡪 T/F T | Al presionar tecla ENTER u OK |
| 17 (c y f)  🡪 T | 26 (c)  🡪 T | Al presionar tecla CHOOSE |
| 18 (f)  🡪 T | 36 (c)  🡪 T | Al presionar tecla TYPES |
| 22 (c) (hay $buf)  🡪 | 46 (c) (no hay $buf)  Valor 🡪 T/F | Validar objeto ingresado |
| 24 (c y f)  🡪 T | 38 (c)  🡪 T | Al presionar +/- en campo choose o check.  (choose o combochoose en DoInputForm) |
| 25 (c y f)  valor 🡪 | 23 (c)  🡪 | Al presionar la tecla EDIT |

38.10 Ejemplos DoInputForm

**Ejemplo 1a DoInputForm**

**Un NULLNAME para actualizar un campo al cambiar su valor.**

Cuando se crean formularios de entrada con ^IfMain, podemos usar el comando

FLASHPTR IfSetFieldValue ( valor #c 🡪 )

el cual cambia el valor del campo especificado, muestra el cambio en la pantalla y llama a los message handlers correspondientes al cambio en el campo.

Para formularios de entrada en DoInputForm tenemos al comando

ROMPTR sFldVal ( valor #c 🡪 )

Desgraciadamente este comando sólo cambia el valor del campo especificado, pero no muestra cambios en pantalla ni llama a los mensajes.

El siguiente NULLNAME lo puedes usar en formularios DoInputForm para cambiar el valor de un campo.

\* Cambia el valor de un campo.

\* Muestra el nuevo valor en pantalla.

\* Llama al mensaje 47 del campo que se está cambiando.

NULLNAME DoSetFieldValue ( valor #c -> )

:: ( valor #c )

DUPUNROT ( #c valor #c )

ROMPTR sFldVal ( #c ) ( cambia valor del campo )

DUP ( #c #c )

FLASHPTR IFEDispField ( #c ) ( muestra valor del campo )

ROMPTR 0B0 041 ( ) ( llama al mensaje 47 del campo )

;

**Ejemplo 1b DoInputForm**

**Un NULLNAME para cambiar el enfoque a otro campo.**

Cuando se crean formularios de entrada con ^IfMain, podemos usar el comando

FLASHPTR IfSetField ( #c 🡪 )

el cual permite cambiar el enfoque hacia cualquier campo.

Para formularios de entrada en DoInputForm tenemos a los comandos

FLASHPTR ChangeFocus ( #c 🡪 )

ROMPTR 0B0 097 ( #c 🡪 )

El primero sólo cambia el enfoque pero no actualiza la pantalla. El segundo cambia el enfoque y actualiza la visualización de los campos (invierte píxeles), pero no actualiza la ayuda del campo.

El siguiente NULLNAME lo puedes usar en formularios DoInputForm para cambiar el campo que tendrá el enfoque, actualizar los campos (invertir píxeles) y actualizar la ayuda.

\* Mueve el enfoque al campo especificado

\* Actualiza visualización de campos (invierte píxeles).

\* Actualiza la zona de la ayuda.

NULLNAME ChangeFocus3 ( #c -> )

:: ( #c )

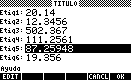
DUP ( #c #c )

ROMPTR 0B0 097 ( #c ) ( cambia el enfoque e invierte píxeles )

TRUESWAP\_ ( T #c )

ROMPTR 0B0 017 ( ) ( muestra ayuda del campo justo encima de los menús )

;

Ejemplo 2 DoInputForm

**Un formulario de entrada con 6 campos**

En este ejemplo se muestra un formulario de entrada con 6 campos **TEXTO** en una columna.

Las etiquetas tienen sus posiciones #x en la ubicación 0.

Los campos tienen sus posiciones #y en las ubicaciones 8, 17, 26, 35, 44 y 53.

Desgraciadamente, en DoInputForm no está disponible toda la pantalla para mostrar etiquetas (sólo hasta la fila 46). Sin embargo, en este ejemplo podemos vencer este obstáculo usando el message handler número 4 para poder dibujar etiquetas en toda la pantalla.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoInFo1COL ( -> ob1 ob2 ob3 ob4 ob5 ob6 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq1:" 0 10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Etiq2:" 0 19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Etiq3:" 0 28 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Etiq4:" 0 37 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Etiq5:" 0 46 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Etiq6:" 0 55 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT106 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 2

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT106 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

12.3456 ( ValorInicial )

\* CAMPO 3

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT26 ( #y: 26 para fila 3/6 )

BINT106 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

502.367 ( ValorInicial )

\* CAMPO 4

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT35 ( #y: 35 para fila 4/6 )

BINT106 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

111.2561 ( ValorInicial )

\* CAMPO 5

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT44 ( #y: 44 para fila 5/6 )

BINT106 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

87.25948 ( ValorInicial )

\* CAMPO 6

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT53 ( #y: 53 para fila 6/6 )

BINT106 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

19.356 ( ValorInicial )

BINT6 ( #Netiq )

BINT6 ( #Ncamp )

\* Message handler del formulario

' ::

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT63 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

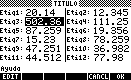
DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 ob3 ob4 ob5 ob6 T // F )

;

Ejemplo 3 DoInputForm

**Un formulario de entrada con 12 campos en 2 columnas**

En este ejemplo se muestra un formulario de entrada con 12 campos **TEXTO** en 2 columnas.

Las etiquetas tienen sus posiciones #x en las ubicaciones 0 y 67.

Los campos tienen sus posiciones #y en las ubicaciones 8, 17, 26, 35, 44 y 53.

Se ha colocado una línea vertical (grob de ancho 1) como etiqueta en la posición x=65, y=8

Desgraciadamente, en DoInputForm no está disponible toda la pantalla para mostrar etiquetas (sólo hasta la fila 46). Sin embargo, en este ejemplo podemos vencer este obstáculo usando el message handler número 4 para poder dibujar etiquetas en toda la pantalla y para que una etiqueta también pueda definirse como grob en DoInputForm

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoInFo2COL ( -> ob1 ob2 ob3 ... ob12 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq1:" 0 10 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+2 )

"Etiq2:" 67 10 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+2 )

"Etiq3:" 0 19 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+2 )

"Etiq4:" 67 19 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+2 )

"Etiq5:" 0 28 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+2 )

"Etiq6:" 67 28 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+2 )

"Etiq7:" 0 37 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+2 )

"Etiq8:" 67 37 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+2 )

"Etiq9:" 0 46 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+2 )

"Eti10:" 67 46 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+2 )

"Eti11:" 0 55 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+2 )

"Eti12:" 67 55 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+2 )

BINT54 BINT1 MAKEGROB INVGROB 65 8 ( #xetiq=65 ) ( #yetiq=8 )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 2

'DROPFALSE ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

12.345 ( ValorInicial )

\* CAMPO 3

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

502.36 ( ValorInicial )

\* CAMPO 4

'DROPFALSE ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

111.25 ( ValorInicial )

\* CAMPO 5

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT26 ( #y: 26 para fila 3/6 )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

87.259 ( ValorInicial )

\* CAMPO 6

'DROPFALSE ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT26 ( #y: 26 para fila 3/6 )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

19.356 ( ValorInicial )

\* CAMPO 7

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT35 ( #y: 35 para fila 4/6 )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

15.23 ( ValorInicial )

\* CAMPO 8

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT35 ( #y: 35 para fila 4/6 )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

78.259 ( ValorInicial )

\* CAMPO 9

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT44 ( #y: 44 para fila 5/6 )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

47.251 ( ValorInicial )

\* CAMPO 10

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT44 ( #y: 44 para fila 5/6 )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

36.982 ( ValorInicial )

\* CAMPO 11

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT53 ( #y: 53 para fila 6/6 )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

44.512 ( ValorInicial )

\* CAMPO 12

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT53 ( #y: 53 para fila 6/6 )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

77.98 ( ValorInicial )

BINT13 ( #Netiq )

BINT12 ( #Ncamp )

\* Message handler del formulario

' ::

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT63 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

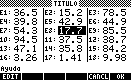
DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( -> ob1 ob2 ob3 ... ob12 T // F )

;

Ejemplo 4 DoInputForm

**Un formulario de entrada con 18 campos en 3 columnas**

En este ejemplo se muestra un formulario de entrada con 18 campos **TEXTO** en 3 columnas.

Las etiquetas tienen sus posiciones #x en las ubicaciones 0 y 67.

Los campos tienen sus posiciones #y en las ubicaciones 8, 17, 26, 35, 44 y 53.

Se han colocado dos líneas verticales (grobs de ancho 1) como etiquetas en las posiciones (43,8) y (87,8)

Desgraciadamente, en DoInputForm no está disponible toda la pantalla para mostrar etiquetas (sólo hasta la fila 46). Sin embargo, en este ejemplo podemos vencer este obstáculo usando el message handler número 4 para poder dibujar etiquetas en toda la pantalla y para que una etiqueta también pueda definirse como grob en DoInputForm

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoInFo3COL ( -> ob1 ob2 ob3 ... ob18 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"E1:" 0 10 ( #xetiq=0 COL1/3 ) ( #yetiq=#ycampo+2 )

"E2:" 45 10 ( #xetiq=45 COL2/3 ) ( #yetiq=#ycampo+2 )

"E3:" 89 10 ( #xetiq=89 COL3/3 ) ( #yetiq=#ycampo+2 )

"E4:" 0 19 ( #xetiq=0 COL1/3 ) ( #yetiq=#ycampo+2 )

"E5:" 45 19 ( #xetiq=45 COL2/3 ) ( #yetiq=#ycampo+2 )

"E6:" 89 19 ( #xetiq=89 COL3/3 ) ( #yetiq=#ycampo+2 )

"E7:" 0 28 ( #xetiq=0 COL1/3 ) ( #yetiq=#ycampo+2 )

"E8:" 45 28 ( #xetiq=45 COL2/3 ) ( #yetiq=#ycampo+2 )

"E9:" 89 28 ( #xetiq=89 COL3/3 ) ( #yetiq=#ycampo+2 )

"10:" 0 37 ( #xetiq=0 COL1/3 ) ( #yetiq=#ycampo+2 )

"11:" 45 37 ( #xetiq=45 COL2/3 ) ( #yetiq=#ycampo+2 )

"12:" 89 37 ( #xetiq=89 COL3/3 ) ( #yetiq=#ycampo+2 )

"13:" 0 46 ( #xetiq=0 COL1/3 ) ( #yetiq=#ycampo+2 )

"14:" 45 46 ( #xetiq=45 COL2/3 ) ( #yetiq=#ycampo+2 )

"15:" 89 46 ( #xetiq=89 COL3/3 ) ( #yetiq=#ycampo+2 )

"16:" 0 55 ( #xetiq=0 COL1/3 ) ( #yetiq=#ycampo+2 )

"17:" 45 55 ( #xetiq=45 COL2/3 ) ( #yetiq=#ycampo+2 )

"18:" 89 55 ( #xetiq=89 COL3/3 ) ( #yetiq=#ycampo+2 )

BINT54 BINT1 MAKEGROB INVGROB 43 8 ( #xetiq=65 ) ( #yetiq=8 )

BINT54 BINT1 MAKEGROB INVGROB 87 8 ( #xetiq=65 ) ( #yetiq=8 )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT13 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT29 ( #w: 42-#x para columna 1/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

36.5 ( ValorInicial )

\* CAMPO 2

'DROPFALSE ( MH del campo )

BINT58 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT28 ( #w: 86-#x para columna 2/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

15.2 ( ValorInicial )

\* CAMPO 3

'DROPFALSE ( MH del campo )

BINT102 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT29 ( #w: 131-#x para columna 3/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

78.5 ( ValorInicial )

\* CAMPO 4

'DROPFALSE ( MH del campo )

BINT13 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT29 ( #w: 42-#x para columna 1/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

39.8 ( ValorInicial )

\* CAMPO 5

'DROPFALSE ( MH del campo )

BINT58 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT28 ( #w: 86-#x para columna 2/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

42.9 ( ValorInicial )

\* CAMPO 6

'DROPFALSE ( MH del campo )

BINT102 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT29 ( #w: 131-#x para columna 3/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

44.9 ( ValorInicial )

\* CAMPO 7

'DROPFALSE ( MH del campo )

BINT13 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT26 ( #y: 26 para fila 3/6 )

BINT29 ( #w: 42-#x para columna 1/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

54.3 ( ValorInicial )

\* CAMPO 8

'DROPFALSE ( MH del campo )

BINT58 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT26 ( #y: 26 para fila 3/6 )

BINT28 ( #w: 86-#x para columna 2/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

17.7 ( ValorInicial )

\* CAMPO 9

'DROPFALSE ( MH del campo )

BINT102 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT26 ( #y: 26 para fila 3/6 )

BINT29 ( #w: 131-#x para columna 3/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

85.5 ( ValorInicial )

\* CAMPO 10

'DROPFALSE ( MH del campo )

BINT13 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT35 ( #y: 35 para fila 4/6 )

BINT29 ( #w: 42-#x para columna 1/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

94.5 ( ValorInicial )

\* CAMPO 11

'DROPFALSE ( MH del campo )

BINT58 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT35 ( #y: 35 para fila 4/6 )

BINT28 ( #w: 86-#x para columna 2/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

37.5 ( ValorInicial )

\* CAMPO 12

'DROPFALSE ( MH del campo )

BINT102 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT35 ( #y: 35 para fila 4/6 )

BINT29 ( #w: 131-#x para columna 3/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

38.7 ( ValorInicial )

\* CAMPO 13

'DROPFALSE ( MH del campo )

BINT13 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT44 ( #y: 44 para fila 5/6 )

BINT29 ( #w: 42-#x para columna 1/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

47.1 ( ValorInicial )

\* CAMPO 14

'DROPFALSE ( MH del campo )

BINT58 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT44 ( #y: 44 para fila 5/6 )

BINT28 ( #w: 86-#x para columna 2/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

35.8 ( ValorInicial )

\* CAMPO 15

'DROPFALSE ( MH del campo )

BINT102 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT44 ( #y: 44 para fila 5/6 )

BINT29 ( #w: 131-#x para columna 3/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

84.2 ( ValorInicial )

\* CAMPO 16

'DROPFALSE ( MH del campo )

BINT13 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT53 ( #y: 53 para fila 6/6 )

BINT29 ( #w: 42-#x para columna 1/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

3.26 ( ValorInicial )

\* CAMPO 17

'DROPFALSE ( MH del campo )

BINT58 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT53 ( #y: 53 para fila 6/6 )

BINT28 ( #w: 86-#x para columna 2/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

1.41 ( ValorInicial )

\* CAMPO 18

'DROPFALSE ( MH del campo )

BINT102 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT53 ( #y: 53 para fila 6/6 )

BINT29 ( #w: 131-#x para columna 3/3 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

9.98 ( ValorInicial )

BINT20 ( #Netiq )

BINT18 ( #Ncamp )

\* Message handler del formulario

' ::

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT63 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 ob3 ... ob18 T // F )

;

Ejemplo 5 DoInputForm

**Título inverso en DoInputForm**

En este ejemplo el **Titulo** del formulario de entrada será mostrado con fondo oscuro.

Para esto se llamará al mensaje número 2.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoTituloInverso ( -> ob T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiqueta:" 0 10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT37 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT94 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

20.14 ( ValorInicial )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

\* Message handler del formulario

' ::

\* Retorna un grob de tamaño 131x7 que será el título.

\* Recuerda que para llamar al título debes usar el comando 12GETLAM

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* El título será un grob de tamaño 131 x 7 con fondo oscuro

BINT2 #=casedrop ( F ) ( -> grob131x7 T // F )

:: ( )

12GETLAM ( $/# ) ( retorna el título )

TITULO->GROB131x7\_INVERSA ( grob131x7 )

TRUE ( grob131x7 T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob T // F )

;

\*\*\* Dada una cadena o bint la pone en un grob de

\*\*\* tamaño 131 x 7 con fondo oscuro en la pila

NULLNAME TITULO->GROB131x7\_INVERSA ( $/# -> grob131x7 )

:: ( $/# )

DUPTYPEBINT? ( $/# flag )

IT JstGETTHEMSG

( $ )

BINT1 BINT32 SUB$ ( $ ) ( corta la cadena si es mayor a 32 caract )

BINT7 BINT131 ( $ 7 131 )

MAKEGROB ( $ grob131x7\_blanco )

BINT33 ( $ grob131x7\_blanco 33 )

3PICK ( $ grob131x7\_blanco 33 $ )

LEN$ ( $ grob131x7\_blanco 33 #w )

#-#2/ ( $ grob131x7\_blanco #[33-w]/2 )

Blank$ ( $ grob131x7\_blanco $' )

ROT ( grob131x7\_blanco $' $ )

&$ ( grob131x7\_blanco $'' )

$>grob ( grob131x7\_blanco grob' )

ONEONE ( grob131x7\_blanco grob' #1 #1 )

Repl ( grob131x7 ) ( Copia grb2 en grb1 en modo REPLACE )

INVGROB ( grob131x7\_inversa )

\* Lo siguiente es sólo para redondear las esquinas del grob

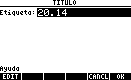
ZEROZERO PixonW ( grob131x7\_inversa )

BINT130 BINT0 PixonW ( grob131x7\_inversa )

BINT0 BINT6 PixonW ( grob131x7\_inversa )

BINT130 BINT6 PixonW ( grob131x7\_inversa )

;

Ejemplo 6 DoInputForm

**Título subrayado en DoInputForm**

En este ejemplo el **Titulo** del formulario de entrada será mostrado con una línea debajo.

Para esto se llamará al mensaje número 2.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoTitSubrayado ( -> ob T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiqueta:" 0 10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT37 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT94 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

20.14 ( ValorInicial )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

\* Message handler del formulario

' ::

\* Retorna un grob de tamaño 131x7 que será el título.

\* Recuerda que para llamar al título debes usar el comando 12GETLAM

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* El título será mostrado con una línea horizontal debajo

BINT2 #=casedrop ( F ) ( -> grob131x7 T // F )

:: ( )

12GETLAM ( $/# ) ( retorna el título )

TITULO->GROB131x7\_SUBRAYADO ( grob131x7 )

TRUE ( grob131x7 T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob T // F )

;

\*\*\* Dada una cadena o bint la pone en un grob de

\*\*\* tamaño 131 x 7 con una línea horizontal debajo

NULLNAME TITULO->GROB131x7\_SUBRAYADO ( $/# -> grob131x7 )

:: ( $/# )

DUPTYPEBINT? ( $/# flag )

IT JstGETTHEMSG

( $ )

BINT1 BINT32 SUB$ ( $ ) ( corta la cadena si es mayor a 32 caract )

BINT7 BINT131 ( $ 7 131 )

MAKEGROB ( $ grob131x7\_blanco )

BINT33 ( $ grob131x7\_blanco 33 )

3PICK ( $ grob131x7\_blanco 33 $ )

LEN$ ( $ grob131x7\_blanco 33 #w )

#-#2/ ( $ grob131x7\_blanco #[33-w]/2 )

Blank$ ( $ grob131x7\_blanco $' )

ROT ( grob131x7\_blanco $' $ )

&$ ( grob131x7\_blanco $'' )

$>grob ( grob131x7\_blanco grob' )

BINT1

BINT0 ( grob131x7\_blanco grob' #1 #0 )

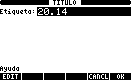
Repl ( grob131x7 ) ( Copia grb2 en grb1 en modo REPLACE )

BINT0 BINT6

BINT130 BINT6

LineB ( grob131x7 ) ( Dibuja una línea negra en el grob )

;

Ejemplo 7 DoInputForm

**Título mostrado más arriba en DoInputForm**

En este ejemplo el **Titulo** del formulario de entrada será mostrado un píxel mas arriba, lo cual permite mostrar los campos 1 píxel más arriba.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoTitArriba ( -> ob T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiqueta:" 0 9 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT37 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT7 ( #y )

BINT94 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

20.14 ( ValorInicial )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

\* Message handler del formulario

' ::

\* Retorna un grob de tamaño 131x7 que será el título.

\* Recuerda que para llamar al título debes usar el comando 12GETLAM

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* El título será mostrado un pixel más arriba de lo normal

BINT2 #=casedrop ( F ) ( -> grob131x7 T // F )

:: ( )

12GETLAM ( $/# ) ( retorna el título )

TITULO->GROB131x7\_ARRIBA ( grob131x7 )

TRUE ( grob131x7 T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob T // F )

;

\*\*\* Dada una cadena o bint la pone en un grob de

\*\*\* tamaño 131 x 7 en la parte superior

NULLNAME TITULO->GROB131x7\_ARRIBA ( $/# -> grob131x7 )

:: ( $/# )

DUPTYPEBINT? ( $/# flag )

IT JstGETTHEMSG

( $ )

BINT1 BINT30 SUB$ ( $ ) ( corta la cadena si es mayor a 30 caract )

SPACE$ SWAP&$ ( $ ) ( agrega espacio al inicio de la cadena )

APPEND\_SPACE ( $ ) ( agrega espacio al final de la cadena )

BINT7 BINT131 ( $ 7 131 )

MAKEGROB ( $ grob131x7\_blanco )

0 0 131 5 ( $ grob131x7\_blanco 0 0 131 5 )

FBoxB ( $ grob131x7 ) ( Dibuja rectángulo negro relleno )

OVER ( $ grob131x7 $ )

$>grob ( $ grob131x7 grob )

BINT131 ( $ grob131x7 grob 131 )

BINT4 ( $ grob131x7 grob 131 4 )

5ROLL ( grob131x7 grob 131 4 $ )

LEN$ ( grob131x7 grob 131 4 #ncaract )

#\* ( grob131x7 grob 131 #4·ncaract )

#- ( grob131x7 grob #131-4·ncaract )

#2/ ( grob131x7 grob #[131-4·ncaract]/2 )

BINT0 ( grob131x7 grob #[131-4·ncaract]/2 0 )

Repl ( grob131x7 ) ( Copia grb2 en grb1 en modo REPLACE )

\* Lo siguiente es sólo para redondear las esquinas del grob

ZEROZERO PixonW ( grob131x7\_inversa )

BINT130 BINT0 PixonW ( grob131x7\_inversa )

BINT0 BINT4 PixonW ( grob131x7\_inversa )

BINT130 BINT4 PixonW ( grob131x7\_inversa )

;

Ejemplo 8 DoInputForm

**Un campo motrado con Minifuente**

En este ejemplo se muestra un formulario de entrada con 1 campo **TEXTO** mostrado en Minifuente.

Para esto, se llama al mensaje número 9 en el campo que se desea mostrar con Minifuente, el cual es llamado cuando se va a dibujar un campo y debe retornar el campo convertido en grob.

En este código, en el mensaje 9 se retorna un grob que representa al campo usando la minifuentey se respeta al parámetro **Decompile** del campo.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoInputFMinif ( -> ob T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiqueta:" 0 9 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

\* CAMPO 1

\* Message handler del campo

' ::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Retorna un grob que representa al campo usando la minifuente

\*\*\* Respeta al parámetro Decompile del campo

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 045 ( #c $ ) ( Retorna $ del campo respetando Decompile )

OVER ( #c $ #c )

ROMPTR 0B0 0CA ( #c $ #b #h ) ( Retorna ancho y alto del campo )

ROTDUP ( #c #b #h $ $ )

LEN$ ( #c #b #h $ #car )

BINT4 ( #c #b #h $ #car #4 )

#\* ( #c #b #h $ #4·car )

4PICK ( #c #b #h $ #4·car #b )

DUPUNROT ( #c #b #h $ #b #4·car #b )

#< ( #c #b #h $ #b flag )

ITE

DROP ( #c #b #h $ )

:: ( #c #b #h $ #b )

BINT4 ( #c #b #h $ #b #4 )

#/ ( #c #b #h $ #residuo #b/4 )

SWAPDROP ( #c #b #h $ #b/4 )

1\_#1-SUB$ ( #c #b #h $' )

CHR\_... ( #c #b #h $' chr )

>T$ ( #c #b #h $'' )

;

( #c #b #h $'' )

$>grob ( #c #b #h grob )

3UNROLL ( #c grob #b #h )

SWAP ( #c grob #h #b )

MAKEGROB ( #c grob grobbxh )

DUPUNROT ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' )

TRUE ( #c grobbxh' T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT37 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y )

BINT94 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

20.14 ( ValorInicial )

BINT1 ( #Netiq )

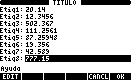
BINT1 ( #Ncamp )

'DROPFALSE ( MH del DoInputForm )

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob T // F )

;

Ejemplo 9 DoInputForm

**Un formulario de entrada con 8 campos en Minifuente**

En este ejemplo se muestra un formulario de entrada con 8 campos **TEXTO** en una columna.

Las etiquetas tienen sus posiciones #x en la ubicación 0.

Los campos tienen sus posiciones #y en la ubicaciones 6, 13, 20, 27, 34, 41, 48 y 55.

Las etiquetas tienen sus posiciones #y en una unidad más que las del campo correspondiente.

Se llama al mensaje número 9 en los campos que se desean mostrar con Minifuente.

Se llama al mensaje número 2 en el formulario para mostrar el título 1 píxel más arriba de lo normal.

Se llama al mensaje número 4 en el formulario para poder mostrar las etiquetas en toda la pantalla.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME Do1ColMini ( -> ob1 ob2 ob3 ob4 ob5 ob6 ob7 ob8 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq1:" 0 7 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq2:" 0 14 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq3:" 0 21 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq4:" 0 28 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq5:" 0 35 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq6:" 0 42 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq7:" 0 49 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq8:" 0 56 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

\* CAMPO 1

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT6 ( #y: 6 para fila 1/8 MiniF )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 2

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT13 ( #y: 13 para fila 2/8 MiniF )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

12.3456 ( ValorInicial )

\* CAMPO 3

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT20 ( #y: 20 para fila 3/8 MiniF )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

502.367 ( ValorInicial )

\* CAMPO 4

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT27 ( #y: 27 para fila 4/8 MiniF )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

111.2561 ( ValorInicial )

\* CAMPO 5

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT34 ( #y: 34 para fila 5/8 MiniF )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

87.25948 ( ValorInicial )

\* CAMPO 6

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT41 ( #y: 41 para fila 6/8 MiniF )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

19.356 ( ValorInicial )

\* CAMPO 7

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT48 ( #y: 48 para fila 7/8 MiniF )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

42.589 ( ValorInicial )

\* CAMPO 8

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT55 ( #y: 55 para fila 8/8 MiniF )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

777.15 ( ValorInicial )

BINT8 ( #Netiq )

BINT8 ( #Ncamp )

\* Message handler del formulario

' ::

\* Retorna un grob de tamaño 131x7 que será el título.

\* Recuerda que para llamar al título debes usar el comando 12GETLAM

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* El título será mostrado un pixel más arriba de lo normal

BINT2 #=casedrop ( F ) ( -> grob131x7 T // F )

:: ( )

12GETLAM ( $/# ) ( retorna el título )

TITULO->GROB131x7\_ARRIBA ( grob131x7 )

TRUE ( grob131x7 T )

;

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT64 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 ob3 ob4 ob5 ob6 ob7 ob8 T // F )

;

\*\*\* Dada una cadena o bint la pone en un grob de

\*\*\* tamaño 131 x 7 en la parte superior

NULLNAME TITULO->GROB131x7\_ARRIBA ( $/# -> grob131x7 )

:: ( $/# )

DUPTYPEBINT? ( $/# flag )

IT JstGETTHEMSG

( $ )

BINT1 BINT30 SUB$ ( $ ) ( corta la cadena si es mayor a 30 caract )

SPACE$ SWAP&$ ( $ ) ( agrega espacio al inicio de la cadena )

APPEND\_SPACE ( $ ) ( agrega espacio al final de la cadena )

BINT7 BINT131 ( $ 7 131 )

MAKEGROB ( $ grob131x7\_blanco )

0 0 131 5 ( $ grob131x7\_blanco 0 0 131 5 )

FBoxB ( $ grob131x7 ) ( Dibuja rectángulo negro relleno )

OVER ( $ grob131x7 $ )

$>grob ( $ grob131x7 grob )

BINT131 ( $ grob131x7 grob 131 )

BINT4 ( $ grob131x7 grob 131 4 )

5ROLL ( grob131x7 grob 131 4 $ )

LEN$ ( grob131x7 grob 131 4 #ncaract )

#\* ( grob131x7 grob 131 #4·ncaract )

#- ( grob131x7 grob #131-4·ncaract )

#2/ ( grob131x7 grob #[131-4·ncaract]/2 )

BINT0 ( grob131x7 grob #[131-4·ncaract]/2 0 )

Repl ( grob131x7 ) ( Copia grb2 en grb1 en modo REPLACE )

\* Lo siguiente es sólo para redondear las esquinas del grob

ZEROZERO PixonW ( grob131x7\_inversa )

BINT130 BINT0 PixonW ( grob131x7\_inversa )

BINT0 BINT4 PixonW ( grob131x7\_inversa )

BINT130 BINT4 PixonW ( grob131x7\_inversa )

;

\* Message handler del campo:

NULLNAME DoMH9\_MiniFuente

::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Retorna un grob que representa al campo usando la minifuente

\*\*\* Respeta al parámetro Decompile del campo

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 045 ( #c $ ) ( Retorna $ del campo respetando Decompile )

OVER ( #c $ #c )

ROMPTR 0B0 0CA ( #c $ #b #h ) ( Retorna ancho y alto del campo )

ROTDUP ( #c #b #h $ $ )

LEN$ ( #c #b #h $ #car )

BINT4 ( #c #b #h $ #car #4 )

#\* ( #c #b #h $ #4·car )

4PICK ( #c #b #h $ #4·car #b )

DUPUNROT ( #c #b #h $ #b #4·car #b )

#< ( #c #b #h $ #b flag )

ITE

DROP ( #c #b #h $ )

:: ( #c #b #h $ #b )

BINT4 ( #c #b #h $ #b #4 )

#/ ( #c #b #h $ #residuo #b/4 )

SWAPDROP ( #c #b #h $ #b/4 )

1\_#1-SUB$ ( #c #b #h $' )

CHR\_... ( #c #b #h $' chr )

>T$ ( #c #b #h $'' )

;

( #c #b #h $'' )

$>grob ( #c #b #h grob )

3UNROLL ( #c grob #b #h )

SWAP ( #c grob #h #b )

MAKEGROB ( #c grob grobbxh )

DUPUNROT ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' )

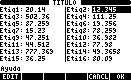
TRUE ( #c grobbxh' T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

Ejemplo 10 DoInputForm

**Un formulario de entrada con 16 campos en 2 columnas en Minifuente**

En este ejemplo se muestra un formulario de entrada con 16 campos **TEXTO** en 2 columnas.

Las etiquetas tienen sus posiciones #x en las ubicaciones 0 y 67.

Los campos tienen sus posiciones #y en la ubicaciones 6, 13, 20, 27, 34, 41, 48 y 55.

Las etiquetas tienen sus posiciones #y en una unidad más que las del campo correspondiente.

Se ha colocado una línea vertical (grob de ancho 1) como etiqueta en la posición x=65, y=6

Se llama al mensaje número 9 en los campos que se desean mostrar con Minifuente.

Se llama al mensaje número 2 en el formulario para mostrar el título 1 píxel más arriba de lo normal.

Se llama al mensaje número 4 en el formulario para poder mostrar las etiquetas en toda la pantalla y para permitir que las etiquetas también puedan definirse como grobs.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME Do2ColMini ( -> ob1 ob2 ... ob16 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq1:" 0 7 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq2:" 67 7 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq3:" 0 14 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq4:" 67 14 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq5:" 0 21 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq6:" 67 21 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq7:" 0 28 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq8:" 67 28 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq9:" 0 35 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti10:" 67 35 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti11:" 0 42 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti12:" 67 42 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti13:" 0 49 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti14:" 67 49 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti15:" 0 56 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti16:" 67 56 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

BINT55 BINT1 MAKEGROB INVGROB 65 7 ( #xetiq=65 ) ( #yetiq=7 )

\* CAMPO 1

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT6 ( #y: 6 para fila 1/8 MiniF )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 2

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT6 ( #y: 6 para fila 1/8 MiniF )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

12.345 ( ValorInicial )

\* CAMPO 3

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT13 ( #y: 13 para fila 2/8 MiniF )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

502.36 ( ValorInicial )

\* CAMPO 4

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT13 ( #y: 13 para fila 2/8 MiniF )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

111.25 ( ValorInicial )

\* CAMPO 5

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT20 ( #y: 20 para fila 3/8 MiniF )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

87.259 ( ValorInicial )

\* CAMPO 6

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT20 ( #y: 20 para fila 3/8 MiniF )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

19.356 ( ValorInicial )

\* CAMPO 7

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT27 ( #y: 27 para fila 4/8 MiniF )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

15.23 ( ValorInicial )

\* CAMPO 8

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT27 ( #y: 27 para fila 4/8 MiniF )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

78.259 ( ValorInicial )

\* CAMPO 9

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT34 ( #y: 34 para fila 5/8 MiniF )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

47.251 ( ValorInicial )

\* CAMPO 10

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT34 ( #y: 34 para fila 5/8 MiniF )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

36.982 ( ValorInicial )

\* CAMPO 11

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT41 ( #y: 41 para fila 6/8 MiniF )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

44.512 ( ValorInicial )

\* CAMPO 12

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT41 ( #y: 41 para fila 6/8 MiniF )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

77.98 ( ValorInicial )

\* CAMPO 13

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT48 ( #y: 48 para fila 7/8 MiniF )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

777.369 ( ValorInicial )

\* CAMPO 14

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT48 ( #y: 48 para fila 7/8 MiniF )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

49.3658 ( ValorInicial )

\* CAMPO 15

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT55 ( #y: 55 para fila 8/8 MiniF )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

36.25 ( ValorInicial )

\* CAMPO 16

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT55 ( #y: 55 para fila 8/8 MiniF )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

80.09 ( ValorInicial )

BINT17 ( #Netiq )

BINT16 ( #Ncamp )

\* Message handler del formulario

' ::

\* Retorna un grob de tamaño 131x7 que será el título.

\* Recuerda que para llamar al título debes usar el comando 12GETLAM

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* El título será mostrado un pixel más arriba de lo normal

BINT2 #=casedrop ( F ) ( -> grob131x7 T // F )

:: ( )

12GETLAM ( $/# ) ( retorna el título )

TITULO->GROB131x7\_ARRIBA ( grob131x7 )

TRUE ( grob131x7 T )

;

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT64 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 ... ob16 T // F )

;

\*\*\* Dada una cadena o bint la pone en un grob de

\*\*\* tamaño 131 x 7 en la parte superior

NULLNAME TITULO->GROB131x7\_ARRIBA ( $/# -> grob131x7 )

:: ( $/# )

DUPTYPEBINT? ( $/# flag )

IT JstGETTHEMSG

( $ )

BINT1 BINT30 SUB$ ( $ ) ( corta la cadena si es mayor a 30 caract )

SPACE$ SWAP&$ ( $ ) ( agrega espacio al inicio de la cadena )

APPEND\_SPACE ( $ ) ( agrega espacio al final de la cadena )

BINT7 BINT131 ( $ 7 131 )

MAKEGROB ( $ grob131x7\_blanco )

0 0 131 5 ( $ grob131x7\_blanco 0 0 131 5 )

FBoxB ( $ grob131x7 ) ( Dibuja rectángulo negro relleno )

OVER ( $ grob131x7 $ )

$>grob ( $ grob131x7 grob )

BINT131 ( $ grob131x7 grob 131 )

BINT4 ( $ grob131x7 grob 131 4 )

5ROLL ( grob131x7 grob 131 4 $ )

LEN$ ( grob131x7 grob 131 4 #ncaract )

#\* ( grob131x7 grob 131 #4·ncaract )

#- ( grob131x7 grob #131-4·ncaract )

#2/ ( grob131x7 grob #[131-4·ncaract]/2 )

BINT0 ( grob131x7 grob #[131-4·ncaract]/2 0 )

Repl ( grob131x7 ) ( Copia grb2 en grb1 en modo REPLACE )

\* Lo siguiente es sólo para redondear las esquinas del grob

ZEROZERO PixonW ( grob131x7\_inversa )

BINT130 BINT0 PixonW ( grob131x7\_inversa )

BINT0 BINT4 PixonW ( grob131x7\_inversa )

BINT130 BINT4 PixonW ( grob131x7\_inversa )

;

\* Message handler del campo:

NULLNAME DoMH9\_MiniFuente

::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Retorna un grob que representa al campo usando la minifuente

\*\*\* Respeta al parámetro Decompile del campo

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 045 ( #c $ ) ( Retorna $ del campo respetando Decompile )

OVER ( #c $ #c )

ROMPTR 0B0 0CA ( #c $ #b #h ) ( Retorna ancho y alto del campo )

ROTDUP ( #c #b #h $ $ )

LEN$ ( #c #b #h $ #car )

BINT4 ( #c #b #h $ #car #4 )

#\* ( #c #b #h $ #4·car )

4PICK ( #c #b #h $ #4·car #b )

DUPUNROT ( #c #b #h $ #b #4·car #b )

#< ( #c #b #h $ #b flag )

ITE

DROP ( #c #b #h $ )

:: ( #c #b #h $ #b )

BINT4 ( #c #b #h $ #b #4 )

#/ ( #c #b #h $ #residuo #b/4 )

SWAPDROP ( #c #b #h $ #b/4 )

1\_#1-SUB$ ( #c #b #h $' )

CHR\_... ( #c #b #h $' chr )

>T$ ( #c #b #h $'' )

;

( #c #b #h $'' )

$>grob ( #c #b #h grob )

3UNROLL ( #c grob #b #h )

SWAP ( #c grob #h #b )

MAKEGROB ( #c grob grobbxh )

DUPUNROT ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' )

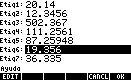
TRUE ( #c grobbxh' T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

Ejemplo 11 DoInputForm

**Un formulario de entrada con 7 campos  
No se muestra el título.**

En este ejemplo se muestra un formulario de entrada con 7 campos **TEXTO** en una columna.

Las etiquetas tienen sus posiciones #x en la ubicación 0.

Los campos tienen sus posiciones #y en las ubicaciones 0, 9, 18, 27, 36, 45 y 54.

Para no mostrar el título, se usa el mensaje número 1 en el formulario.

Desgraciadamente, en DoInputForm no está disponible toda la pantalla para mostrar etiquetas (sólo hasta la fila 46). Sin embargo, en este ejemplo podemos vencer este obstáculo usando el mensaje número 4 para poder dibujar etiquetas en toda la pantalla.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME Do1COLFS ( -> ob1 ob2 ob3 ob4 ob5 ob6 ob7 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq1:" 0 2 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Etiq2:" 0 11 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Etiq3:" 0 20 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Etiq4:" 0 29 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Etiq5:" 0 38 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Etiq6:" 0 47 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Etiq7:" 0 56 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT0 ( #y: 0 para fila 1/7 PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 2

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT9 ( #y: 9 para fila 2/7 PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

12.3456 ( ValorInicial )

\* CAMPO 3

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT18 ( #y: 18 para fila 3/7 PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

502.367 ( ValorInicial )

\* CAMPO 4

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT27 ( #y: 27 para fila 4/7 PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

111.2561 ( ValorInicial )

\* CAMPO 5

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT36 ( #y: 36 para fila 5/7 PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

87.25948 ( ValorInicial )

\* CAMPO 6

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT45 ( #y: 45 para fila 6/7 PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

19.356 ( ValorInicial )

\* CAMPO 7

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT54 ( #y: 54 para fila 7/7 PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

36.335 ( ValorInicial )

BINT7 ( #Netiq )

BINT7 ( #Ncamp )

\* Message handler del formulario

' ::

\* Dibuja el título, el cual es un grob de tamaño 131x7

\* Sólo cuando la línea de edición no ocupe toda la pantalla

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se dibujará el título en la pantalla

\*\*\* Esto permite que camp y etiq puedan dibujarse en la zona del título

BINT1 #=casedrop ( F ) ( -> T // F )

TRUE

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT63 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

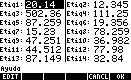
DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 ob3 ob4 ob5 ob6 ob7 T // F )

;

Ejemplo 12 DoInputForm

**Un formulario de entrada con 14 campos  
No se muestra el título.**

En este ejemplo se muestra un formulario de entrada con 14 campos **TEXTO** en dos columnas.

Las etiquetas tienen sus posiciones #x en las ubicaciones 0 y 67.

Los campos tienen sus posiciones #y en las ubicaciones 0, 9, 18, 27, 36, 45 y 54.

Para no mostrar el título, se usa el mensaje número 1 en el formulario.

Se ha colocado una línea vertical (grob de ancho 1) como etiqueta en la posición x=65, y=8

Desgraciadamente, en DoInputForm no está disponible toda la pantalla para mostrar etiquetas (sólo hasta la fila 46). Sin embargo, en este ejemplo podemos vencer este obstáculo usando el message handler número 4 para poder dibujar etiquetas en toda la pantalla y para que una etiqueta también pueda definirse como grob en DoInputForm

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME Do2COLFS ( -> ob1 ob2 ob3 ... ob14 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq1:" 0 2 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+2 )

"Etiq2:" 67 2 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+2 )

"Etiq3:" 0 11 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+2 )

"Etiq4:" 67 11 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+2 )

"Etiq5:" 0 20 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+2 )

"Etiq6:" 67 20 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+2 )

"Etiq7:" 0 29 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+2 )

"Etiq8:" 67 29 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+2 )

"Etiq9:" 0 38 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+2 )

"Eti10:" 67 38 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+2 )

"Eti11:" 0 47 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+2 )

"Eti12:" 67 47 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+2 )

"Eti13:" 0 56 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+2 )

"Eti14:" 67 56 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+2 )

BINT62 BINT1 MAKEGROB INVGROB 65 0 ( #xetiq=62 ) ( #yetiq=0 )

\* CAMPO 1

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT0 ( #y: 0 para fila 1/7 PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 2

'DROPFALSE ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT0 ( #y: 0 para fila 1/7 PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

12.345 ( ValorInicial )

\* CAMPO 3

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT9 ( #y: 9 para fila 2/7 PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

502.36 ( ValorInicial )

\* CAMPO 4

'DROPFALSE ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT9 ( #y: 9 para fila 2/7 PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

111.25 ( ValorInicial )

\* CAMPO 5

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT18 ( #y: 18 para fila 3/7 PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

87.259 ( ValorInicial )

\* CAMPO 6

'DROPFALSE ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT18 ( #y: 18 para fila 3/7 PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

19.356 ( ValorInicial )

\* CAMPO 7

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT27 ( #y: 27 para fila 4/7 PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

15.23 ( ValorInicial )

\* CAMPO 8

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT27 ( #y: 27 para fila 4/7 PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

78.259 ( ValorInicial )

\* CAMPO 9

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT36 ( #y: 36 para fila 5/7 PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

47.251 ( ValorInicial )

\* CAMPO 10

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT36 ( #y: 36 para fila 5/7 PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

36.982 ( ValorInicial )

\* CAMPO 11

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT45 ( #y: 45 para fila 6/7 PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

44.512 ( ValorInicial )

\* CAMPO 12

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT45 ( #y: 45 para fila 6/7 PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

77.98 ( ValorInicial )

\* CAMPO 13

'DROPFALSE ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT54 ( #y: 54 para fila 7/7 PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

87.149 ( ValorInicial )

\* CAMPO 14

'DROPFALSE ( MH del campo )

BINT93 ( #x: #xetiq + 4\*#ncetiq + 2 )

BINT54 ( #y: 54 para fila 7/7 PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

32.84 ( ValorInicial )

BINT15 ( #Netiq )

BINT14 ( #Ncamp )

\* Message handler del formulario

' ::

\* Dibuja el título, el cual es un grob de tamaño 131x7

\* Sólo cuando la línea de edición no ocupe toda la pantalla

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se dibujará el título en la pantalla

\*\*\* Esto permite que camp y etiq puedan dibujarse en la zona del título

BINT1 #=casedrop ( F ) ( -> T // F )

TRUE

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT63 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

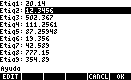
DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( -> ob1 ob2 ob3 ... ob14 T // F )

;

Ejemplo 13 DoInputForm

**Un formulario de entrada con 9 campos en Minifuente  
No se muestra el título.**

En este ejemplo se muestra un formulario de entrada con 9 campos **TEXTO** en una columna.

Las etiquetas tienen sus posiciones #x en la ubicación 0.

Los campos tienen sus posiciones #y en las ubicaciones 0, 7, 14, 21, 28, 35, 42, 49 y 56.

Las etiquetas tienen sus posiciones #y en una unidad más que las del campo correspondiente.

Se llama al mensaje número 9 en los campos que se desean mostrar con Minifuente.

Para no mostrar el título, se usa el mensaje número 1 en el formulario.

Se llama al mensaje número 4 en el formulario para poder mostrar las etiquetas en toda la pantalla.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME Do1ColMiniFS ( -> ob1 ob2 ... ob9 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq1:" 0 1 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq2:" 0 8 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq3:" 0 15 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq4:" 0 22 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq5:" 0 29 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq6:" 0 36 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq7:" 0 43 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq8:" 0 50 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq9:" 0 57 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+1 en MiniF )

\* CAMPO 1

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT0 ( #y: 0 para fila 1/9 MiniF PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 2

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT7 ( #y: 7 para fila 2/9 MiniF PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

12.3456 ( ValorInicial )

\* CAMPO 3

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT14 ( #y: 14 para fila 3/9 MiniF PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

502.367 ( ValorInicial )

\* CAMPO 4

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT21 ( #y: 21 para fila 4/9 MiniF PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

111.2561 ( ValorInicial )

\* CAMPO 5

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT28 ( #y: 28 para fila 5/9 MiniF PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

87.25948 ( ValorInicial )

\* CAMPO 6

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT35 ( #y: 35 para fila 6/9 MiniF PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

19.356 ( ValorInicial )

\* CAMPO 7

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT42 ( #y: 42 para fila 7/9 MiniF PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

42.589 ( ValorInicial )

\* CAMPO 8

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT49 ( #y: 49 para fila 8/9 MiniF PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

777.15 ( ValorInicial )

\* CAMPO 9

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT56 ( #y: 56 para fila 9/9 MiniF PANTALLA COMPLETA )

BINT106 ( #w: 131-#x para 1 columna )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

354.89 ( ValorInicial )

BINT9 ( #Netiq )

BINT9 ( #Ncamp )

\* Message handler del formulario

' ::

\* Dibuja el título, el cual es un grob de tamaño 131x7

\* Sólo cuando la línea de edición no ocupe toda la pantalla

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se dibujará el título en la pantalla

\*\*\* Esto permite que camp y etiq puedan dibujarse en la zona del título

BINT1 #=casedrop ( F ) ( -> T // F )

TRUE

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT64 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 ... ob9 T // F )

;

\* Message handler del campo:

NULLNAME DoMH9\_MiniFuente

::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Retorna un grob que representa al campo usando la minifuente

\*\*\* Respeta al parámetro Decompile del campo

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 045 ( #c $ ) ( Retorna $ del campo respetando Decompile )

OVER ( #c $ #c )

ROMPTR 0B0 0CA ( #c $ #b #h ) ( Retorna ancho y alto del campo )

ROTDUP ( #c #b #h $ $ )

LEN$ ( #c #b #h $ #car )

BINT4 ( #c #b #h $ #car #4 )

#\* ( #c #b #h $ #4·car )

4PICK ( #c #b #h $ #4·car #b )

DUPUNROT ( #c #b #h $ #b #4·car #b )

#< ( #c #b #h $ #b flag )

ITE

DROP ( #c #b #h $ )

:: ( #c #b #h $ #b )

BINT4 ( #c #b #h $ #b #4 )

#/ ( #c #b #h $ #residuo #b/4 )

SWAPDROP ( #c #b #h $ #b/4 )

1\_#1-SUB$ ( #c #b #h $' )

CHR\_... ( #c #b #h $' chr )

>T$ ( #c #b #h $'' )

;

( #c #b #h $'' )

$>grob ( #c #b #h grob )

3UNROLL ( #c grob #b #h )

SWAP ( #c grob #h #b )

MAKEGROB ( #c grob grobbxh )

DUPUNROT ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' )

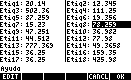
TRUE ( #c grobbxh' T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

Ejemplo 14 DoInputForm

**Un formulario de entrada con 18 campos en Minifuente  
No se muestra el título.**

En este ejemplo se muestra un formulario de entrada con 18 campos **TEXTO** en 2 columnas.

Las etiquetas tienen sus posiciones #x en las ubicaciones 0 y 67.

Los campos tienen sus posiciones #y en las ubicaciones 0, 7, 14, 21, 28, 35, 42, 49 y 56.

Las etiquetas tienen sus posiciones #y en una unidad más que las del campo correspondiente.

Se llama al mensaje número 9 en los campos que se desean mostrar con Minifuente.

Para no mostrar el título, se usa el mensaje número 1 en el formulario.

Se llama al mensaje número 4 en el formulario para poder mostrar las etiquetas en toda la pantalla y para permitir que las etiquetas también puedan definirse como grobs.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME Do2ColMiniFS ( -> ob1 ob2 ... ob16 T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

"Etiq1:" 0 1 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq2:" 67 1 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq3:" 0 8 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq4:" 67 8 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq5:" 0 15 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq6:" 67 15 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq7:" 0 22 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq8:" 67 22 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Etiq9:" 0 29 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti10:" 67 29 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti11:" 0 36 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti12:" 67 36 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti13:" 0 43 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti14:" 67 43 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti15:" 0 50 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti16:" 67 50 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti17:" 0 57 ( #xetiq=0 COL1/2 ) ( #yetiq=#ycampo+1 en MiniF )

"Eti18:" 67 57 ( #xetiq=67 COL2/2 ) ( #yetiq=#ycampo+1 en MiniF )

BINT62 BINT1 MAKEGROB INVGROB 65 0 ( #xetiq=62 ) ( #yetiq=0 )

\* CAMPO 1

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT0 ( #y: 0 para fila 1/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* CAMPO 2

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT0 ( #y: 0 para fila 1/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

12.345 ( ValorInicial )

\* CAMPO 3

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT7 ( #y: 7 para fila 2/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

502.36 ( ValorInicial )

\* CAMPO 4

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT7 ( #y: 7 para fila 2/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

111.25 ( ValorInicial )

\* CAMPO 5

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT14 ( #y: 14 para fila 3/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

87.259 ( ValorInicial )

\* CAMPO 6

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT14 ( #y: 14 para fila 3/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

19.356 ( ValorInicial )

\* CAMPO 7

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT21 ( #y: 21 para fila 4/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

15.23 ( ValorInicial )

\* CAMPO 8

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT21 ( #y: 21 para fila 4/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

78.259 ( ValorInicial )

\* CAMPO 9

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT28 ( #y: 28 para fila 5/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

47.251 ( ValorInicial )

\* CAMPO 10

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT28 ( #y: 28 para fila 5/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

36.982 ( ValorInicial )

\* CAMPO 11

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT35 ( #y: 35 para fila 6/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

44.512 ( ValorInicial )

\* CAMPO 12

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT35 ( #y: 35 para fila 6/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

77.98 ( ValorInicial )

\* CAMPO 13

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT42 ( #y: 42 para fila 7/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

777.369 ( ValorInicial )

\* CAMPO 14

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT42 ( #y: 42 para fila 7/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

49.3658 ( ValorInicial )

\* CAMPO 15

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT49 ( #y: 49 para fila 8/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

36.25 ( ValorInicial )

\* CAMPO 16

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT49 ( #y: 49 para fila 8/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

159.35 ( ValorInicial )

\* CAMPO 17

' DoMH9\_MiniFuente ( MH del campo )

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT56 ( #y: 56 para fila 9/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 64-#x para columna 1/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

36.25 ( ValorInicial )

\* CAMPO 18

' DoMH9\_MiniFuente ( MH del campo )

BINT92 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT56 ( #y: 56 para fila 9/9 MiniF PANTALLA COMPLETA )

BINT39 ( #w: 131-#x para columna 2/2 )

BINT7 ( #h: 7 para MiniFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

425.98 ( ValorInicial )

BINT19 ( #Netiq )

BINT18 ( #Ncamp )

\* Message handler del formulario

' ::

\* Dibuja el título, el cual es un grob de tamaño 131x7

\* Sólo cuando la línea de edición no ocupe toda la pantalla

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se dibujará el título en la pantalla

\*\*\* Esto permite que camp y etiq puedan dibujarse en la zona del título

BINT1 #=casedrop ( F ) ( -> T // F )

TRUE

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT64 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 ... ob16 T // F )

;

\* Message handler del campo:

NULLNAME DoMH9\_MiniFuente

::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Retorna un grob que representa al campo usando la minifuente

\*\*\* Respeta al parámetro Decompile del campo

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 045 ( #c $ ) ( Retorna $ del campo respetando Decompile )

OVER ( #c $ #c )

ROMPTR 0B0 0CA ( #c $ #b #h ) ( Retorna ancho y alto del campo )

ROTDUP ( #c #b #h $ $ )

LEN$ ( #c #b #h $ #car )

BINT4 ( #c #b #h $ #car #4 )

#\* ( #c #b #h $ #4·car )

4PICK ( #c #b #h $ #4·car #b )

DUPUNROT ( #c #b #h $ #b #4·car #b )

#< ( #c #b #h $ #b flag )

ITE

DROP ( #c #b #h $ )

:: ( #c #b #h $ #b )

BINT4 ( #c #b #h $ #b #4 )

#/ ( #c #b #h $ #residuo #b/4 )

SWAPDROP ( #c #b #h $ #b/4 )

1\_#1-SUB$ ( #c #b #h $' )

CHR\_... ( #c #b #h $' chr )

>T$ ( #c #b #h $'' )

;

( #c #b #h $'' )

$>grob ( #c #b #h grob )

3UNROLL ( #c grob #b #h )

SWAP ( #c grob #h #b )

MAKEGROB ( #c grob grobbxh )

DUPUNROT ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' )

TRUE ( #c grobbxh' T )

;

\* Fin del Message handler del campo:

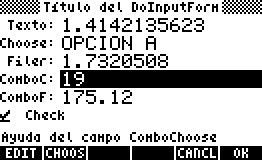
DROPFALSE

;

**Ejemplo 15 DoInputForm**

**Un formulario de entrada con los 6 tipos de campos en DoInputForm.**

**Usando toda la pantalla para mostrar etiquetas.**

En este ejemplo se muestra un formulario de entrada con los seis tipos de campos: TEXTO, CHOOSE, FILER, COMBOCHOOSE, COMBOFILER y CHECK.

Desgraciadamente, en DoInputForm no está disponible toda la pantalla para mostrar etiquetas (sólo hasta la fila 46). Sin embargo, en este ejemplo podemos vencer este obstáculo usando el message handler número 4 para poder dibujar etiquetas en toda la pantalla.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoInFo6Tipos ( -> ob1...obn T // F )

::

CK0 ( ) ( No se requieren argumentos )

\* Definiciones de las etiquetas

" Texto:" 0 10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Choose:" 0 19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

" Filer:" 0 28 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"ComboC:" 0 37 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"ComboF:" 0 46 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Check" 8 55 ( #xetiq=#xcampo+8 en CHK ) ( #yetiq=#ycampo+2 )

\* Campo 0: TEXTO

'DROPFALSE ( MH del campo )

BINT29 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT102 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda del campo Texto" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

32.15 ( ValorReset )

20.14 ( ValorInicial )

\* Campo 1: CHOOSE

'DROPFALSE ( MH del campo )

BINT29 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT102 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT17 ( Decompile: 16=1ºObjC 1=NoDesc ) ( $ #[ME] ) ( $ )

"Ayuda del campo Choose" ( Ayuda )

{ { "OPCION A" 11. }

{ "OPCION B" 12. }

{ "OPCION C" 13. }

} ( ChooseData: Items del browser 48 )

' :: INCOMPDROP " " SWAP DecompEdit &$ &$

; ( ChooseDecompile: Es el Converter del browser 48 )

OVER CARCOMP ( ValorReset: 1º obj de ChooseData en CHOOSE )

DUP ( ValorInicial: El mismo que ValorReset )

\* Campo 2: FILER

'DROPFALSE ( MH del campo )

BINT29 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT26 ( #y: 26 para fila 3/6 )

BINT102 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT20 ( #TipoDeCampo: FILER )

{ BINT0 BINT1 BINT6 } ( TiposPermitidos: R,C,id )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda del campo Filer" ( Ayuda )

MINUSONE ( ChooseData en FILER: F=NoCHK TiposP=TiposPC T=RetObj )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo FILER )

1.7320508 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* Campo 3: COMBOCHOOSE

'DROPFALSE ( MH del campo )

BINT29 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT35 ( #y: 35 para fila 4/6 )

BINT102 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT13 ( #TipoDeCampo: COMBOCHOOSE )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda del campo ComboChoose" ( Ayuda )

{ 12. 13. 14. 15. } ( ChooseData: Items del browser 48 )

3PICK ( ChooseDecompile: El mismo valor que Decompile )

18. ( ValorReset )

19. ( ValorInicial )

\* Campo 4: COMBOFILER

'DROPFALSE ( MH del campo )

BINT29 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT44 ( #y: 44 para fila 5/6 )

BINT102 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT21 ( #TipoDeCampo: COMBOFILER )

MINUSONE ( TiposPermitidos: Todos los tipos existentes )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda del campo ComboFiler" ( Ayuda )

MINUSONE ( ChooseData en FILER: F=NoCHK TiposP=TiposPC T=RetObj )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo COMBOFILER )

175.12 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* Campo 5: CHECKBOX

'DROPFALSE ( MH del campo )

BINT0 ( #x: mayor o igual a 0 en CHECKBOX )

BINT53 ( #y: 53 para fila 6/6 )

BINT6 ( #w: 6 siempre en campo CHECKBOX )

BINT9 ( #h: 9 siempre en campo CHECKBOX )

BINT32 ( #TipoDeCampo: CHECKBOX )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHECKBOX )

MINUSONE ( Decompile: MINUSONE siempre en campo CHECKBOX )

"Ayuda del campo Check" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo CHECKBOX )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo CHECKBOX )

FALSE ( ValorReset: campo CHECKBOX desactivado )

FALSE ( ValorInicial: campo CHECKBOX desactivado )

BINT6 ( #Netiq )

BINT6 ( #Ncamp )

\* Message handler del formulario

' ::

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT63 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 ob3 ob4 ob5 ob6 T // F )

;

**Ejemplo 16 DoInputForm**

**Un campo COMBOCHOOSE que muestre los ítems con una breve explicación (Usando Browser 49)**

En este ejemplo se muestra un campo COMBOCHOOSE.

En este campo puedes escribir un valor directamente con el teclado o también escoger un valor desde una lista accesible mediante la tecla de menú CHOOS.

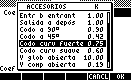
El parámetro ChooseData del campo COMBOCHOOSE tiene como elementos a listas de la forma:

{ $ ob }

Donde ob es un número real que se fijará como el valor del campo actual, si uno escoge esa opción.

Se llama al mensaje 26 para mostrar cada una de las opciones del parámetro ChooseData.

En este ejemplo se usa el browser 49.

** **

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoInFoComboB49 ( -> % T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

" Coef:" 0 10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: COMBOCHOOSE

\* Message handler del campo

' ::

\* Al presionar tecla CHOOS en campo CHOOSE FILER COMBOCHOOSE COMBOFILER

\* Debe mostrar opciones, guardar nuevo valor del campo y mostrarlo.

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* En el campo COMBOCHOOSE muestra items con breve explicación

\*\*\* Usa el browser 49

BINT26 #=casedrop ( C ) ( -> T // F )

:: ( )

5GETLAM ( #campo ) ( Número del campo actual )

ROMPTR 0B0 0C5 ( {}ChooseDATA )

INNERCOMP ( meta )

dup\_ ( meta meta )

DUP ZERO\_DO (DO)

ROLL

TWONTHCOMPDROP\_

ISTOP@

LOOP

{}N ( meta {%} )

5GETLAM ( meta {%} #campo )

ROMPTR gFldVal ( meta {%} %ValorCampo )

EQUALPOSCOMP ( meta #pos/#0 )

DUP#0=IT

DROPONE

#1- ( meta #i )

" ACCESORIOS K" ( meta #i $ )

SWAP ( meta $ #i )

FLASHPTR Choose2\_ ( {$ %} T // F ) ( BROWSER 49 )

IT

:: ( {$ %} )

TWONTHCOMPDROP\_ ( % )

5GETLAM ( % #c )

ROMPTR sFldVal ( ) ( Cambia el valor de un campo )

;

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT106 ( #w:131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT13 ( #TipoDeCampo: COMBOCHOOSE )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Coef de pérdidas locales" ( Ayuda )

{ { "Entrada b agud 0.50" %.5 }

{ "Entr b acampan 0.04" % .04 }

{ "Entr b entrant 1.00" %1 }

{ "Salida a depÓs 1.00" %1 }

{ "Codo a 90° 0.90" % .9 }

{ "Codo a 45° 0.42" % .42 }

{ "Codo curv fuerte 0.75" % .75 }

{ "Codo curv suave 0.60" % .6 }

{ "V glob abierta 10.00" %10 }

{ "V comp abierta 0.19" % .19 }

{ "V check abierta 2.50" % 2.5 }

} ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: No es necesario pues se llama al mensaje 26 )

12. ( ValorReset )

13. ( ValorInicial )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

'DROPFALSE ( MH del DoInputForm )

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( % T // F )

;

**Ejemplo 17 DoInputForm**

**Un campo COMBOCHOOSE que muestre los ítems con una breve explicación (Usando Browser 48)**

En este ejemplo se muestra un campo COMBOCHOOSE.

En este campo puedes escribir un valor directamente con el teclado o también escoger un valor desde una lista accesible mediante la tecla de menú CHOOS.

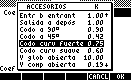
El parámetro ChooseData del campo COMBOCHOOSE tiene como elementos a listas de la forma:

{ $ ob }

Donde ob es un número real que se fijará como el valor del campo actual, si uno escoge esa opción.

Se llama al mensaje 26 para mostrar cada una de las opciones del parámetro ChooseData.

En este ejemplo se usa el browser 48.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoInFoComboB48 ( -> % T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

" Coef:" 0 10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: COMBOCHOOSE

\* Message handler del campo

' ::

\* Al presionar tecla CHOOS en campo CHOOSE FILER COMBOCHOOSE COMBOFILER

\* Debe mostrar opciones, guardar nuevo valor del campo y mostrarlo.

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* En el campo COMBOCHOOSE muestra items con breve explicación

\*\*\* Usa el browser 48

BINT26 #=casedrop ( C ) ( -> T // F )

:: ( )

'DROPFALSE ( MH )

" ACCESORIOS K" ( MH $ )

BINT17 ( MH $ conv ) ( 16=1ºObjC 1=NoDesc )

5GETLAM ( MH $ conv #campo )

ROMPTR 0B0 0C5 ( MH $ conv {}ChooseDATA )

DUPINCOMP ( MH $ conv {}ChooseDATA meta )

DUP ZERO\_DO (DO)

ROLL

TWONTHCOMPDROP\_

ISTOP@

LOOP

{}N ( MH $ conv {}ChooseDATA {ob} )

5GETLAM ( MH $ conv {}ChooseDATA {ob} #campo )

ROMPTR gFldVal ( MH $ conv {}ChooseDATA {ob} %ValorCampo )

EQUALPOSCOMP ( MH $ conv {}ChooseDATA #pos/#0 )

DUP#0=IT

DROPONE

ROMPTR Choose ( {$ %} T // F ) ( BROWSER 49 )

IT

:: ( {$ %} )

TWONTHCOMPDROP\_ ( % )

5GETLAM ( % #c )

ROMPTR sFldVal ( ) ( Cambia el valor de un campo )

;

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT25 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT106 ( #w:131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT13 ( #TipoDeCampo: COMBOCHOOSE )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Coef de pérdidas locales" ( Ayuda )

{ { "Entrada b agud 0.50" %.5 }

{ "Entr b acampan 0.04" % .04 }

{ "Entr b entrant 1.00" %1 }

{ "Salida a depÓs 1.00" %1 }

{ "Codo a 90° 0.90" % .9 }

{ "Codo a 45° 0.42" % .42 }

{ "Codo curv fuerte 0.75" % .75 }

{ "Codo curv suave 0.60" % .6 }

{ "V glob abierta 10.00" %10 }

{ "V comp abierta 0.19" % .19 }

{ "V check abierta 2.50" % 2.5 }

} ( ChooseData: Lista )

MINUSONE ( ChooseDecompile: No es necesario pues se llama al mensaje 26 )

12. ( ValorReset )

13. ( ValorInicial )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

'DROPFALSE ( MH del DoInputForm )

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( % T // F )

;

**Ejemplo 18 DoInputForm**

**Actualizar el valor de un campo al cambiar el valor de otro campo.**

Para hacer esto debes usar el mensaje número 47 en el message handler de un campo.

Este mensaje es llamado cuando el valor del campo cambia.

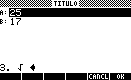
Además debes usar el NULLNAME DoSetFieldValue que permite cambiar el valor de cualquier campo. Entonces:

1) Poner el mensaje nº 47 en el campo cuyos cambios afectarán el valor de otros campos.

2) En este debemos obtener el valor(es) que queremos asignar al otro(s) campo(s) y luego usar el comando NULLNAME DoSetFieldValue.

A continuación un ejemplo sencillo con dos campos y dos etiquetas con su respectiva explicación que puedes copiar y pegar en el editor de Debug4x.

Si se actualiza el valor del primer campo, el segundo campo se actualizará de manera automática, fijándose su valor como la décima parte del valor del primer campo.

\* Llama al NULLNAME DoSetFieldValue

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoActualizaVal ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"B:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: A

\* ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL. CUANDO EL VALOR DE

\* ESTE CAMPO CAMBIA, EL VALOR DEL OTRO CAMPO TAMBIÉN CAMBIARÁ.

\* Message handler del campo

' ::

\* Llamado inmediatamente después de que el campo ha cambiado su valor

\* Puedes hacer cualquier cosa aquí. No hay diferencia si retornas T o F

BINT47 #=casedrop ( C ) ( -> flag )

:: ( )

5GETLAM ( #c )

ROMPTR gFldVal ( valor )

%10 %/ ( valor/10 )

BINT2 ( valor/10 #2 )

DoSetFieldValue ( )

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Escribe un número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

25. ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO Nº #2

\* ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* CUANDO EL VALOR DEL OTRO CAMPO CAMBIE, EL

\* VALOR DE ESTE CAMBIA. SE ACTUALIZA A LA DÉCIMA PARTE DEL OTRO

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Escribe un número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

17. ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

'DROPFALSE ( MH del DoInputForm )

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 T // F )

;

\* Cambia el valor de un campo.

\* Muestra el nuevo valor en pantalla.

\* Llama al mensaje 47 del campo que se está cambiando.

NULLNAME DoSetFieldValue ( valor #c -> )

:: ( valor #c )

DUPUNROT ( #c valor #c )

ROMPTR sFldVal ( #c ) ( cambia valor del campo )

DUP ( #c #c )

FLASHPTR IFEDispField ( #c ) ( muestra valor del campo )

ROMPTR 0B0 041 ( ) ( llama al mensaje 47 del campo )

;

**Ejemplo 19 DoInputForm**

**Bloquear un campo.**

Para hacer que un campo esté bloqueado en un formulario de entrada puedes usar el mensaje número 55 el cual es llamado cuando un campo está a punto de recibir el enfoque para decidir si ese campo merece recibir el enfoque o no.

En este ejemplo hay tres campos (1, 2 y 3). Vamos a bloquear el campo 2. Para esto usamos el message handler número 55 en el campo que vamos a bloquear.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoInBlockField ( -> ob1 ob2 ob3 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"B:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"C:" BINT0 BINT28 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"A: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

12.16 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* Message handler del campo

' ::

\* Es llamado cuando se intenta que el campo tenga el enfoque.

\* Si colocas FALSE TRUE, el campo no tendrá el enfoque.

BINT55 #=casedrop ( C ) ( -> T/F T // F )

:: ( )

FalseTrue ( F T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"B: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

98.57 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 3: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT26 ( #y: 27 para fila 3/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"C: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

47.29 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT3 ( #Netiq )

BINT3 ( #Ncamp )

'DROPFALSE ( MH del DoInputForm )

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 ob3 T // F )

;

**Ejemplo 20 DoInputForm**

**Bloquear campos con una condición.**

**Actualizar el valor de un campo al cambiar el valor de otro campo.**

En este ejemplo se verá como bloquear campos sólo si se cumple una condición.

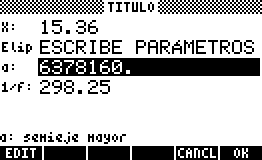
Aquí hay 4 campos (1, 2, 3 y 4)

El campo 1 contiene a un campo TEXTO y no es bloqueado.

El campo 2 es un campo CHOOSE.

Los campos 3 y 4 son campos TEXTO y serán bloqueados cuando en el campo 2 la opción escogida no sea la opción “ESCRIBE PARÁMETROS”. Para esto se usa el mensaje 55 en los campos que serán bloqueados (campos 3 y 4).

Además, cada vez que se escoge en el campo CHOOSE a una opción que no sea “ESCRIBE PARÁMETROS”, los campos 3 y 4 se actualizarán con los valores correspondientes al elipsoide escogido. Además se usa el mensaje 17 (el cual es llamado al inicio) en el message handler del formulario para que dicha actualización también ocurra al inicio del formulario.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoBlockCond ( -> ob1 ob2 ob3 ob4 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

" X:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Elip:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

" a:" BINT0 BINT28 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

" 1/f:" BINT0 BINT37 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT21 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT110 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"X: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

12.16 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO CHOOSE

\* Message handler del campo

' ::

\* Llamado inmediatamente después de que el campo ha cambiado su valor

\* Puedes hacer cualquier cosa aquí. No hay diferencia si retornas T o F

BINT47 #=casedrop ( C ) ( -> flag )

:: ( )

BINT2 ( #campo )

ROMPTR gFldVal ( {$ % %'} ) ( Retorna Valor )

BINT2 ( {$ % %'} #campo )

ROMPTR 0B0 0C5 ( {$ % %'} {{}} ) ( Retorna Choosedata )

FLASHPTR LASTCOMP ( {$ % %'} {$ % %'} )

OVER ( {$ % %'} {$ % %'} {$ % %'} )

EQUALNOT ( {$ % %'} flag )

ITE

:: DUP ( {$ % %'} )

TWONTHCOMPDROP\_ ( % )

BINT3 ( {$ % %'} % #c )

DoSetFieldValue ( {$ % %'} )

FLASHPTR LASTCOMP ( % )

BINT4 ( % #c )

DoSetFieldValue ( )

;

DROP

( )

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT21 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT110 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT17 ( Decompile: 16=1ºObjC 1=NoDesc ) ( $ #[ME] ) ( $ )

"Escoge un elipsoide" ( Ayuda )

{ { "SouthAmerican 1969" % 6378160. % 298.25 }

{ "WGS 1960" % 6378165. % 298.3 }

{ "WGS 1966" % 6378145. % 298.25 }

{ "WGS 1972" % 6378135. % 298.26 }

{ "WGS 1984" % 6378137. % 298.257223563 }

{ "ESCRIBE PARÁMETROS" %-1 %-1 }

} ( ChooseData: Items del browser 48 )

BINT17 ( ChooseDecompile: 16=1ºObjC 1=NoDesc ) ( $ #[ME] ) ( $ )

OVER BINT5 NTHCOMPDROP ( ValorReset: 5º obj de ChooseData en CHOOSE )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 3: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* CON EL MESSAGE HANDLER 55, PODEMOS BLOQUEAR ESTE CAMPO.

\* Message handler del campo

' ::

\* Es llamado cuando se intenta que el campo tenga el enfoque.

\* Si colocas FALSE TRUE, el campo no tendrá el enfoque.

BINT55 #=casedrop ( C ) ( -> T/F T // F )

:: ( )

BINT2 ( #campo )

ROMPTR gFldVal ( {$ % %'} ) ( Retorna Valor )

BINT2 ( {$ % %'} #campo )

ROMPTR 0B0 0C5 ( {$ % %'} {{}} ) ( Retorna Choosedata )

FLASHPTR LASTCOMP ( {$ % %'} {$ % %'} )

EQUAL ( flag' )

TRUE ( flag' T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT21 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT26 ( #y: 27 para fila 3/6 )

BINT110 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"a: semieje mayor" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

98.57 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 4: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* CON EL MESSAGE HANDLER 55, PODEMOS BLOQUEAR ESTE CAMPO.

\* Message handler del campo

' ::

\* Es llamado cuando se intenta que el campo tenga el enfoque.

\* Si colocas FALSE TRUE, el campo no tendrá el enfoque.

BINT55 #=casedrop ( C ) ( -> T/F T // F )

:: ( )

BINT2 ( #campo )

ROMPTR gFldVal ( {$ % %'} ) ( Retorna Valor )

BINT2 ( {$ % %'} #campo )

ROMPTR 0B0 0C5 ( {$ % %'} {{}} ) ( Retorna Choosedata )

FLASHPTR LASTCOMP ( {$ % %'} {$ % %'} )

EQUAL ( flag' )

TRUE ( flag' T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT21 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT35 ( #y: 36 para fila 4/6 )

BINT110 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"f: achatamiento" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

47.29 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT4 ( #Netiq )

BINT4 ( #Ncamp )

\* Message handler del formulario

' ::

\* Es llamado al inicio. Cuando ya estan fijados los lams y antes del POL.

\* Puedes hacer cualquier cosa aquí. No hay diferencia si retornas T o F

BINT17 #=casedrop ( F ) ( -> flag )

:: ( )

BINT2 ( #2 )

ROMPTR 0B0 041 ( ) ( ejecuta el mensaje 47 del campo 2 )

TRUE ( flag' T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 ob3 ob4 T // F )

;

\* Cambia el valor de un campo.

\* Muestra el nuevo valor en pantalla.

\* Llama al mensaje 47 del campo que se está cambiando.

NULLNAME DoSetFieldValue ( valor #c -> )

:: ( valor #c )

DUPUNROT ( #c valor #c )

ROMPTR sFldVal ( #c ) ( cambia valor del campo )

DUP ( #c #c )

FLASHPTR IFEDispField ( #c ) ( muestra valor del campo )

ROMPTR 0B0 041 ( ) ( llama al mensaje 47 del campo )

;

**Ejemplo 21 DoInputForm**

**Hacer aparecer o desaparecer campos y etiquetas.**

Si en el primer campo hay un número positivo, entonces el segundo campo es invisible.

Si en el primer campo hay un número no positivo, entonces el segundo campo será visible.

El mensaje 47 es llamado inmediatamente después de que un campo ha cambiado su valor.

Se llama al mensaje 47 en el campo cuyos cambios afectarán la visibilidad o no de otros campos.

Se usará el mensaje 47 en el campo 1. Dentro de este mensaje 47 se llama al comando ROMPTR 0B0 0B7 y al comando ROMPTR 0B0 0D3

El comando ROMPTR 0B0 0B7 guarda un nuevo valor en el contenido de una etiqueta y se usa en este ejemplo para volver visible o invisible a la etiqueta 2.

El comando ROMPTR 0B0 0D3 guarda un nuevo objeto en el parámetro Converter de un campo y se usa en este ejemplo para volver visible o invisible al campo 2.

Finalmente, dentro del mensaje 47 se debe usar el comando ROMPTR 0B0 00C el cual redibuja toda la zona de etiquetas y de campos.

Además, para bloquear el campo 2, se usa el mensaje 55 en el campo 2, con el cual se evitará que dicho campo reciba el enfoque bajo ciertas condiciones.

A continuación, un ejemplo sencillo con dos campos y dos etiquetas con su respectiva explicación que puedes copiar y pegar en el editor de Debug4x.

Si en el primer campo hay un número positivo, entonces el segundo campo es invisible.

Si en el primer campo hay un número no positivo, entonces el segundo campo será visible.

Además se usa el mensaje 17 (el cual es llamado al inicio) en el message handler del formulario para que dicha actualización también ocurra al inicio del formulario.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoVisibleONo ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"B:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* SI EL VALOR DE ESTE CAMPO ES MAYOR QUE CERO, DESAPARECEN

\* LA ETIQUETA Y EL CAMPO 2

\* DE LO CONTRARIO, LA ETIQUETA Y EL CAMPO #2 SE VERÁN

\* Message handler del campo

' ::

\* Llamado inmediatamente después de que el campo ha cambiado su valor

\* Puedes hacer cualquier cosa aquí. No hay diferencia si retornas T o F

BINT47 #=casedrop ( C ) ( -> flag )

:: ( )

BINT1 ( #c )

ROMPTR gFldVal ( %Valor )

%0> ( flag )

ITE

:: NULL$ ( $ )

BINT2 ( $ #etiq )

ROMPTR 0B0 0B7 ( ) ( Cambia el contenido de la etiqueta )

' DROPNULL$ ( prog )

BINT2 ( #campo )

ROMPTR 0B0 0D3 ( ) ( Cambia el Decompile del campo )

;

:: "B:" ( $ )

BINT2 ( $ #etiq )

ROMPTR 0B0 0B7 ( ) ( Cambia el contenido de la etiqueta )

BINT4 ( prog )

BINT2 ( #campo )

ROMPTR 0B0 0D3 ( ) ( Cambia el Decompile del campo )

;

ROMPTR 0B0 00C ( ) ( redibuja zona de etiquetas y campos )

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Escribe un número real" ( "Ayuda" )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%5 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* PARA QUE PUEDA SER VISIBLE ESTE CAMPO, EL CAMPO #1 DEBE

\* CONTENER UN NÚMERO NEGATIVO

\* Message handler del campo

' ::

\* Es llamado cuando se intenta que el campo tenga el enfoque.

\* Si colocas FALSE TRUE, el campo no tendrá el enfoque.

BINT55 #=casedrop ( C ) ( -> T/F T // F )

:: ( )

BINT1 ( #c )

ROMPTR gFldVal ( %Valor )

%0> ( flag )

NOT ( flag' )

TRUE ( flag' T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Escribe un número real" ( "Ayuda" )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%17 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT64 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Es llamado al inicio. Cuando ya estan fijados los lams y antes del POL.

\* Puedes hacer cualquier cosa aquí. No hay diferencia si retornas T o F

BINT17 #=casedrop ( F ) ( -> flag )

:: ( )

BINT1 ( #campo )

ROMPTR 0B0 041 ( ) ( ejecuta el mensaje 47 del campo 1 )

TRUE ( flag' T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 T // F )

;

**Ejemplo 22 DoInputForm**

**Desaparecer y hacer aparecer campos y etiquetas en el mismo lugar en que estaban otros campos y etiquetas.**

Problema:

Si tengo una etiqueta con su campo CHOOSE, dentro de CHOOSE tengo tres opciones

{ {"Uno"} {"Dos"} {"Tres"} }

Cuando escoja “Uno”, no aparezca ninguna ETIQUETA/CAMPO.

Cuando escoja “Dos”, aparezca la etiqueta\_2 y campo\_2.

Cuando escoja “Tres”, aparezca la etiqueta\_3 y campo\_3 (en el mismo lugar que etiqueta\_2/campo\_2).

Solución:

El mensaje 47 es llamado inmediatamente después de que un campo ha cambiado su valor.

Se llama al mensaje 47 en el campo cuyos cambios afectarán la visibilidad o no de otros campos.

Se usará el mensaje 47 en el campo 1. Dentro de este mensaje 47 se llama al comando ROMPTR 0B0 0B7 y al comando ROMPTR 0B0 0D3

El comando ROMPTR 0B0 0B7 guarda un nuevo valor en el contenido de una etiqueta y se usa en este ejemplo para volver visible o invisible a las etiquetas 2 y 3.

El comando ROMPTR 0B0 0D3 guarda un nuevo objeto en el parámetro Converter de un campo y se usa en este ejemplo para volver visible o invisible a los campo 2 y 3.

Finalmente, dentro del mensaje 47 se debe usar el comando ROMPTR 0B0 00C el cual redibuja toda la zona de etiquetas y de campos.

Además, para bloquear el campo 2, se usa el mensaje 55 en los campos 2 y 3, con el cual se evitará que dichos campos reciban el enfoque bajo ciertas condiciones.

A continuación, un ejemplo sencillo con 3 campos y 3 etiquetas con su respectiva explicación que puedes copiar y pegar en el editor de Debug4x.

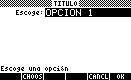
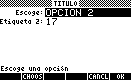
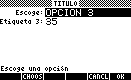
Si en el primer campo se escoge la opción 1, entonces los campos 2 y 3 y sus etiquetas correspondientes serán invisibles.

Si en el primer campo se escoge la opción 2, entonces el campo 2 y su etiqueta correspondiente serán visibles y el campo 3 y su etiqueta correspondiente serán invisibles.

Si en el primer campo se escoge la opción 3, entonces el campo 3 y su etiqueta correspondiente serán visibles y el campo 2 y su etiqueta correspondiente serán invisibles.

Además se usa el mensaje 17 (el cual es llamado al inicio) en el message handler del formulario para que dicha actualización también ocurra al inicio del formulario.

A continuación el código con su respectiva explicación que puedes copiar y pegar en el editor de Debug4x

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoVisibleEncima ( -> ob1 ob2 ob3 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

\* ETIQUETA 1

" Escoge:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* ETIQUETA 2

"Etiqueta 2:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* ETIQUETA 3

"Etiqueta 3:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 0: ES UN CAMPO CHOOSE.

\* CON LA OPCION 1, SE HACEN INVISIBLES LAS ETIQUETAS Y LOS CAMPOS 1 Y 2

\* CON LA OPCION 2, SE HACE VISIBLES LA ETIQUETA Y EL CAMPO 1

\* CON LA OPCION 3, SE HACE VISIBLES LA ETIQUETA Y EL CAMPO 2

\* Message handler del campo

' ::

\* Llamado inmediatamente después de que el campo ha cambiado su valor

\* Puedes hacer cualquier cosa aquí. No hay diferencia si retornas T o F

BINT47 #=casedrop ( C ) ( -> flag )

:: ( )

BINT1 ( #campo )

ROMPTR gFldVal ( {$ #} )

TWONTHCOMPDROP\_ ( # )

:: BINT1 #=casedrop

:: NULL$

BINT2

ROMPTR 0B0 0B7 ( ) ( Cambia el contenido de la etiqueta )

' DROPNULL$

BINT2

ROMPTR 0B0 0D3 ( ) ( Cambia el Decompile del campo )

NULL$

BINT3

ROMPTR 0B0 0B7 ( ) ( Cambia el contenido de la etiqueta )

' DROPNULL$

BINT3

ROMPTR 0B0 0D3 ( ) ( Cambia el Decompile del campo )

ROMPTR 0B0 00C ( ) ( redibuja zona de etiquetas y campos )

;

BINT2 #=casedrop

:: "Etiqueta 2:"

BINT2

ROMPTR 0B0 0B7 ( ) ( Cambia el contenido de la etiqueta )

BINT4

BINT2

ROMPTR 0B0 0D3 ( ) ( Cambia el Decompile del campo )

ROMPTR 0B0 00C ( ) ( redibuja zona de etiquetas y campos )

BINT2

FLASHPTR IFEDispField ( ) ( redibuja campo 2 )

;

BINT3 #=casedrop

:: "Etiqueta 3:"

BINT3

ROMPTR 0B0 0B7 ( ) ( Cambia el contenido de la etiqueta )

BINT4

BINT3

ROMPTR 0B0 0D3 ( ) ( Cambia el Decompile del campo )

ROMPTR 0B0 00C ( ) ( redibuja zona de etiquetas y campos )

BINT3

FLASHPTR IFEDispField ( ) ( redibuja campo 3 )

;

;

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT45 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT86 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT12 ( #TipoDeCampo: CHOOSE )

MINUSONE ( TiposPermitidos: MINUSONE siempre en campo CHOOSE )

BINT17 ( Decompile: 16=1ºObjC 1=NoDesc ) ( $ #[ME] ) ( $ )

"Escoge una opción" ( Ayuda )

{ { "OPCION 1" BINT1 }

{ "OPCION 2" BINT2 }

{ "OPCION 3" BINT3 }

} ( ChooseData: Items del browser 48 )

BINT17 ( ChooseDecompile: 16=1ºObjC 1=NoDesc ) ( $ #[ME] ) ( $ )

OVER CARCOMP ( ValorReset: 1º obj de ChooseData en CHOOSE )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* PARA QUE PUEDA SER VISIBLE ESTE CAMPO, EL CAMPO 1 DEBE CONTENER LA OPCION 1

\* Message handler del campo

' ::

\* Es llamado cuando se intenta que el campo tenga el enfoque.

\* Si colocas FALSE TRUE, el campo no tendrá el enfoque.

BINT55 #=casedrop ( C ) ( -> T/F T // F )

:: ( )

BINT1 ( #campo )

ROMPTR gFldVal ( {$ #} )

TWONTHCOMPDROP\_ ( # )

#2= ( flag )

TRUE ( flag T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT45 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT86 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Campo 1: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%17 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 3: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* PARA QUE PUEDA SER VISIBLE ESTE CAMPO, EL CAMPO 1 DEBE CONTENER LA OPCION 2

\* Message handler del campo

' ::

\* Es llamado cuando se intenta que el campo tenga el enfoque.

\* Si colocas FALSE TRUE, el campo no tendrá el enfoque.

BINT55 #=casedrop ( C ) ( -> T/F T // F )

:: ( )

BINT1 ( #campo )

ROMPTR gFldVal ( {$ #} )

TWONTHCOMPDROP\_ ( # )

#3= ( flag )

TRUE ( flag T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT45 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT86 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Campo 2: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%35\_ ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT3 ( #Netiq )

BINT3 ( #Ncamp )

\* Message handler del formulario

' ::

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT64 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Es llamado al inicio. Cuando ya estan fijados los lams y antes del POL.

\* Puedes hacer cualquier cosa aquí. No hay diferencia si retornas T o F

BINT17 #=casedrop ( F ) ( -> flag )

:: ( )

BINT1 ( #campo )

ROMPTR 0B0 041 ( ) ( ejecuta el mensaje 47 del campo 1 )

TRUE ( flag' T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 ob3 ob4 T // F )

;

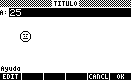
**Ejemplo 23 DoInputForm**

**Insertar un grob en un formulario de entrada**

Ahora veremos como insertar un grob en un formulario de entrada.

Lo insertamos como una etiqueta más, pero tendremos que usar el mensaje número 4, pues una etiqueta en DoInputForm normalmente no aceptaría grobs.

A continuación un ejemplo sencillo con dos etiquetas y un campo que puedes copiar y pegar en el editor de Debug4x.



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoPoneGrob ( -> ob T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

\* ETIQUETA 1: ES UNA CADENA

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* ETIQUETA 2: ES UN GROB

GROB 00036 B0000B00008F00401020209D401040154010409F40202040108F00

BINT20 BINT31

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%25 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT2 ( #Netiq )

BINT1 ( #Ncamp )

\* Message handler del formulario

' ::

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT64 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

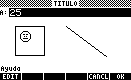
DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob T // F )

;

El siguiente código hace lo mismo que el programa anterior. Pero aquí el grob no lo ponemos como una etiqueta más. En vez de eso, dibujamos el grob (y cualquier otro grob que queramos) directamente en la pantalla, al llamar al mensaje 4.

Además en este ejemplo también se han dibujado un rectángulo y una línea recta.

A continuación un ejemplo sencillo con una etiqueta y un campo que puedes copiar y pegar en el editor de Debug4x.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoPoneGrob ( -> ob T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

\* ETIQUETA 1: ES UNA CADENA

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%25 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

\* Message handler del formulario

' ::

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT64 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

GROB 00036 B0000B00008F00401020209D401040154010409F40202040108F00

OVER ( HARDBUFF grob HARDBUFF )

BINT20 ( HARDBUFF grob HARDBUFF #Xetiq )

BINT31 ( HARDBUFF grob HARDBUFF #Xetiq #Yetiq )

GROB! ( HARDBUFF ) ( Reempl grob1 en grob2 )

BINT15 BINT26

BINT45 BINT56 ( HARDBUFF #x1 #y1 #x2 #y2 )

LBoxB ( HARDBUFF )

BINT66 BINT26

BINT106 BINT56 ( HARDBUFF #x1 #y1 #x2 #y2 )

LineB ( HARDBUFF )

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob T // F )

;

**Ejemplo 24 DoInputForm**

**Dar formato a las etiquetas.**

En este ejemplo damos formatos a las etiquetas del formulario de entrada.

Se llama al mensaje 4 para permitir que las etiquetas también sean grobs (y no sean sólo cadenas o bints).

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoEtiquetEstilo ( -> ob T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

" Negrita:" $->NEGRITA $>GROB BINT0 BINT9

" Cursiva:" $->CURSIVA $>GROB BINT0 BINT18

"Subrayado:" $->SUBRAYADA $>GROB BINT0 BINT27

" Inversa:" $->INVERSA $>GROB BINT0 BINT36

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT62 ( #x: #xetiq + 6\*#ncetiq + 2 ) ( Etiquetas en SysFont )

BINT8 ( #y: 8 para fila 1/6 )

BINT69 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%25 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT4 ( #Netiq )

BINT1 ( #Ncamp )

\* Message handler del formulario

' ::

\* Dibuja la pantalla en la región de los campos y las etiquetas

\* Si hay línea de edic con más de 1 renglón, sólo dibuja ayuda arriba

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Ya no se usa un grob de etiquetas

\*\*\* Los campos y etiquetas son dibujados directamente en la pantalla

\*\*\* Las etiquetas pueden dibujarse en toda la pantalla

\*\*\* Se pueden usar grobs en la definición de las etiquetas

\*\*\* Las etiq se dibujarán de nuevo, cada vez que se redibuje la pant

\*\*\* Líneas rectang círc grobs se pueden dibujar direct en la pantalla

\*\*\* Los mensajes 5 y 6 no serán llamados

BINT4 #=casedrop ( F ) ( -> T // F )

:: ( )

EDITPARTS ( # )

BINT1 ( # #1 )

#> ( flag )

case

:: ROMPTR 0B0 00D ( ) ( Dibuja ayuda en la parte superior )

TRUE ( T )

;

( )

HARDBUFF ( HARDBUFF )

BINT0 ( HARDBUFF 0 )

BINT7 ( HARDBUFF 0 7 )

BINT131 ( HARDBUFF 0 7 131 )

BINT64 ( HARDBUFF 0 7 131 63 )

GROB!ZERO ( HARDBUFF ) ( Limpia una región del grob )

\* En esta zona se pueden dibujar líneas, rectángulos, circulos, grobs

\* directamente en la pantalla

DROP ( )

15GETLAM ( #Netiq )

#1+\_ONE\_DO (DO)

INDEX@ ( #etiq )

DUP ( #etiq #etiq )

ROMPTR 0B0 0B3 ( #etiq $etiq/grob ) ( Contenido de etiq )

DUPTYPECSTR? ( #etiq $etiq/grob flag )

IT $>grob

( #etiq grob )

HARDBUFF ( #etiq grob HARDBUFF )

ROT ( grob HARDBUFF #etiq )

ROMPTR 0B0 0B6 ( grob HARDBUFF #X #Y ) ( Coord de etiq )

GROB! ( ) ( Reempl grob1 en grob2 ) ( Dibuja etiq )

LOOP

( )

14GETLAM ( #Ncamp )

#1+\_ONE\_DO (DO)

INDEX@ ( #c )

KEYINBUFFER?

ATTN?

OR ( #c flag )

ITE

ROMPTR 0B0 0A4 ( ) ( Pone flag del campo como F en LAM9 )

FLASHPTR IFEDispField ( ) ( Dibuja el campo )

( )

LOOP

( )

TRUE ( T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob T // F )

;

NULLNAME $->NEGRITA ( $ -> $ en negrita )

:: "\13\01\13" SWAPOVER &$ &$ ;

NULLNAME $->CURSIVA ( $ -> $ en cursiva )

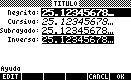
:: "\13\02\13" SWAPOVER &$ &$ ;

NULLNAME $->SUBRAYADA ( $ -> $ subrayada )

:: "\13\03\13" SWAPOVER &$ &$ ;

NULLNAME $->INVERSA ( $ -> $ fondo oscuro )

:: "\13\04\13" SWAPOVER &$ &$ ;

**Ejemplo 25 DoInputForm**

**Dar formato a los campos.**

En este ejemplo damos formatos a los campos.

Primero se muestra el programa **DoCampEstilo1**, el cual no usa ningún message handler. En este, el parámetro Decompile de un campo es un programa en el cual, luego de convertirse un objeto a cadena, se le da un formato.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoCampEstilo1 ( -> ob1 ob2 ob3 ob4 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

" Negrita:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

" Cursiva:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Subrayado:" BINT0 BINT28 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

" Inversa:" BINT0 BINT37 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO TEXTO. SE MOSTRARÁ EN NEGRITA

'DROPFALSE ( MH del campo )

BINT41 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT90 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

' :: BINT4 ( valor #4 )

ROMPTR 0B0 075 ( $ ) ( Convierte ob en $ usando Decompile )

$->NEGRITA ( $' )

; ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

25.123456789 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO TEXTO. SE MOSTRARÁ EN CURSIVA

'DROPFALSE ( MH del campo )

BINT41 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT90 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

' :: BINT4 ( valor #4 )

ROMPTR 0B0 075 ( $ ) ( Convierte ob en $ usando Decompile )

$->CURSIVA ( $' )

; ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

25.123456789 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO TEXTO. SE MOSTRARÁ SUBRAYADA

'DROPFALSE ( MH del campo )

BINT41 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT26 ( #y: 27 para fila 3/6 )

BINT90 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

' :: BINT4 ( valor #4 )

ROMPTR 0B0 075 ( $ ) ( Convierte ob en $ usando Decompile )

$->SUBRAYADA ( $' )

; ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

25.123456789 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO TEXTO. SE MOSTRARÁ EN INVERSA

'DROPFALSE ( MH del campo )

BINT41 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT35 ( #y: 36 para fila 4/6 )

BINT90 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

' :: BINT4 ( valor #4 )

ROMPTR 0B0 075 ( $ ) ( Convierte ob en $ usando Decompile )

$->INVERSA ( $' )

; ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

25.123456789 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT4 ( #Netiq )

BINT4 ( #Ncamp )

'DROPFALSE ( MH del DoInputForm )

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 ob3 ob4 T // F )

;

NULLNAME $->NEGRITA ( $ -> $ en negrita )

:: "\13\01\13" SWAPOVER &$ &$ ;

NULLNAME $->CURSIVA ( $ -> $ en cursiva )

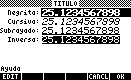
:: "\13\02\13" SWAPOVER &$ &$ ;

NULLNAME $->SUBRAYADA ( $ -> $ subrayada )

:: "\13\03\13" SWAPOVER &$ &$ ;

NULLNAME $->INVERSA ( $ -> $ fondo oscuro )

:: "\13\04\13" SWAPOVER &$ &$ ;

A continuación presentamos el programa **DoCampEstilo2**, el cual usa el message handler número 9 en cada campo donde mostrará el contenido con un formato. Recordemos que la misión del mensaje número 9 es convertir el valor del campo en un grob.

Este programa realiza la misma tarea que el programa **DoCampEstilo1**.

La diferencia es que en este programa se ve bien el contenido del campo cuando este contenido es largo.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoCampEstilo2 ( -> ob1 ob2 ob3 ob4 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

" Negrita:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

" Cursiva:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Subrayado:" BINT0 BINT28 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

" Inversa:" BINT0 BINT37 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. SE MOSTRARÁ EN NEGRITA

\* Message handler del campo:

' ::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h ) ( Retorna ancho y alto del campo )

OVER ( #c #b #h #b )

MAKEGROB ( #c #b grobbxh )

3PICK ( #c #b grobbxh #c )

ROMPTR gFldVal ( #c #b grobbxh valor ) ( Retorna Valor )

\* ----------------------------------------------------

BINT4 ( #c #b grobbxh valor #4 )

ROMPTR 0B0 075 ( #c #b grobbxh $ ) ( ob en $ usando Decompile )

$->NEGRITA ( #c #b grobbxh $' ) ( \* )

DUPLEN$ ( #c #b grobbxh $' #car ) ( \* )

#2- ( #c #b grobbxh $' #car-2 ) ( \* )

1\_#1-SUB$ ( #c #b grobbxh $' ) ( \* )

DUPLEN$ ( #c #b grobbxh $' #car )

BINT6 ( #c #b grobbxh $' #car #6 )

#\* ( #c #b grobbxh $' #6\*car )

4ROLL ( #c grobbxh $' #6\*car #b )

BINT18 #+ ( #c grobbxh $' #6\*car #b+18 ) ( \* )

DUPUNROT ( #c grobbxh $' #b+18 #6\*car #b+18 )

#<ITE

DROP ( #c grobbxh $' )

:: ( #c grobbxh $' #b+18 )

BINT6 ( #c grobbxh $' #b+18 6 )

#/ ( #c grobbxh $' #r #q )

SWAPDROP ( #c grobbxh $' #q )

1\_#1-SUB$ ( #c grobbxh $'' )

CHR\_... ( #c grobbxh $'' chr )

>T$ ( #c grobbxh $''' )

;

$>GROB ( #c grobbxh grob )

\* -----------------------------------------------------

OVER ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' ) ( Reempl grob1 en grob2 )

TRUE ( #c grobbxh' T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT41 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT90 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

25.1234567898 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO TEXTO. SE MOSTRARÁ EN CURSIVA

\* Message handler del campo:

' ::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h ) ( Retorna ancho y alto del campo )

OVER ( #c #b #h #b )

MAKEGROB ( #c #b grobbxh )

3PICK ( #c #b grobbxh #c )

ROMPTR gFldVal ( #c #b grobbxh valor ) ( Retorna Valor )

\* ----------------------------------------------------

BINT4 ( #c #b grobbxh valor #4 )

ROMPTR 0B0 075 ( #c #b grobbxh $ ) ( ob en $ usando Decompile )

$->CURSIVA ( #c #b grobbxh $' ) ( \* )

DUPLEN$ ( #c #b grobbxh $' #car ) ( \* )

#2- ( #c #b grobbxh $' #car-2 ) ( \* )

1\_#1-SUB$ ( #c #b grobbxh $' ) ( \* )

DUPLEN$ ( #c #b grobbxh $' #car )

BINT6 ( #c #b grobbxh $' #car #6 )

#\* ( #c #b grobbxh $' #6\*car )

4ROLL ( #c grobbxh $' #6\*car #b )

BINT18 #+ ( #c grobbxh $' #6\*car #b+18 ) ( \* )

DUPUNROT ( #c grobbxh $' #b+18 #6\*car #b+18 )

#<ITE

DROP ( #c grobbxh $' )

:: ( #c grobbxh $' #b+18 )

BINT6 ( #c grobbxh $' #b+18 6 )

#/ ( #c grobbxh $' #r #q )

SWAPDROP ( #c grobbxh $' #q )

1\_#1-SUB$ ( #c grobbxh $'' )

CHR\_... ( #c grobbxh $'' chr )

>T$ ( #c grobbxh $''' )

;

$>GROB ( #c grobbxh grob )

\* -----------------------------------------------------

OVER ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' ) ( Reempl grob1 en grob2 )

TRUE ( #c grobbxh' T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT41 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT90 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

25.1234567898 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 3: ES UN CAMPO TEXTO. SE MOSTRARÁ SUBRAYADA

\* Message handler del campo:

' ::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h ) ( Retorna ancho y alto del campo )

OVER ( #c #b #h #b )

MAKEGROB ( #c #b grobbxh )

3PICK ( #c #b grobbxh #c )

ROMPTR gFldVal ( #c #b grobbxh valor ) ( Retorna Valor )

\* ----------------------------------------------------

BINT4 ( #c #b grobbxh valor #4 )

ROMPTR 0B0 075 ( #c #b grobbxh $ ) ( ob en $ usando Decompile )

$->SUBRAYADA ( #c #b grobbxh $' ) ( \* )

DUPLEN$ ( #c #b grobbxh $' #car ) ( \* )

#2- ( #c #b grobbxh $' #car-2 ) ( \* )

1\_#1-SUB$ ( #c #b grobbxh $' ) ( \* )

DUPLEN$ ( #c #b grobbxh $' #car )

BINT6 ( #c #b grobbxh $' #car #6 )

#\* ( #c #b grobbxh $' #6\*car )

4ROLL ( #c grobbxh $' #6\*car #b )

BINT18 #+ ( #c grobbxh $' #6\*car #b+18 ) ( \* )

DUPUNROT ( #c grobbxh $' #b+18 #6\*car #b+18 )

#<ITE

DROP ( #c grobbxh $' )

:: ( #c grobbxh $' #b+18 )

BINT6 ( #c grobbxh $' #b+18 6 )

#/ ( #c grobbxh $' #r #q )

SWAPDROP ( #c grobbxh $' #q )

1\_#1-SUB$ ( #c grobbxh $'' )

CHR\_... ( #c grobbxh $'' chr )

>T$ ( #c grobbxh $''' )

;

$>GROB ( #c grobbxh grob )

\* -----------------------------------------------------

OVER ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' ) ( Reempl grob1 en grob2 )

TRUE ( #c grobbxh' T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT41 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT26 ( #y: 27 para fila 3/6 )

BINT90 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

25.1234567898 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 4: ES UN CAMPO TEXTO. SE MOSTRARÁ EN INVERSA

\* Message handler del campo:

' ::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h ) ( Retorna ancho y alto del campo )

OVER ( #c #b #h #b )

MAKEGROB ( #c #b grobbxh )

3PICK ( #c #b grobbxh #c )

ROMPTR gFldVal ( #c #b grobbxh valor ) ( Retorna Valor )

\* ----------------------------------------------------

BINT4 ( #c #b grobbxh valor #4 )

ROMPTR 0B0 075 ( #c #b grobbxh $ ) ( ob en $ usando Decompile )

$->INVERSA ( #c #b grobbxh $' ) ( \* )

DUPLEN$ ( #c #b grobbxh $' #car ) ( \* )

#2- ( #c #b grobbxh $' #car-2 ) ( \* )

1\_#1-SUB$ ( #c #b grobbxh $' ) ( \* )

DUPLEN$ ( #c #b grobbxh $' #car )

BINT6 ( #c #b grobbxh $' #car #6 )

#\* ( #c #b grobbxh $' #6\*car )

4ROLL ( #c grobbxh $' #6\*car #b )

BINT18 #+ ( #c grobbxh $' #6\*car #b+18 ) ( \* )

DUPUNROT ( #c grobbxh $' #b+18 #6\*car #b+18 )

#<ITE

DROP ( #c grobbxh $' )

:: ( #c grobbxh $' #b+18 )

BINT6 ( #c grobbxh $' #b+18 6 )

#/ ( #c grobbxh $' #r #q )

SWAPDROP ( #c grobbxh $' #q )

1\_#1-SUB$ ( #c grobbxh $'' )

CHR\_... ( #c grobbxh $'' chr )

>T$ ( #c grobbxh $''' )

;

$>GROB ( #c grobbxh grob )

\* -----------------------------------------------------

OVER ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' ) ( Reempl grob1 en grob2 )

TRUE ( #c grobbxh' T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT41 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT35 ( #y: 36 para fila 4/6 )

BINT90 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

25.1234567898 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT4 ( #Netiq )

BINT4 ( #Ncamp )

'DROPFALSE ( MH del DoInputForm )

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 ob3 ob4 T // F )

;

NULLNAME $->NEGRITA ( $ -> $ en negrita )

:: "\13\01\13" SWAPOVER &$ &$ ;

NULLNAME $->CURSIVA ( $ -> $ en cursiva )

:: "\13\02\13" SWAPOVER &$ &$ ;

NULLNAME $->SUBRAYADA ( $ -> $ subrayada )

:: "\13\03\13" SWAPOVER &$ &$ ;

NULLNAME $->INVERSA ( $ -> $ fondo oscuro )

:: "\13\04\13" SWAPOVER &$ &$ ;

**Ejemplo 26 DoInputForm**

**Verificar que todos los campos estén llenos al terminar el formulario con OK.**

En este ejemplo, al presionar OK o ENTER se verifica que todos los campos estén llenos.

Si hay algún campo vacío, se evita la finalización del formulario (y se muestra el mensaje “escribe todos los datos”).

El comando FLASHPTR GetFieldVals ( 🡪 ob1 ob2 ... obn )

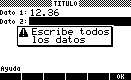
pone todos los objetos de los campos en la pila.

El número de campos se encuentra guardado en el lam14 como un bint. Y es llamado con el comando 14GETLAM.

El comando FLASHPTR ListPos ( ob {} 🡪 #i/#0 )

pone en la pila la ubicación de un objeto dentro de una lista como un BINT,

si el objeto no está presente en la lista, pone BINT0.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoCamposLlenos ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"Dato 1:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"Dato 2:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT29 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT102 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

MINUSONE ( ValorReset: campo TEXTO en blanco )

MINUSONE ( ValorInicial: campo TEXTO en blanco )

\* CAMPO 2: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT29 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT102 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Ayuda" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

MINUSONE ( ValorReset: campo TEXTO en blanco )

MINUSONE ( ValorInicial: campo TEXTO en blanco )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

BINT29 #=casedrop ( F ) ( -> flag T // F )

::

FLASHPTR GetFieldVals ( ob1 ob2...obn )

14GETLAM ( ob1 ob2...obn #n )

{}N ( { ob1 ob2...obn } )

MINUSONE ( { ob1 ob2...obn } MINUSONE )

SWAP ( MINUSONE { ob1 ob2...obn } )

FLASHPTR ListPos ( #pos/#0 )

#0=ITE

TrueTrue

:: "Escribe todos los datos" FlashWarning FalseTrue ;

( flag T )

;

\* Fin del Message handler del formulario

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 T // F )

;

**Ejemplo 27 DoInputForm**

**Verificar que los valores de los campos cumplan alguna condición antes de confirmar la salida con OK o ENTER.**

En este ejemplo, al presionar OK o ENTER se verifica que los valores de los campos cumplan alguna condición.

Si algún campo no cumple una condición, entonces se muestra un mensaje de advertencia y no se termina el formulario. Además, el enfoque va hacia ese campo.

El comando ROMPTR gFldVal ( #campo -> ob )

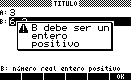
retorna el valor de un campo.

El comando FLASHPTR ChangeFocus ( #campo -> )

fija al campo especificado como el nuevo campo actual (el que tendrá el enfoque).

El comando ROMPTR 0B0 016 ( #campo -> )

invierte los píxeles en un campo.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoCamposValidOK ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"B:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"A: número real entero positivo" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

MINUSONE ( ValorReset: campo TEXTO en blanco )

MINUSONE ( ValorInicial: campo TEXTO en blanco )

\* CAMPO 2: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"B: número real entero positivo" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

MINUSONE ( ValorReset: campo TEXTO en blanco )

MINUSONE ( ValorInicial: campo TEXTO en blanco )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

BINT29 #=casedrop ( F ) ( -> flag T // F )

::

BINT1 ( #c )

ROMPTR gFldVal ( ob ) ( Retorna Valor )

TestRealEnteroPositivo ( flag )

NOTcase

:: ( )

BINT1 ( #1 )

ChangeFocus3 ( )

"A debe ser un entero positivo" FlashWarning

FalseTrue ( F T )

;

BINT2 ( #c )

ROMPTR gFldVal ( ob ) ( Retorna Valor )

TestRealEnteroPositivo ( flag )

NOTcase

:: ( )

BINT2 ( #2 )

ChangeFocus3 ( )

"B debe ser un entero positivo" FlashWarning

FalseTrue ( F T )

;

TrueTrue ( T T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 T // F )

;

\* Test realizado a un objeto.

\* Retorna TRUE si en la pila hay un número real que sea entero y positivo.

NULLNAME TestRealEnteroPositivo ( ob -> flag )

:: ( ob )

DUPTYPEREAL? ( ob flag )

NOTcase DROPFALSE ( sale con FALSE )

( % )

DUP ( % % )

%0> ( % flag )

NOTcase DROPFALSE ( sale con FALSE )

( % )

%FP ( %ParteDecimal )

%0= ( flag )

;

\* Mueve el enfoque al campo especificado

\* Actualiza visualización de campos (invierte píxeles).

\* Actualiza la zona de la ayuda.

NULLNAME ChangeFocus3 ( #c -> )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 097 ( #c ) ( cambia el enfoque e invierte píxeles )

TRUESWAP\_ ( T #c )

ROMPTR 0B0 017 ( ) ( muestra ayuda del campo justo encima de los menús )

;

**Ejemplo 28 DoInputForm.**

**Retornar el valor de un campo como grob.**

Este es un message handler para un campo (que no sea campo CHECK).

Este tiene el mismo efecto que el código por defecto para retornar el valor del campo como grob (de dimensiones #b por #h).

Lo presentamos para que hagas los cambios que creas conveniente para tu programa.

NULLNAME DoMH9\_AccionesPorDefecto

:: BINT9 #=casedrop ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 045 ( #c $ )

OVER ( #c $ #c )

ROMPTR 0B0 0CA ( #c $ #b #h )

ROTDUP ( #c #b #h $ $ )

LEN$ ( #c #b #h $ #car )

BINT6 ( #c #b #h $ #car #6 )

#\* ( #c #b #h $ #6ocar )

4PICK ( #c #b #h $ #6ocar #b )

DUPUNROT ( #c #b #h $ #b #6ocar #b )

#< ( #c #b #h $ #b flag )

ITE

DROP ( #c #b #h $ )

:: ( #c #b #h $ #b )

BINT6 ( #c #b #h $ #b #6 )

#/ ( #c #b #h $ #residuo #b/6 )

SWAPDROP ( #c #b #h $ #b/6 )

1\_#1-SUB$ ( #c #b #h $' )

CHR\_... ( #c #b #h $' chr )

>T$ ( #c #b #h $'' )

;

$>GROB ( #c #b #h grob )

3UNROLL ( #c grob #b #h )

SWAP ( #c grob #h #b )

MAKEGROB ( #c grob grobbxh )

DUPUNROT ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' )

TRUE ( #c grobbxh' T )

;

DROPFALSE

;

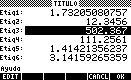
**Ejemplo 29 DoInputForm**

**Alinear campos a la derecha respetando el parámetro Decompile.**

**Fuente normal**

Usando este message handler, el valor del campo (que no sea campo CHECK) será mostrado a la derecha respetando el parámetro Decompile del campo y con fuente normal.

Puedes poner este NULLNAME como el message handler en cada campo donde desees que su contenido se muestre pegado a la derecha.



\* Este es el message handler que puedes usar para alinear el

\* contenido del campo a la derecha (Usando Fuente normal)

\* Ponlo como el message handler de cada campo que desees.

NULLNAME DoMH9\_AlinCampDerecha

:: BINT9 #=casedrop ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h )

3PICK ( #c #b #h #c )

ROMPTR 0B0 045 ( #c #b #h $ )

3PICK ( #c #b #h $ #b )

BINT6 ( #c #b #h $ #b #6 )

#/ ( #c #b #h $ #residuo #b/6 )

SWAPDROP ( #c #b #h $ #b/6 )

FUERZA$ ( #c #b #h $' )

$>GROB ( #c #b #h grob )

3UNROLL ( #c grob #b #h )

SWAP ( #c grob #h #b )

MAKEGROB ( #c grob grobbxh )

DUPUNROT ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' )

TRUE ( #c grobbxh' T )

;

DROPFALSE

;

\* Este subprograma retorna siempre una cadena que tenga #max caracteres

\* Si $ tiene pocos caracteres, le agrega caract en blanco a la izquierda

\* Si $ tiene muchos caracteres, lo corta y agrega "..." al final

NULLNAME FUERZA$ ( $ #max -> $ )

:: ( $ #max )

OVERLEN$ ( $ #max #len )

2DUP#= ( $ #max #len flag )

case2DROP

( $ #max #len )

2DUP#< ( $ #max #len flag )

casedrop

:: 1\_#1-SUB$ "\1E" &$ ;

( $ #max #len )

#- ( $ #adicionales )

Blank$ ( $ $' )

SWAP&$ ( $'' )

;

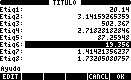
**Ejemplo 30 DoInputForm**

**Alinear campos a la derecha respetando el parámetro Decompile.**

**MiniFuente**

Usando este message handler, el valor del campo (que no sea campo CHECK) será mostrado a la derecha respetando el parámetro Decompile del campo y con Minifuente.

Puedes poner este NULLNAME como el message handler en cada campo donde desees que su contenido se muestre pegado a la derecha.



\* Este es el message handler que puedes usar para alinear el

\* contenido del campo a la derecha (Usando MiniFuente)

\* Ponlo como el message handler de cada campo que desees.

NULLNAME DoMH9\_AlinCampDerechaMF

:: BINT9 #=casedrop ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h )

3PICK ( #c #b #h #c )

ROMPTR 0B0 045 ( #c #b #h $ )

3PICK ( #c #b #h $ #b )

BINT4 ( #c #b #h $ #b #4 )

#/ ( #c #b #h $ #residuo #b/4 )

SWAPDROP ( #c #b #h $ #b/4 )

FUERZA$ ( #c #b #h $' )

$>grob ( #c #b #h grob )

3UNROLL ( #c grob #b #h )

SWAP ( #c grob #h #b )

MAKEGROB ( #c grob grobbxh )

DUPUNROT ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' )

TRUE ( #c grobbxh' T )

;

DROPFALSE

;

\* Este subprograma retorna siempre una cadena que tenga #max caracteres

\* Si $ tiene pocos caracteres, le agrega caract en blanco a la izquierda

\* Si $ tiene muchos caracteres, lo corta y agrega "..." al final

NULLNAME FUERZA$ ( $ #max -> $ )

:: ( $ #max )

OVERLEN$ ( $ #max #len )

2DUP#= ( $ #max #len flag )

case2DROP

( $ #max #len )

2DUP#< ( $ #max #len flag )

casedrop

:: 1\_#1-SUB$ "\1E" &$ ;

( $ #max #len )

#- ( $ #adicionales )

Blank$ ( $ $' )

SWAP&$ ( $'' )

;

**Ejemplo 31 DoInputForm**

**Validación de datos.**

**Número real y positivo.**

En este ejemplo veremos como usar el mensaje 46 para validar un campo que acepte a números reales.

En este caso hay dos campos que contienen a números reales. La validación se hará solamente en el primer campo. Por ejemplo, para este campo el número real será válido si es positivo. De esta manera, si el objeto es de un valor inválido, deberemos de retornar FALSE TRUE. En otro caso, sólo se deberá agregar FALSE, para que se realice la acción por defecto, es decir, aceptar el valor ingresado.

Por ejemplo, si el usuario ingresa -12.29 (como se muestra en la figura), en el campo no se guardará el número y se mostrará la alerta “Invalid Object Value”.

De esta manera, está asegurado que al salir del formulario con ENTER u OK, siempre será retornado un valor válido; y ya no será necesario usar el mensaje 29 en el message handler del formulario.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoValidReal ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"B:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* Message handler del campo:

' ::

\* Cuando se quiere confirmar línea de edición con ENTER u OK

\* o cuando se quiere asignar valor al campo desde la tecla CALC

\* Su misión es verificar que el objeto tenga un valor válido.

\* Con TRUE TRUE, el obj será asignado al campo (similar sin efecto)

\* Con FALSE TRUE, el obj no se asignará al campo y se mostrará el

\* mensaje "Invalid object value".

BINT46 #=casedrop ( C ) ( valor -> flag T // valor F )

:: ( % )

%0> ( % flag )

TRUE ( flag T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"A: número real positivo" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

MINUSONE ( ValorReset: campo TEXTO en blanco )

MINUSONE ( ValorInicial: campo TEXTO en blanco )

\* CAMPO 2: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"B: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

MINUSONE ( ValorReset: campo TEXTO en blanco )

MINUSONE ( ValorInicial: campo TEXTO en blanco )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

'DROPFALSE ( MH del DoInputForm )

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 T // F )

;

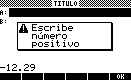
**Ejemplo 32 DoInputForm**

**Validación de datos con mensaje personalizado.**

**Número real y positivo.**

Este ejemplo es similar al anterior. La diferencia es que ahora puedes colocar un mensaje personalizado cuando hay un objeto inválido.

Aquí se manipula la pila de retornos. A continuación mostramos el código sin explicación.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoValidRealPers ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"B:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* Message handler del campo:

' ::

\* Cuando se quiere confirmar línea de edición con ENTER u OK

\* o cuando se quiere asignar valor al campo desde la tecla CALC

\* Su misión es verificar que el objeto tenga un valor válido.

\* Con TRUE TRUE, el obj será asignado al campo (similar sin efecto)

\* Con FALSE TRUE, el obj no se asignará al campo y se mostrará el

\* mensaje "Invalid object value".

BINT46 #=casedrop ( C ) ( valor -> flag T // valor F )

:: ( ob %Valor #c %Valor )

DUP ( ob %Valor #c %Valor %Valor )

%0> ( ob %Valor #c %Valor flag )

caseFALSE ( sale con: %Valor F )

( ob %Valor #c %Valor )

RDROP ( ob %Valor #c %Valor )

RDROP ( ob %Valor #c %Valor )

RDROP ( ob %Valor #c %Valor )

2DROP ( ob %Valor )

' ' ( ob %Valor ' )

SWAP ( ob ' %Valor )

BINT2 ( ob ' %Valor #2 )

::N ( ob :: ' %Valor ; )

' :: ( %Valor )

DecompEdit ( $ )

InitEdLine ( $ ) ( Limpia línea de edic. y finaliza editor )

EditString ( ) ( Inicia línea de edición )

FLASHPTR 001 347 ( )

"Escribe número positivo" ( $ )

FlashWarning ( )

;

( ob :: ' %Valor ; prog )

&COMP ( ob prog' )

FALSE ( ob prog' F )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"A: número real positivo" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

MINUSONE ( ValorReset: campo TEXTO en blanco )

MINUSONE ( ValorInicial: campo TEXTO en blanco )

\* CAMPO 2: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"B: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

MINUSONE ( ValorReset: campo TEXTO en blanco )

MINUSONE ( ValorInicial: campo TEXTO en blanco )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

'DROPFALSE ( MH del DoInputForm )

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 T // F )

;

**Ejemplo 33 DoInputForm**

**Cambiando automáticamente una entrada inválida.**

**Número real entero y positivo.**

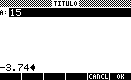
En este ejemplo hay un campo TEXTO que acepta sólo números reales.

Sólo se aceptan números reales que sean enteros y positivos.

Si el usuario escribe un número negativo, la calculadora automáticamente le cambia de signo. También, si el número tiene parte decimal, automáticamente el número es redondeado de manera que el valor guardado en el campo (valor interno) sea siempre un número real entero y positivo.

Por ejemplo, si el usuario ingresa -3.74 (como se muestra en la figura), en el campo se guardará el número 4 en lugar del número ingresado.

Por lo tanto, el valor del campo devuelto al finalizar con OK o ENTER será siempre un número real que sea entero y positivo.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoCampoEntPosit ( -> ob T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

\* Message handler del campo:

' ::

\* Llamado inmediatamente después de que el campo ha cambiado su valor

\* Puede hacer cualquier cosa aquí. No hay diferencia si retornas T o F

\*\*\* CARACTERISTICAS DE ESTE CODIGO:

\*\*\* Si hay un número real en el campo, entonces retorna su valor

\*\*\* absoluto y lo redondea al entero positivo más cercano.

\*\*\* y lo guarda como valor del campo.

BINT47 #=casedrop ( -> flag )

:: ( )

4GETLAM ( #c )

ROMPTR gFldVal ( valor ) ( Retorna Valor )

DUP ( valor valor )

MINUSONE ( valor valor MINUSONE )

EQUAL ( valor flag )

case DROPTRUE ( sale con TRUE )

( % )

%ABS %0 RNDXY ( %' )

DUP%0= ( %' flag )

IT %1+

( %'' )

4GETLAM ( %'' #c )

ROMPTR sFldVal ( ) ( cambia valor del campo )

TRUE ( T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"A: número real entero positivo" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

BINT1 ( #Netiq )

BINT1 ( #Ncamp )

'DROPFALSE ( MH del DoInputForm )

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob T // F )

;

**Ejemplo 34 DoInputForm**

**Campo combofiler con una formación.**

**Manejando los mensajes 9, 23, 36, 37 y 29.**

En este ejemplo hay un campo COMBOFILER que contiene una formación (matriz o arreglo).

Con el mensaje 9, podemos ver la formación como un grob.

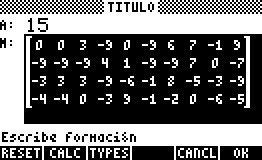
Con el mensaje 23, podemos editar la formación en el MTRW al presionar EDIT.

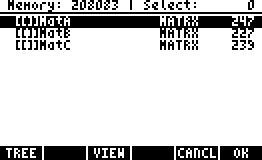
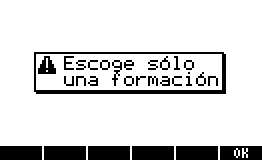
Con el mensaje 36, el campo también aceptará a matrices simbólicas (tipo 29).

Con el mensaje 37, evitaremos que en el filer sean escogidas varias formaciones a la vez (selección múltiple).

En DoInputForm normalmente no es posible ingresar Matrices Simbólicas en un campo. Para lograr poder ingresar matrices simbólicas en un campo se coloca el BINT29 en la lista de TiposPermitidos del campo y además se usa el mensaje 36 en el campo.

Con el mensaje 29 en el formulario, podemos evitar la finalización del formulario (al presionar ENTER u OK), si el campo 2 está vacío.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoFormacion ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"M:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"A: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO COMBOFILER. CONTIENE A UNA FORMACIÓN

' MH9&23&36&37\_Formacion ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT45 ( #h: 62-#y para llegar hasta abajo )

BINT21 ( #TipoDeCampo: COMBOFILER )

{ BINT3 BINT4 BINT29 } ( TiposPermit: 3=ArrReales 4=ArrComp 29=Matr )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Escribe formación" ( "Ayuda" )

MINUSONE ( ChooseData en FILER: F=NoCHK TiposP=TiposPC T=RetObj )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo COMBOFILER )

MINUSONE ( ValorReset: campo COMBOFILER en blanco )

MINUSONE ( ValorInicial: campo COMBOFILER en blanco )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

BINT29 #=casedrop ( F ) ( -> flag T // F )

::

BINT2 ( #c )

ROMPTR gFldVal ( ob )

MINUSONE ( ob MINUSONE )

EQUAL ( flag )

case

:: ( )

BINT2 ( #c )

ChangeFocus3 ( )

"ESCRIBE FORMACIÓN" ( $ )

FlashWarning ( )

FalseTrue ( F T )

;

TrueTrue ( T T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 T // F )

;

NULLNAME MH9&23&36&37\_Formacion

::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Convierte objeto a grob con ob>grobmini (MiniFuente)

\*\*\* y retorna el grob correspondiente del tamaño del campo

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h ) ( Retorna ancho y alto del campo )

SWAP ( #c #h #b )

MAKEGROB ( #c grobbxh )

OVER ( #c grobbxh #c )

ROMPTR gFldVal ( #c grobbxh valor )

DUP ( #c grobbxh valor valor )

MINUSONE ( #c grobbxh valor valor MINUSONE )

EQUALcase

DROPTRUE ( sale con: #c grobbxh T )

( #c grobbxh valor )

ob>grobmini ( #c grobbxh grob )

OVER ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' ) ( Reempl grob1 en grob2 )

TRUE ( #c grobbxh' T )

;

\* Al presionar la tecla EDIT y su misión es editar el valor del campo

\* Acción por defecto: inicia línea de edic con valor actual del campo

BINT23 #=casedrop ( C ) ( -> T/F )

:: ( )

4GETLAM ( #c )

ROMPTR gFldVal ( valor )

EditaFormacionEnMTRW ( RealArry/CArry/MATRIX T // F )

IT

:: 5GETLAM

ROMPTR sFldVal

;

TRUE ( T )

;

\* Cuando es presionada TYPES en campos TEXTO, COMBOCHOOSE, COMBOFILER

\* También cuando después de ingresar un valor de tipo incorrecto en un

\* campo, aparece una alerta y el usuario presiona tecla de menú TYPES

\* Este mensaje debe retornar una lista de listas.

\* Cada una de las sublistas debe tener la forma:

\* { #/$ :: #pos FlagALG $EditLine ; }

\* - #/$ Es lo que se mostrará en el browser

\* - #pos Es la posición del cursor en la nueva línea de edición.

\* - FlagALG Si es TRUE, se activará el modo algebraico.

\* - $EditLine Es la cadena de la nueva línea de edición que se abrirá.

\* Con este mensaje puedes tener más TiposPermitidos en DoInputForm

\* (los BINTS se deben colocar en la lista de TiposPeritidos del campo)

BINT36 #=casedrop ( C ) ( -> {{}} T // F )

:: ( )

{ { "Arreglo Real" :: BINT2 FALSE "[]" ; }

{ "Arreglo complejo" :: BINT3 FALSE "[()]" ; }

{ "Matriz simbólica" :: BINT2 FALSE "[]" ; }

} ( {{}} )

TRUE ( {{}} T )

;

\* Cuando es presionada la tecla CHOOSE en un campos CHOOSE, FILER,

\* COMBOCHOOSE o COMBOFILER.

\* Su misión es presentar las opciones al usuario

\* si una opción es escogida, retornarla en la pila seguida de T T

\* Si el usuario no escoge ninguna opción, en la pila debe ponerse F T

BINT37 #=casedrop ( C ) ( -> ob T T // F T // F )

:: ( )

FALSE ( F )

4GETLAM ( F #c )

\* Retorna los parámetros de ^BrowseMem.1 a partir de ChooseData

\* Puedes cambiar este comando por otro código

ROMPTR 0B0 078 ( F ob2 flagCheck {}tipos flagObjNomb )

SWAP

TOTEMPOB

SWAP ( F ob2 flagCheck {}tipos flagObjNomb )

BrowMem3&4 ( ob T // F )

TRUE ( ob T T // F T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

\* CONVIERTE ob A GROB PEQUEÑO USANDO MINIFUENTE

\* Intenta convertir a grob con FLASHPTR EQW3GROBmini

\* Si no es posible hace lo siguiente:

\* Si ob es un grob, no hace nada.

\* Si ob es una cadena, hace $>grobCR

\* Si ob es de otro tipo, hace FLASHPTR FSTR11 $>grobCR

NULLNAME ob>grobmini ( ob -> grob )

:: ( ob )

BEGIN

:: DUP ( ob ob )

FLASHPTR EQW3GROBmini ( ob ext grob #0 // ob ob #1/2 )

BINT0 #=casedrop

:: UNROT2DROP TRUE ; ( OBS: SALE CON grob T ) ( SALE DE BUCLE )

( ob ob #1/2 )

BINT1 #=case

:: GARBAGE DROPFALSE ; ( OBS: SALE CON ob F ) ( REPITE BUCLE )

( ob ob )

TYPEGROB? ( ob flag )

caseTRUE

( ob )

DUPTYPECSTR? ( ob flag )

?SKIP

FLASHPTR FSTR11

( $ )

$>grobCR ( grob )

TRUE ( grob T ) ( SALE DE BUCLE )

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( grob )

;

\* Si en la pila se encuentra xNOVAL o MINUSONE, abre el MTRW.

\* Si hay otro objeto, lo edita en MTRW (de ser posible)

NULLNAME EditaFormacionEnMTRW ( ob -> RealArry/CArry/MATRIX T // F )

::

RunSafeFlags

:: BEGIN

BINT91 ClrSysFlag ( ob ) ( 91 MTRW: no list of list )

DUPTYPEBINT? ( ob flag )

OVER ( ob flag ob )

TYPECOL? ( ob flag flag' )

OR ( ob flag'' )

ITE

:: DROP ( )

FLASHPTR RunDoNewMatrix ( ob' T // F ) ( abre MTRW )

;

FLASHPTR RunDoOldMatrix ( ob' T // F ) ( edita ob en MTRW )

ITE

:: DUPTYPELIST? ( ob' flag )

ITE

FALSE ( ob' F )

TrueTrue ( ob' T T )

;

FalseTrue

UNTIL

( RealArry/CArry/MATRIX T // F )

;

;

\* Parecido al comando FLASHPTR BrowseMem.1

\* Sólo es posible retornar Arreglos y Matrices.

\* Sólo es permitido retornar un objeto.

\* De lo contrario, se muestra el mensaje "Escoge sÓlo una formaciÓn"

NULLNAME BrowMem3&4 ( F ob2 flagCheck {}tipos flagObjNomb -> ob T // F )

:: ( F ob2 flagCheck {}tipos flagObjNomb )

BINT5 NDUP ( ... F ob2 flagCheck {}tipos flagObjNomb )

FLASHPTR BrowseMem.1 ( ... ob T // F // tag )

TRUE

EQUAL ( ... ob T // F )

NOTcase

:: BINT5 NDROPFALSE ; ( sale con: FALSE )

( ... ob )

DUPTYPEARRY? ( ... ob flag )

OVER TYPEMATRIX?\_ ( ... ob flag flag' )

OR ( ... ob flag'' )

case

:: BINT6 UNROLL ( ob ... )

BINT5 NDROP ( ob )

TRUE ( ob T )

; ( sale con: ob T )

( ... {} )

DROP ( ... )

CLEARVDISP ( ... )

"Escoge sÓlo una formaciÓn" ( ... $ )

FlashWarning ( ... )

COLA

BrowMem3&4

;

\* Mueve el enfoque al campo especificado

\* Actualiza visualización de campos (invierte píxeles).

\* Actualiza la zona de la ayuda.

NULLNAME ChangeFocus3 ( #c -> )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 097 ( #c ) ( cambia el enfoque e invierte píxeles )

TRUESWAP\_ ( T #c )

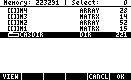
ROMPTR 0B0 017 ( ) ( muestra ayuda del campo justo encima de menús )

;

También se puede usar el siguiente código, en el cual ya no usa el comando

FLASHPTR BrowseMem.1

Ahora se usará el comando FLASHPTR FILER\_MANAGERTYPE



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoFormacion ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"M:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"A: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO COMBOFILER. CONTIENE A UNA FORMACIÓN

' MH9&23&36&37\_Formacion ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT45 ( #h: 62-#y para llegar hasta abajo )

BINT21 ( #TipoDeCampo: COMBOFILER )

{ BINT3 BINT4 BINT29 } ( TiposPermit: 3=ArrReales 4=ArrComp 29=Matr )

MINUSONE ( Decompile: No es necesario, pues mensaje 6 es manejado )

"Escribe formación" ( "Ayuda" )

MINUSONE ( ChooseData en FILER: No es necesario, por mensaje 37 )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo COMBOFILER )

MINUSONE ( ValorReset: campo COMBOFILER en blanco )

MINUSONE ( ValorInicial: campo COMBOFILER en blanco )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

BINT29 #=casedrop ( F ) ( -> flag T // F )

::

BINT2 ( #c )

ROMPTR gFldVal ( ob )

MINUSONE ( ob MINUSONE )

EQUAL ( flag )

case

:: ( )

BINT2 ( #c )

ChangeFocus3 ( )

"ESCRIBE FORMACIÓN" ( $ )

FlashWarning ( )

FalseTrue ( F T )

;

TrueTrue ( T T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 T // F )

;

NULLNAME MH9&23&36&37\_Formacion

::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Convierte objeto a grob con ob>grobmini (MiniFuente)

\*\*\* y retorna el grob correspondiente del tamaño del campo

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h ) ( Retorna ancho y alto del campo )

SWAP ( #c #h #b )

MAKEGROB ( #c grobbxh )

OVER ( #c grobbxh #c )

ROMPTR gFldVal ( #c grobbxh valor )

DUP ( #c grobbxh valor valor )

MINUSONE ( #c grobbxh valor valor MINUSONE )

EQUALcase

DROPTRUE ( sale con: #c grobbxh T )

( #c grobbxh valor )

ob>grobmini ( #c grobbxh grob )

OVER ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' ) ( Reempl grob1 en grob2 )

TRUE ( #c grobbxh' T )

;

\* Al presionar la tecla EDIT y su misión es editar el valor del campo

\* Acción por defecto: inicia línea de edic con valor actual del campo

BINT23 #=casedrop ( C ) ( -> T/F )

:: ( )

4GETLAM ( #c )

ROMPTR gFldVal ( valor )

EditaFormacionEnMTRW ( RealArry/CArry/MATRIX T // F )

IT

:: 5GETLAM

ROMPTR sFldVal

;

TRUE ( T )

;

\* Cuando es presionada TYPES en campos TEXTO, COMBOCHOOSE, COMBOFILER

\* También cuando después de ingresar un valor de tipo incorrecto en un

\* campo, aparece una alerta y el usuario presiona tecla de menú TYPES

\* Este mensaje debe retornar una lista de listas.

\* Cada una de las sublistas debe tener la forma:

\* { #/$ :: #pos FlagALG $EditLine ; }

\* - #/$ Es lo que se mostrará en el browser

\* - #pos Es la posición del cursor en la nueva línea de edición.

\* - FlagALG Si es TRUE, se activará el modo algebraico.

\* - $EditLine Es la cadena de la nueva línea de edición que se abrirá.

\* Con este mensaje puedes tener más TiposPermitidos en DoInputForm

\* (los BINTS se deben colocar en la lista de TiposPeritidos del campo)

BINT36 #=casedrop ( C ) ( -> {{}} T // F )

:: ( )

{ { "Arreglo Real" :: BINT2 FALSE "[]" ; }

{ "Arreglo complejo" :: BINT3 FALSE "[()]" ; }

{ "Matriz simbólica" :: BINT2 FALSE "[]" ; }

} ( {{}} )

TRUE ( {{}} T )

;

\* Cuando es presionada la tecla CHOOSE en un campos CHOOSE, FILER,

\* COMBOCHOOSE o COMBOFILER.

\* Su misión es presentar las opciones al usuario

\* si una opción es escogida, retornarla en la pila seguida de T T

\* Si el usuario no escoge ninguna opción, en la pila debe ponerse F T

BINT37 #=casedrop ( C ) ( -> ob T T // F T // F )

:: ( )

Filer\_Home\_S1\_NoDir\_FORMACION ( Formacion T // F )

TRUE ( Formacion T T // F T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

\* CONVIERTE ob A GROB PEQUEÑO USANDO MINIFUENTE

\* Intenta convertir a grob con FLASHPTR EQW3GROBmini

\* Si no es posible hace lo siguiente:

\* Si ob es un grob, no hace nada.

\* Si ob es una cadena, hace $>grobCR

\* Si ob es de otro tipo, hace FLASHPTR FSTR11 $>grobCR

NULLNAME ob>grobmini ( ob -> grob )

:: ( ob )

BEGIN

:: DUP ( ob ob )

FLASHPTR EQW3GROBmini ( ob ext grob #0 // ob ob #1/2 )

BINT0 #=casedrop

:: UNROT2DROP TRUE ; ( OBS: SALE CON grob T ) ( SALE DE BUCLE )

( ob ob #1/2 )

BINT1 #=case

:: GARBAGE DROPFALSE ; ( OBS: SALE CON ob F ) ( REPITE BUCLE )

( ob ob )

TYPEGROB? ( ob flag )

caseTRUE

( ob )

DUPTYPECSTR? ( ob flag )

?SKIP

FLASHPTR FSTR11

( $ )

$>grobCR ( grob )

TRUE ( grob T ) ( SALE DE BUCLE )

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( grob )

;

\* Si en la pila se encuentra xNOVAL o MINUSONE, abre el MTRW.

\* Si hay otro objeto, lo edita en MTRW (de ser posible)

NULLNAME EditaFormacionEnMTRW ( ob -> RealArry/CArry/MATRIX T // F )

::

RunSafeFlags

:: BEGIN

BINT91 ClrSysFlag ( ob ) ( 91 MTRW: no list of list )

DUPTYPEBINT? ( ob flag )

OVER ( ob flag ob )

TYPECOL? ( ob flag flag' )

OR ( ob flag'' )

ITE

:: DROP ( )

FLASHPTR RunDoNewMatrix ( ob' T // F ) ( abre MTRW )

;

FLASHPTR RunDoOldMatrix ( ob' T // F ) ( edita ob en MTRW )

ITE

:: DUPTYPELIST? ( ob' flag )

ITE

FALSE ( ob' F )

TrueTrue ( ob' T T )

;

FalseTrue

UNTIL

( RealArry/CArry/MATRIX T // F )

;

;

\* Busca una FORMACIÓN en el FILER

\* Inicia en el directorio HOME

\* Se puede desplazar por todos lados

\* No permite selección múltiple

\* Solo retorna una formación seguida de TRUE.

\* Si se cancela, sólo retorna FALSE.

NULLNAME Filer\_Home\_S1\_NoDir\_FORMACION ( -> ob T // F )

::

{ # 29E8 # 2686 # 2A96 } ( Filer\_Tipos: ARRAY, MATRIX y Directorios )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* HAY 6 TECLAS DEL MENU

\* La 1º es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* Esta lista es para quitar la tecla +/- "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT28 ( Tecla\_Atajo: BINT28 = #1C = Tecla +/- "CHECK" )

}

\* Esta lista es para quitar la tecla ALPHA + ENTER "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 133 ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista no hace nda

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

}

\* Esta lista es para la tecla de menú F5 "CANCL"

{ # DF2C ( Nombre: # DF2C "CANCL" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT37 ( Acción: BINT37= Sale del filer dejando FALSE en la pila )

}

\* Esta lista es para la tecla de menú F6 "OK"

\* También es para asignarla a la tecla ENTER

{ # DF2D ( Nombre: # DF2D "OK" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT17 ( Acción: BINT17= Prog Pers que pone Ruta, Objeto y Nombre )

:: ( Ruta Objeto Nombre )

DROPSWAPDROP ( Objeto )

DUPTYPERRP? ( Objeto flag ) ( Evita finalizar con Directorio )

caseDrpBadKy ( )

' TakeOver ( Objeto TakeOver )

; ( Programa\_Extra: Sale del filer, deja en la pila el Objeto y TRUE )

# 33 ( Tecla\_Atajo: BINT51 = #33 = Tecla ENTER )

}

}

FLASHPTR FILER\_MANAGERTYPE ( ob T // F // :0:{} )

TRUE

EQUAL ( ob T // F )

;

\* Mueve el enfoque al campo especificado

\* Actualiza visualización de campos (invierte píxeles).

\* Actualiza la zona de la ayuda.

NULLNAME ChangeFocus3 ( #c -> )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 097 ( #c ) ( cambia el enfoque e invierte píxeles )

TRUESWAP\_ ( T #c )

ROMPTR 0B0 017 ( ) ( muestra ayuda del campo justo encima de menús )

;

**Ejemplo 35 DoInputForm**

**Campo combofiler con un arreglo real.**

**Manejando los mensajes 9, 23, 37 y 29.**

En este ejemplo hay un campo COMBOFILER que debe contener a un arreglo real.

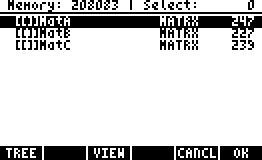
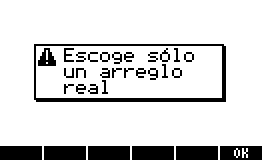
Con el mensaje 9, podemos ver la formación como un grob.

Con el mensaje 23, podemos editar la formación en el MTRW al presionar EDIT.

Con el mensaje 37, evitaremos que en el filer sean escogidas varias formaciones a la vez (selección múltiple). También evitaremos que sean escogidas matrices simbólicas.

Con el mensaje 29, podemos evitar la finalización del formulario (al presionar ENTER u OK), si el campo 2 está vacío.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoArregloReal ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"M:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"A: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO COMBOFILER. CONTIENE A UN ARREGLO REAL

' MH9&23&37\_ArregloReal ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT45 ( #h: 62-#y para llegar hasta abajo )

BINT21 ( #TipoDeCampo: COMBOFILER )

{ BINT3 } ( TiposPermitidos: Arreglos reales, no matrices simbólic )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Escribe arreglo real" ( "Ayuda" )

MINUSONE ( ChooseData en FILER: F=NoCHK TiposP=TiposPC T=RetObj )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo COMBOFILER )

MINUSONE ( ValorReset: campo COMBOFILER en blanco )

MINUSONE ( ValorInicial: campo COMBOFILER en blanco )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

BINT29 #=casedrop ( F ) ( -> flag T // F )

::

BINT2 ( #c )

ROMPTR gFldVal ( ob )

TYPERARRY? ( flag )

NOTcase

:: ( )

BINT2 ( #c )

ChangeFocus3 ( )

"ESCRIBE ARREGLO REAL" ( $ )

FlashWarning ( )

FalseTrue ( F T )

;

TrueTrue ( T T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 T // F )

;

NULLNAME MH9&23&37\_ArregloReal

::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Convierte objeto a grob con ob>grobmini (MiniFuente)

\*\*\* y retorna el grob correspondiente del tamaño del campo

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h ) ( Retorna ancho y alto del campo )

SWAP ( #c #h #b )

MAKEGROB ( #c grobbxh )

OVER ( #c grobbxh #c )

ROMPTR gFldVal ( #c grobbxh valor )

DUP ( #c grobbxh valor valor )

MINUSONE ( #c grobbxh valor valor MINUSONE )

EQUALcase

DROPTRUE ( sale con: #c grobbxh T )

( #c grobbxh valor )

ob>grobmini ( #c grobbxh grob )

OVER ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' ) ( Reempl grob1 en grob2 )

TRUE ( #c grobbxh' T )

;

\* Al presionar la tecla EDIT y su misión es editar el valor del campo

\* Acción por defecto: inicia línea de edic con valor actual del campo

BINT23 #=casedrop ( C ) ( -> T/F )

:: ( )

4GETLAM ( #c )

ROMPTR gFldVal ( valor )

EditaArregloRealEnMTRW ( RealArry T // F )

IT

:: 5GETLAM

ROMPTR sFldVal

;

TRUE ( T )

;

\* Cuando es presionada la tecla CHOOSE en un campos CHOOSE, FILER,

\* COMBOCHOOSE o COMBOFILER.

\* Su misión es presentar las opciones al usuario

\* si una opción es escogida, retornarla en la pila seguida de T T

\* Si el usuario no escoge ninguna opción, en la pila debe ponerse F T

BINT37 #=casedrop ( C ) ( -> ob T T // F T // F )

:: ( )

FALSE ( F )

4GETLAM ( F #c )

\* Retorna los parámetros de ^BrowseMem.1 a partir de ChooseData

\* Puedes cambiar este comando por otro código

ROMPTR 0B0 078 ( F ob2 flagCheck {}tipos flagObjNomb )

SWAP

TOTEMPOB

SWAP ( F ob2 flagCheck {}tipos flagObjNomb )

BrowMem3RA ( ob T // F )

TRUE ( ob T T // F T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

\* CONVIERTE ob A GROB PEQUEÑO USANDO MINIFUENTE

\* Intenta convertir a grob con FLASHPTR EQW3GROBmini

\* Si no es posible hace lo siguiente:

\* Si ob es un grob, no hace nada.

\* Si ob es una cadena, hace $>grobCR

\* Si ob es de otro tipo, hace FLASHPTR FSTR11 $>grobCR

NULLNAME ob>grobmini ( ob -> grob )

:: ( ob )

BEGIN

:: DUP ( ob ob )

FLASHPTR EQW3GROBmini ( ob ext grob #0 // ob ob #1/2 )

BINT0 #=casedrop

:: UNROT2DROP TRUE ; ( OBS: SALE CON grob T ) ( SALE DE BUCLE )

( ob ob #1/2 )

BINT1 #=case

:: GARBAGE DROPFALSE ; ( OBS: SALE CON ob F ) ( REPITE BUCLE )

( ob ob )

TYPEGROB? ( ob flag )

caseTRUE

( ob )

DUPTYPECSTR? ( ob flag )

?SKIP

FLASHPTR FSTR11

( $ )

$>grobCR ( grob )

TRUE ( grob T ) ( SALE DE BUCLE )

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( grob )

;

\* Si en la pila se encuentra xNOVAL o MINUSONE, abre el MTRW.

\* Si hay otro objeto, lo edita en MTRW (de ser posible)

\* Retorna un arreglo real y TRUE o sólo FALSE

NULLNAME EditaArregloRealEnMTRW ( ob -> RealArry T // F )

::

RunSafeFlags

:: BEGIN

BINT91 ClrSysFlag ( ob ) ( 91 MTRW: no list of list )

DUPTYPEBINT? ( ob flag )

OVER ( ob flag ob )

TYPECOL? ( ob flag flag' )

OR ( ob flag'' )

ITE

:: DROP ( )

FLASHPTR DoNewMatrixReal ( ob' T // F ) ( abre MTRW )

;

FLASHPTR DoOldMatrixReal ( ob' T // F ) ( edita ob en MTRW )

( ob' T // F )

ITE

:: DUPTYPELIST? ( ob' flag )

ITE

:: INNERDUP ( ob1...obn #f #f )

ZERO\_DO (DO)

ROLL ( ...obi )

INNERCOMP ( ... ob1'...obm' #c )

TYPEMATRIX\_ ( ... ob1'...obm' #c #2686 )

COMPN\_ ( ... 1DMATRIX )

ISTOP@ ( ... 1DMATRIX #f )

LOOP

TYPEMATRIX\_ ( 1DMATRIX1...1DMATRIXn #f #2686 )

COMPN\_ ( 2DMATRIX1 )

FALSE ( 2DMATRIX1 F )

;

TrueTrue ( ob' T T )

;

FalseTrue

UNTIL

( RealArry T // F )

;

;

\* Parecido al comando FLASHPTR BrowseMem.1

\* Sólo es posible retornar Arreglos reales

\* Sólo es permitido retornar un objeto.

\* De lo contrario, se muestra el mensaje "Escoge sÓlo un arreglo real"

NULLNAME BrowMem3RA ( F ob2 flagCheck {}tipos flagObjNomb -> ob T // F )

:: ( F ob2 flagCheck {}tipos flagObjNomb )

BINT5 NDUP ( ... F ob2 flagCheck {}tipos flagObjNomb )

FLASHPTR BrowseMem.1 ( ... ob T // F // tag )

TRUE

EQUAL ( ... ob T // F )

NOTcase

:: BINT5 NDROPFALSE ; ( sale con: FALSE )

( ... ob )

\* si es una matriz con todos los elementos reales o todos complejos,

\* lo convierte a arreglo. De lo contrario, no hace nada.

FLASHPTR BESTMATRIXTYPE ( ... ob' )

DUP ( ... ob' ob' )

TYPERARRY? ( ... ob flag )

case

:: BINT6 UNROLL ( ob ... )

BINT5 NDROP ( ob )

TRUE ( ob T )

; ( sale con: ob T )

( ... {} )

DROP ( ... )

CLEARVDISP ( ... )

"Escoge sÓlo un arreglo real" ( ... $ )

FlashWarning ( ... )

COLA

BrowMem3RA

;

\* Mueve el enfoque al campo especificado

\* Actualiza visualización de campos (invierte píxeles).

\* Actualiza la zona de la ayuda.

NULLNAME ChangeFocus3 ( #c -> )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 097 ( #c ) ( cambia el enfoque e invierte píxeles )

TRUESWAP\_ ( T #c )

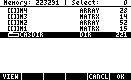
ROMPTR 0B0 017 ( ) ( muestra ayuda del campo justo encima de menús )

;

También se puede usar el siguiente código, en el cual ya no usa el comando

FLASHPTR BrowseMem.1

Ahora se usará el comando FLASHPTR FILER\_MANAGERTYPE



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoArregloReal ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"M:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"A: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO COMBOFILER. CONTIENE A UN ARREGLO REAL

' MH9&23&37\_ArregloReal ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT45 ( #h: 62-#y para llegar hasta abajo )

BINT21 ( #TipoDeCampo: COMBOFILER )

{ BINT3 } ( TiposPermitidos: Arreglos reales, no matrices simbólic )

MINUSONE ( Decompile: No es necesario, pues mensaje 6 es manejado )

"Escribe arreglo real" ( "Ayuda" )

MINUSONE ( ChooseData en FILER: No es necesario, por mensaje 37 )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo COMBOFILER )

MINUSONE ( ValorReset: campo COMBOFILER en blanco )

MINUSONE ( ValorInicial: campo COMBOFILER en blanco )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

BINT29 #=casedrop ( F ) ( -> flag T // F )

::

BINT2 ( #c )

ROMPTR gFldVal ( ob )

TYPERARRY? ( flag )

NOTcase

:: ( )

BINT2 ( #c )

ChangeFocus3 ( )

"ESCRIBE ARREGLO REAL" ( $ )

FlashWarning ( )

FalseTrue ( F T )

;

TrueTrue ( T T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 T // F )

;

NULLNAME MH9&23&37\_ArregloReal

::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Convierte objeto a grob con ob>grobmini (MiniFuente)

\*\*\* y retorna el grob correspondiente del tamaño del campo

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h ) ( Retorna ancho y alto del campo )

SWAP ( #c #h #b )

MAKEGROB ( #c grobbxh )

OVER ( #c grobbxh #c )

ROMPTR gFldVal ( #c grobbxh valor )

DUP ( #c grobbxh valor valor )

MINUSONE ( #c grobbxh valor valor MINUSONE )

EQUALcase

DROPTRUE ( sale con: #c grobbxh T )

( #c grobbxh valor )

ob>grobmini ( #c grobbxh grob )

OVER ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' ) ( Reempl grob1 en grob2 )

TRUE ( #c grobbxh' T )

;

\* Al presionar la tecla EDIT y su misión es editar el valor del campo

\* Acción por defecto: inicia línea de edic con valor actual del campo

BINT23 #=casedrop ( C ) ( -> T/F )

:: ( )

4GETLAM ( #c )

ROMPTR gFldVal ( valor )

EditaArregloRealEnMTRW ( RealArry T // F )

IT

:: 5GETLAM

ROMPTR sFldVal

;

TRUE ( T )

;

\* Cuando es presionada la tecla CHOOSE en un campos CHOOSE, FILER,

\* COMBOCHOOSE o COMBOFILER.

\* Su misión es presentar las opciones al usuario

\* si una opción es escogida, retornarla en la pila seguida de T T

\* Si el usuario no escoge ninguna opción, en la pila debe ponerse F T

BINT37 #=casedrop ( C ) ( -> ob T T // F T // F )

:: ( )

Filer\_Home\_S1\_NoDir\_Arry ( Formacion T // F )

TRUE ( Formacion T T // F T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

\* CONVIERTE ob A GROB PEQUEÑO USANDO MINIFUENTE

\* Intenta convertir a grob con FLASHPTR EQW3GROBmini

\* Si no es posible hace lo siguiente:

\* Si ob es un grob, no hace nada.

\* Si ob es una cadena, hace $>grobCR

\* Si ob es de otro tipo, hace FLASHPTR FSTR11 $>grobCR

NULLNAME ob>grobmini ( ob -> grob )

:: ( ob )

BEGIN

:: DUP ( ob ob )

FLASHPTR EQW3GROBmini ( ob ext grob #0 // ob ob #1/2 )

BINT0 #=casedrop

:: UNROT2DROP TRUE ; ( OBS: SALE CON grob T ) ( SALE DE BUCLE )

( ob ob #1/2 )

BINT1 #=case

:: GARBAGE DROPFALSE ; ( OBS: SALE CON ob F ) ( REPITE BUCLE )

( ob ob )

TYPEGROB? ( ob flag )

caseTRUE

( ob )

DUPTYPECSTR? ( ob flag )

?SKIP

FLASHPTR FSTR11

( $ )

$>grobCR ( grob )

TRUE ( grob T ) ( SALE DE BUCLE )

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( grob )

;

\* Si en la pila se encuentra xNOVAL o MINUSONE, abre el MTRW.

\* Si hay otro objeto, lo edita en MTRW (de ser posible)

\* Retorna un arreglo real y TRUE o sólo FALSE

NULLNAME EditaArregloRealEnMTRW ( ob -> RealArry T // F )

::

RunSafeFlags

:: BEGIN

BINT91 ClrSysFlag ( ob ) ( 91 MTRW: no list of list )

DUPTYPEBINT? ( ob flag )

OVER ( ob flag ob )

TYPECOL? ( ob flag flag' )

OR ( ob flag'' )

ITE

:: DROP ( )

FLASHPTR DoNewMatrixReal ( ob' T // F ) ( abre MTRW )

;

FLASHPTR DoOldMatrixReal ( ob' T // F ) ( edita ob en MTRW )

( ob' T // F )

ITE

:: DUPTYPELIST? ( ob' flag )

ITE

:: INNERDUP ( ob1...obn #f #f )

ZERO\_DO (DO)

ROLL ( ...obi )

INNERCOMP ( ... ob1'...obm' #c )

TYPEMATRIX\_ ( ... ob1'...obm' #c #2686 )

COMPN\_ ( ... 1DMATRIX )

ISTOP@ ( ... 1DMATRIX #f )

LOOP

TYPEMATRIX\_ ( 1DMATRIX1...1DMATRIXn #f #2686 )

COMPN\_ ( 2DMATRIX1 )

FALSE ( 2DMATRIX1 F )

;

TrueTrue ( ob' T T )

;

FalseTrue

UNTIL

( RealArry T // F )

;

;

\* Busca un ARREGLO en el FILER

\* Inicia en el directorio HOME

\* Se puede desplazar por todos lados

\* No permite selección múltiple

\* Solo retorna un arreglo real seguido de TRUE.

\* Si se cancela, sólo retorna FALSE.

NULLNAME Filer\_Home\_S1\_NoDir\_Arry ( -> ob T // F )

::

{ # 29E8 # 2A96 } ( Filer\_Tipos: ARRAY y Directorios )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* HAY 6 TECLAS DEL MENU

\* La 1º es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* Esta lista es para quitar la tecla +/- "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT28 ( Tecla\_Atajo: BINT28 = #1C = Tecla +/- "CHECK" )

}

\* Esta lista es para quitar la tecla ALPHA + ENTER "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 133 ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista no hace nada

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

}

\* Esta lista es para la tecla de menú F5 "CANCL"

{ # DF2C ( Nombre: # DF2C "CANCL" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT37 ( Acción: BINT37= Sale del filer dejando FALSE en la pila )

}

\* Esta lista es para la tecla de menú F6 "OK"

\* También es para asignarla a la tecla ENTER

{ # DF2D ( Nombre: # DF2D "OK" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT17 ( Acción: BINT17= Prog Pers que pone Ruta, Objeto y Nombre )

:: ( Ruta Objeto Nombre )

DROPSWAPDROP ( Objeto )

DUPTYPERRP? ( Objeto flag ) ( Evita finalizar con Directorio )

caseDrpBadKy ( )

DUP ( Objeto Objeto )

TYPERARRY? ( Objeto flag )

NOTcasedrop

:: "Escoge un arreglo real"

FlashWarning

; ( )

( RealArry )

' TakeOver ( RealArry TakeOver )

; ( Programa\_Extra: Sale del filer, deja en la pila el Objeto y TRUE )

# 33 ( Tecla\_Atajo: BINT51 = #33 = Tecla ENTER )

}

}

FLASHPTR FILER\_MANAGERTYPE ( ob T // F // :0:{} )

TRUE

EQUAL ( ob T // F )

;

\* Mueve el enfoque al campo especificado

\* Actualiza visualización de campos (invierte píxeles).

\* Actualiza la zona de la ayuda.

NULLNAME ChangeFocus3 ( #c -> )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 097 ( #c ) ( cambia el enfoque e invierte píxeles )

TRUESWAP\_ ( T #c )

ROMPTR 0B0 017 ( ) ( muestra ayuda del campo justo encima de menús )

;

**Ejemplo 36 DoInputForm**

**Campo texto con un arreglo real que cumpla una condición.**

**Manejando los mensajes 9, 23, 37, 46 y 29.**

Aquí hay un campo COMBOFILER que contendrá un arreglo real que cumpla una condición.

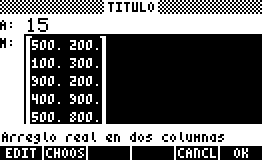
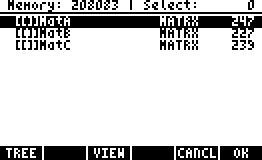
Con el mensaje 9, podemos ver la formación como un grob.

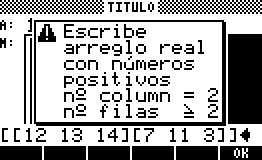
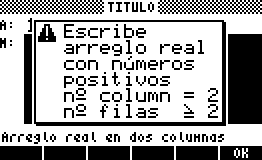
Con el mensaje 23, podemos editar la formación en el MTRW al presionar EDIT.

Con el mensaje 37, evitaremos que en el filer sean escogidas varias formaciones a la vez (selección múltiple). También evitaremos que sean escogidas arreglos inválidos o matrices.

Con el mensaje 46, podemos validar o no la entrada escrita en la línea de comandos.

Con el mensaje 29, podemos evitar la finalización del formulario (al presionar ENTER u OK), si el campo 2 está vacío.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoRACond ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"M:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"A: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO COMBOFILER. CONTIENE A UN ARREGLO REAL

' MH9&23&37&46\_ArregoRealCond ( Message Handler del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT45 ( #h: 62-#y para llegar hasta abajo )

BINT21 ( #TipoDeCampo: COMBOFILER )

{ BINT3 } ( TiposPermitidos: Arreglos reales, no matrices simbólic )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"Escribe arreglo real 2 columnas" ( "Ayuda" )

MINUSONE ( ChooseData en FILER: F=NoCHK TiposP=TiposPC T=RetObj )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo COMBOFILER )

MINUSONE ( ValorReset: campo COMBOFILER en blanco )

MINUSONE ( ValorInicial: campo COMBOFILER en blanco )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

BINT29 #=casedrop ( F ) ( -> flag T // F )

::

BINT2 ( #c )

ROMPTR gFldVal ( ob )

TestRealArry ( flag )

NOTcase

:: ( )

BINT2 ( #c )

ChangeFocus3 ( )

AlertaArregloRealCond ( )

FalseTrue ( F T )

;

TrueTrue ( T T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 T // F )

;

NULLNAME MH9&23&37&46\_ArregoRealCond

::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Convierte objeto a grob con ob>grobmini (MiniFuente)

\*\*\* y retorna el grob correspondiente del tamaño del campo

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h ) ( Retorna ancho y alto del campo )

SWAP ( #c #h #b )

MAKEGROB ( #c grobbxh )

OVER ( #c grobbxh #c )

ROMPTR gFldVal ( #c grobbxh valor )

DUP ( #c grobbxh valor valor )

MINUSONE ( #c grobbxh valor valor MINUSONE )

EQUALcase

DROPTRUE ( sale con: #c grobbxh T )

( #c grobbxh valor )

ob>grobmini ( #c grobbxh grob )

OVER ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' ) ( Reempl grob1 en grob2 )

TRUE ( #c grobbxh' T )

;

\* Al presionar la tecla EDIT y su misión es editar el valor del campo

\* Acción por defecto: inicia línea de edic con valor actual del campo

BINT23 #=casedrop ( -> T/F )

:: ( )

4GETLAM ( #c )

ROMPTR gFldVal ( valor )

BEGIN

:: EditaArregloRealEnMTRW ( RealArry T // F )

NOTcase

FalseTrue ( sale con: F T )

( ob' )

DUP ( ob' ob' )

TestRealArry ( ob' flag ) ( test al objeto de la pila )

NOTcase

:: AlertaArregloRealCond ( ob' )

FALSE ( ob' F )

;

( [[%]] )

TrueTrue ( [[%]] T T )

;

UNTIL

( [[%]] T // F )

IT

:: 5GETLAM ROMPTR sFldVal ;

TRUE ( T )

;

\* Cuando es presionada la tecla CHOOSE en un campos CHOOSE, FILER,

\* COMBOCHOOSE o COMBOFILER.

\* Su misión es presentar las opciones al usuario

\* si una opción es escogida, retornarla en la pila seguida de T T

\* Si el usuario no escoge ninguna opción, en la pila debe ponerse F T

BINT37 #=casedrop ( C ) ( -> ob T T // F T // F )

:: ( )

FALSE ( F )

4GETLAM ( F #c )

\* Retorna los parámetros de ^BrowseMem.1 a partir de ChooseData

\* Puedes cambiar este comando por otro código

ROMPTR 0B0 078 ( F ob2 flagCheck {}tipos flagObjNomb )

SWAP

TOTEMPOB

SWAP ( F ob2 flagCheck {}tipos flagObjNomb )

BrowMem3RACond ( ob T // F )

TRUE ( ob T T // F T )

;

\* Cuando se quiere confirmar línea de edición con ENTER u OK

\* o cuando se quiere asignar valor al campo desde la tecla CALC

\* Su misión es verificar que el objeto tenga un valor válido.

\* Con TRUE TRUE, el obj será asignado al campo (similar sin efecto)

\* Con FALSE TRUE, el obj no se asignará al campo y se mostrará el

\* mensaje "Invalid object value".

BINT46 #=casedrop ( valor -> F T // valor F )

:: ( valor )

DUP ( valor )

TestRealArry ( valor flag )

caseFALSE ( sale con: % FALSE )

( valor )

DROP ( )

AlertaArregloRealCond ( )

FalseTrue ( F T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

\* CONVIERTE ob A GROB PEQUEÑO USANDO MINIFUENTE

\* Intenta convertir a grob con FLASHPTR EQW3GROBmini

\* Si no es posible hace lo siguiente:

\* Si ob es un grob, no hace nada.

\* Si ob es una cadena, hace $>grobCR

\* Si ob es de otro tipo, hace FLASHPTR FSTR11 $>grobCR

NULLNAME ob>grobmini ( ob -> grob )

:: ( ob )

BEGIN

:: DUP ( ob ob )

FLASHPTR EQW3GROBmini ( ob ext grob #0 // ob ob #1/2 )

BINT0 #=casedrop

:: UNROT2DROP TRUE ; ( OBS: SALE CON grob T ) ( SALE DE BUCLE )

( ob ob #1/2 )

BINT1 #=case

:: GARBAGE DROPFALSE ; ( OBS: SALE CON ob F ) ( REPITE BUCLE )

( ob ob )

TYPEGROB? ( ob flag )

caseTRUE

( ob )

DUPTYPECSTR? ( ob flag )

?SKIP

FLASHPTR FSTR11

( $ )

$>grobCR ( grob )

TRUE ( grob T ) ( SALE DE BUCLE )

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( grob )

;

\* Si en la pila se encuentra xNOVAL o MINUSONE, abre el MTRW.

\* Si hay otro objeto, lo edita en MTRW (de ser posible)

\* Retorna un arreglo real y TRUE o sólo FALSE

NULLNAME EditaArregloRealEnMTRW ( ob -> RealArry T // F )

::

RunSafeFlags

:: BEGIN

BINT91 ClrSysFlag ( ob ) ( 91 MTRW: no list of list )

DUPTYPEBINT? ( ob flag )

OVER ( ob flag ob )

TYPECOL? ( ob flag flag' )

OR ( ob flag'' )

ITE

:: DROP ( )

FLASHPTR DoNewMatrixReal ( ob' T // F ) ( abre MTRW )

;

FLASHPTR DoOldMatrixReal ( ob' T // F ) ( edita ob en MTRW )

( ob' T // F )

ITE

:: DUPTYPELIST? ( ob' flag )

ITE

:: INNERDUP ( ob1...obn #f #f )

ZERO\_DO (DO)

ROLL ( ...obi )

INNERCOMP ( ... ob1'...obm' #c )

TYPEMATRIX\_ ( ... ob1'...obm' #c #2686 )

COMPN\_ ( ... 1DMATRIX )

ISTOP@ ( ... 1DMATRIX #f )

LOOP

TYPEMATRIX\_ ( 1DMATRIX1...1DMATRIXn #f #2686 )

COMPN\_ ( 2DMATRIX1 )

FALSE ( 2DMATRIX1 F )

;

TrueTrue ( ob' T T )

;

FalseTrue

UNTIL

( RealArry T // F )

;

;

\* Parecido al comando FLASHPTR BrowseMem.1

\* Sólo es posible retornar Arreglos reales que cumplan unas condiciones

\* Sólo es permitido retornar un objeto.

\* De lo contrario, se muestra el mensaje "Escribe..."

NULLNAME BrowMem3RACond ( F ob2 flagCheck {}tipos flagObjNomb -> ob T // F )

:: ( F ob2 flagCheck {}tipos flagObjNomb )

BINT5 NDUP ( ... F ob2 flagCheck {}tipos flagObjNomb )

FLASHPTR BrowseMem.1 ( ... ob T // F // tag )

TRUE

EQUAL ( ... ob T // F )

NOTcase :: BINT5 NDROPFALSE ; ( sale con FALSE )

( ... ob )

\* si es una matriz con todos los elementos reales o todos complejos,

\* lo convierte a arreglo. De lo contrario, no hace nada.

FLASHPTR BESTMATRIXTYPE ( ... ob' )

DUP ( ... ob' ob' )

TestRealArry ( ... ob flag )

case

:: BINT6 UNROLL ( ob ... )

BINT5 NDROP ( ob )

TRUE ( ob T )

; ( sale con ob T )

( ... {} )

DROP ( ... )

CLEARVDISP ( ... )

AlertaArregloRealCond ( ... )

COLA

BrowMem3RACond

;

\* Realiza un test al objeto del nivel uno de la pila

\* Retorna TRUE si en la pila hay un arreglo real con números positivos, de

\* 2 dimensiones con 2 columnas y con un número de filas mayor o igual a 2.

NULLNAME TestRealArry ( ob -> flag )

::

DUP TYPERARRY? ( ob flag )

NOTcase

DROPFALSE

( RealArry )

DUP ( RealArry RealArry )

FLASHPTR MDIMS ( [[%]] #filas #cols T // [%] #elem F )

NOTcase

2DROPFALSE

( [[%]] #filas #cols )

#2= ( [[%]] #filas flag )

SWAP ( [[%]] flag #filas )

BINT1 #> ( [[%]] flag flag' )

AND ( [[%]] flag'' )

NOTcase

DROPFALSE

( [[%]] ) ( arreglo real de 2 dimensiones en la pila )

FLASHPTR XEQARRY> ( %1...%n {%f %c} )

INCOMPDROP ( %1...%n %f %c )

%\* ( %1...%n %foc )

COERCE ( %1...%n #foc )

ONE\_DO (DO) %MIN LOOP

( %' )

%0> ( flag )

;

\* Mueve el enfoque al campo especificado

\* Actualiza visualización de campos (invierte píxeles).

\* Actualiza la zona de la ayuda.

NULLNAME ChangeFocus3 ( #c -> )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 097 ( #c ) ( cambia el enfoque e invierte píxeles )

TRUESWAP\_ ( T #c )

ROMPTR 0B0 017 ( ) ( muestra ayuda del campo justo encima de menús )

;

NULLNAME AlertaArregloRealCond ( -> )

::

"Escribe arreglo real con números positivos\0Anº column = 2\0Anº filas Š 2"

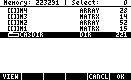
FlashWarning

;

También se puede usar el siguiente código, en el cual ya no usa el comando

FLASHPTR BrowseMem.1

Ahora se usará el comando FLASHPTR FILER\_MANAGERTYPE



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoRACond ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

"A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"M:" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"A: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO Nº #2. ES UN CAMPO COMBOFILER. CONTIENE A UN ARREGLO REAL

' MH9&23&37&43\_ArregoRealCond ( Message Handler del campo )

BINT9 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT122 ( #w: 131-#x para 1 columna )

BINT45 ( #h: 62-#y para llegar hasta abajo )

BINT21 ( #TipoDeCampo: COMBOFILER )

{ BINT3 } ( TiposPermitidos: Arreglos reales, no matrices simbólic )

MINUSONE ( Decompile: No es necesario, pues mensaje 6 es manejado )

"Escribe arreglo real 2 columnas" ( "Ayuda" )

MINUSONE ( ChooseData en FILER: No es necesario, por mensaje 37 )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo COMBOFILER )

MINUSONE ( ValorReset: campo COMBOFILER en blanco )

MINUSONE ( ValorInicial: campo COMBOFILER en blanco )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

BINT29 #=casedrop ( F ) ( -> flag T // F )

::

BINT2 ( #c )

ROMPTR gFldVal ( ob )

TestRealArry ( flag )

NOTcase

:: ( )

BINT2 ( #c )

ChangeFocus3 ( )

AlertaArregloRealCond ( )

FalseTrue ( F T )

;

TrueTrue ( T T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 T // F )

;

NULLNAME MH9&23&37&43\_ArregoRealCond

::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Convierte objeto a grob con ob>grobmini (MiniFuente)

\*\*\* y retorna el grob correspondiente del tamaño del campo

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h ) ( Retorna ancho y alto del campo )

SWAP ( #c #h #b )

MAKEGROB ( #c grobbxh )

OVER ( #c grobbxh #c )

ROMPTR gFldVal ( #c grobbxh valor )

DUP ( #c grobbxh valor valor )

MINUSONE ( #c grobbxh valor valor MINUSONE )

EQUALcase

DROPTRUE ( sale con: #c grobbxh T )

( #c grobbxh valor )

ob>grobmini ( #c grobbxh grob )

OVER ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' ) ( Reempl grob1 en grob2 )

TRUE ( #c grobbxh' T )

;

\* Al presionar la tecla EDIT y su misión es editar el valor del campo

\* Acción por defecto: inicia línea de edic con valor actual del campo

BINT23 #=casedrop ( -> T/F )

:: ( )

4GETLAM ( #c )

ROMPTR gFldVal ( valor )

BEGIN

:: EditaArregloRealEnMTRW ( RealArry T // F )

NOTcase

FalseTrue ( sale con: F T )

( ob' )

DUP ( ob' ob' )

TestRealArry ( ob' flag ) ( test al objeto de la pila )

NOTcase

:: AlertaArregloRealCond ( ob' )

FALSE ( ob' F )

;

( [[%]] )

TrueTrue ( [[%]] T T )

;

UNTIL

( [[%]] T // F )

IT

:: 5GETLAM ROMPTR sFldVal ;

TRUE ( T )

;

\* Cuando es presionada la tecla CHOOSE en un campos CHOOSE, FILER,

\* COMBOCHOOSE o COMBOFILER.

\* Su misión es presentar las opciones al usuario

\* si una opción es escogida, retornarla en la pila seguida de T T

\* Si el usuario no escoge ninguna opción, en la pila debe ponerse F T

BINT37 #=casedrop ( C ) ( -> ob T T // F T // F )

:: ( )

Filer\_Home\_S1\_NoDir\_ArryCond ( Formacion T // F )

TRUE ( Formacion T T // F T )

;

\* Cuando se quiere confirmar línea de edición con ENTER u OK

\* o cuando se quiere asignar valor al campo desde la tecla CALC

\* Su misión es verificar que el objeto tenga un valor válido.

\* Con TRUE TRUE, el obj será asignado al campo (similar sin efecto)

\* Con FALSE TRUE, el obj no se asignará al campo y se mostrará el

\* mensaje "Invalid object value".

BINT46 #=casedrop ( valor -> F T // valor F )

:: ( valor )

DUP ( valor )

TestRealArry ( valor flag )

caseFALSE ( sale con: % FALSE )

( valor )

DROP ( )

AlertaArregloRealCond ( )

FalseTrue ( F T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

\* CONVIERTE ob A GROB PEQUEÑO USANDO MINIFUENTE

\* Intenta convertir a grob con FLASHPTR EQW3GROBmini

\* Si no es posible hace lo siguiente:

\* Si ob es un grob, no hace nada.

\* Si ob es una cadena, hace $>grobCR

\* Si ob es de otro tipo, hace FLASHPTR FSTR11 $>grobCR

NULLNAME ob>grobmini ( ob -> grob )

:: ( ob )

BEGIN

:: DUP ( ob ob )

FLASHPTR EQW3GROBmini ( ob ext grob #0 // ob ob #1/2 )

BINT0 #=casedrop

:: UNROT2DROP TRUE ; ( OBS: SALE CON grob T ) ( SALE DE BUCLE )

( ob ob #1/2 )

BINT1 #=case

:: GARBAGE DROPFALSE ; ( OBS: SALE CON ob F ) ( REPITE BUCLE )

( ob ob )

TYPEGROB? ( ob flag )

caseTRUE

( ob )

DUPTYPECSTR? ( ob flag )

?SKIP

FLASHPTR FSTR11

( $ )

$>grobCR ( grob )

TRUE ( grob T ) ( SALE DE BUCLE )

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( grob )

;

\* Si en la pila se encuentra xNOVAL o MINUSONE, abre el MTRW.

\* Si hay otro objeto, lo edita en MTRW (de ser posible)

\* Retorna un arreglo real y TRUE o sólo FALSE

NULLNAME EditaArregloRealEnMTRW ( ob -> RealArry T // F )

::

RunSafeFlags

:: BEGIN

BINT91 ClrSysFlag ( ob ) ( 91 MTRW: no list of list )

DUPTYPEBINT? ( ob flag )

OVER ( ob flag ob )

TYPECOL? ( ob flag flag' )

OR ( ob flag'' )

ITE

:: DROP ( )

FLASHPTR DoNewMatrixReal ( ob' T // F ) ( abre MTRW )

;

FLASHPTR DoOldMatrixReal ( ob' T // F ) ( edita ob en MTRW )

( ob' T // F )

ITE

:: DUPTYPELIST? ( ob' flag )

ITE

:: INNERDUP ( ob1...obn #f #f )

ZERO\_DO (DO)

ROLL ( ...obi )

INNERCOMP ( ... ob1'...obm' #c )

TYPEMATRIX\_ ( ... ob1'...obm' #c #2686 )

COMPN\_ ( ... 1DMATRIX )

ISTOP@ ( ... 1DMATRIX #f )

LOOP

TYPEMATRIX\_ ( 1DMATRIX1...1DMATRIXn #f #2686 )

COMPN\_ ( 2DMATRIX1 )

FALSE ( 2DMATRIX1 F )

;

TrueTrue ( ob' T T )

;

FalseTrue

UNTIL

( RealArry T // F )

;

;

\* Busca un ARREGLO REAL en el FILER que cumpla el test: TestRealArry

\* Inicia en el directorio HOME

\* Se puede desplazar por todos lados

\* No permite selección múltiple

\* Solo retorna el arreglo real que cumpla condición seguido de TRUE

\* Si se cancela, sólo retorna FALSE.

NULLNAME Filer\_Home\_S1\_NoDir\_ArryCond ( -> ob T // F )

::

{ # 29E8 # 2A96 } ( Filer\_Tipos: ARRAY y Directorios )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* HAY 6 TECLAS DEL MENU

\* La 1º es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* Esta lista es para quitar la tecla +/- "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT28 ( Tecla\_Atajo: BINT28 = #1C = Tecla +/- "CHECK" )

}

\* Esta lista es para quitar la tecla ALPHA + ENTER "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 133 ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista no hace nada

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

}

\* Esta lista es para la tecla de menú F5 "CANCL"

{ # DF2C ( Nombre: # DF2C "CANCL" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT37 ( Acción: BINT37= Sale del filer dejando FALSE en la pila )

}

\* Esta lista es para la tecla de menú F6 "OK"

\* También es para asignarla a la tecla ENTER

{ # DF2D ( Nombre: # DF2D "OK" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT17 ( Acción: BINT17= Prog Pers que pone Ruta, Objeto y Nombre )

:: ( Ruta Objeto Nombre )

DROPSWAPDROP ( Objeto )

DUPTYPERRP? ( Objeto flag ) ( Evita finalizar con Directorio )

caseDrpBadKy ( )

DUP ( Objeto Objeto )

TestRealArry ( Objeto flag )

NOTcasedrop

AlertaArregloRealCond ( )

( Objeto )

' TakeOver ( Objeto TakeOver )

; ( Programa\_Extra: Sale del filer, deja en la pila el Objeto y TRUE )

# 33 ( Tecla\_Atajo: BINT51 = #33 = Tecla ENTER )

}

}

FLASHPTR FILER\_MANAGERTYPE ( ob T // F // :0:{} )

TRUE

EQUAL ( ob T // F )

;

\* Realiza un test al objeto del nivel uno de la pila

\* Retorna TRUE si en la pila hay un arreglo real con números positivos, de

\* 2 dimensiones con 2 columnas y con un número de filas mayor o igual a 2.

NULLNAME TestRealArry ( ob -> flag )

::

DUP TYPERARRY? ( ob flag )

NOTcase

DROPFALSE

( RealArry )

DUP ( RealArry RealArry )

FLASHPTR MDIMS ( [[%]] #filas #cols T // [%] #elem F )

NOTcase

2DROPFALSE

( [[%]] #filas #cols )

#2= ( [[%]] #filas flag )

SWAP ( [[%]] flag #filas )

BINT1 #> ( [[%]] flag flag' )

AND ( [[%]] flag'' )

NOTcase

DROPFALSE

( [[%]] ) ( arreglo real de 2 dimensiones en la pila )

FLASHPTR XEQARRY> ( %1...%n {%f %c} )

INCOMPDROP ( %1...%n %f %c )

%\* ( %1...%n %foc )

COERCE ( %1...%n #foc )

ONE\_DO (DO) %MIN LOOP

( %' )

%0> ( flag )

;

\* Mueve el enfoque al campo especificado

\* Actualiza visualización de campos (invierte píxeles).

\* Actualiza la zona de la ayuda.

NULLNAME ChangeFocus3 ( #c -> )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 097 ( #c ) ( cambia el enfoque e invierte píxeles )

TRUESWAP\_ ( T #c )

ROMPTR 0B0 017 ( ) ( muestra ayuda del campo justo encima de menús )

;

NULLNAME AlertaArregloRealCond ( -> )

::

"Escribe arreglo real con números positivos\0Anº column = 2\0Anº filas Š 2"

FlashWarning

;

**Ejemplo 37 DoInputForm**

**Campo texto con un arreglo real que cumpla una condición.**

**Manejando los mensajes 9, 23, 37, 46 y 29.**

Aquí hay un campo COMBOFILER que contendrá un arreglo real que cumpla una condición.

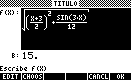
Con el mensaje 9, podemos ver la formación como un grob.

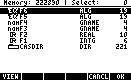
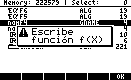
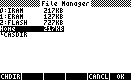
Con el mensaje 23, podemos editar la formación en el MTRW al presionar EDIT.

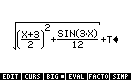
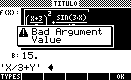
Con el mensaje 37, evitaremos que en el filer sean escogidas varias formaciones a la vez (selección múltiple). También evitaremos que sean escogidas funciones que tengan variables distintas a X.

Con el mensaje 46, podemos validar o no la entrada escrita en la línea de comandos.

Con el mensaje 29, podemos evitar la finalización del formulario (al presionar ENTER u OK), si el campo 2 está vacío.



ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME DoFuncCond ( -> ob1 ob2 T // F )

:: CK0 ( ) ( No se requieren argumentos )

\* Definiciones de etiquetas

" A:" BINT0 BINT10 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

"f(X):" BINT0 BINT19 ( #xetiq=0 COL1 ) ( #yetiq=#ycampo+2 )

\* CAMPO 1: ES UN CAMPO TEXTO. CONTIENE A UN NÚMERO REAL.

'DROPFALSE ( MH del campo )

BINT21 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT8 ( #y: 8 para fila 1/6 )

BINT110 ( #w: 131-#x para 1 columna )

BINT9 ( #h: 9 para SysFont )

BINT1 ( #TipoDeCampo: TEXTO )

{ BINT0 } ( TiposPermitidos: Reales )

BINT4 ( Decompile: 4=STD ) ( ) ( "$" id u )

"A: número real" ( Ayuda )

MINUSONE ( ChooseData: MINUSONE siempre en campo TEXTO )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo TEXTO )

%15 ( ValorReset )

DUP ( ValorInicial: El mismo que ValorReset )

\* CAMPO 2: ES UN CAMPO COMBOFILER. CONTIENE A UNA FUNCIÓN

' MH9&23&37&46\_FuncCond ( Message Handler del campo )

BINT21 ( #x: #xetiq + 4\*#ncetiq + 1 )

BINT17 ( #y: 17 para fila 2/6 )

BINT110 ( #w: 131-#x para 1 columna )

BINT45 ( #h: 62-#y para llegar hasta abajo )

BINT21 ( #TipoDeCampo: COMBOFILER )

{ BINT0 BINT6 BINT9 } ( TiposPermitidos: %, id, symb )

MINUSONE ( Decompile: No es necesario, pues mensaje 6 es manejado )

"Escribe función f(X)" ( "Ayuda" )

MINUSONE ( ChooseData en FILER: No es necesario, por mensaje 37 )

MINUSONE ( ChooseDecompile: MINUSONE siempre en campo COMBOFILER )

MINUSONE ( ValorReset: campo COMBOFILER en blanco )

MINUSONE ( ValorInicial: campo COMBOFILER en blanco )

BINT2 ( #Netiq )

BINT2 ( #Ncamp )

\* Message handler del formulario

' ::

\* Este mensaje es llamado cuando se presiona la tecla ENTER u OK

\* Puede evitar que el formulario sea finalizado

BINT29 #=casedrop ( F ) ( -> flag T // F )

::

BINT2 ( #c )

ROMPTR gFldVal ( ob )

TestFuncion ( flag )

NOTcase

:: ( )

BINT2 ( #c )

ChangeFocus3 ( )

AlertaFuncCond ( )

FalseTrue ( F T )

;

TrueTrue ( T T )

;

\* Fin del Message handler del formulario:

DROPFALSE

;

"TITULO" ( Titulo del DoInputForm )

DoInputForm ( ob1 ob2 T // F )

;

NULLNAME MH9&23&37&46\_FuncCond

::

\* Este mensaje es llamado cuando se va a dibujar un campo

\* Debe retornar el valor del campo convertido en grob.

\*\*\* CARACTERÍSTICAS DE ESTE CÓDIGO:

\*\*\* Convierte objeto a grob con ob>grobmini (MiniFuente)

\*\*\* y retorna el grob correspondiente del tamaño del campo

BINT9 #=casedrop ( C ) ( #c -> #c grob T // #c F )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 0CA ( #c #b #h ) ( Retorna ancho y alto del campo )

SWAP ( #c #h #b )

MAKEGROB ( #c grobbxh )

OVER ( #c grobbxh #c )

ROMPTR gFldVal ( #c grobbxh valor )

DUP ( #c grobbxh valor valor )

MINUSONE ( #c grobbxh valor valor MINUSONE )

EQUALcase

DROPTRUE ( sale con: #c grobbxh T )

( #c grobbxh valor )

ob>grobmini ( #c grobbxh grob )

OVER ( #c grobbxh grob grobbxh )

ONEONE ( #c grobbxh grob grobbxh #1 #1 )

GROB! ( #c grobbxh' ) ( Reempl grob1 en grob2 )

TRUE ( #c grobbxh' T )

;

\* Al presionar la tecla EDIT y su misión es editar el valor del campo

\* Acción por defecto: inicia línea de edic con valor actual del campo

BINT23 #=casedrop ( -> T/F )

:: ( )

4GETLAM ( #c )

ROMPTR gFldVal ( valor )

BEGIN

:: EditaEnEQW ( RealArry T // F )

NOTcase

FalseTrue ( sale con: F T )

( ob' )

DUP ( ob' ob' )

TestFuncion ( ob' flag ) ( test al objeto de la pila )

NOTcase

:: AlertaFuncCond ( ob' )

FALSE ( ob' F )

;

( [[%]] )

TrueTrue ( [[%]] T T )

;

UNTIL

( [[%]] T // F )

IT

:: 5GETLAM ROMPTR sFldVal ;

TRUE ( T )

;

\* Cuando es presionada la tecla CHOOSE en un campos CHOOSE, FILER,

\* COMBOCHOOSE o COMBOFILER.

\* Su misión es presentar las opciones al usuario

\* si una opción es escogida, retornarla en la pila seguida de T T

\* Si el usuario no escoge ninguna opción, en la pila debe ponerse F T

BINT37 #=casedrop ( C ) ( -> ob T T // F T // F )

:: ( )

Filer\_Home\_S1\_NoDir\_FuncCond ( Formacion T // F )

TRUE ( Formacion T T // F T )

;

\* Cuando se quiere confirmar línea de edición con ENTER u OK

\* o cuando se quiere asignar valor al campo desde la tecla CALC

\* Su misión es verificar que el objeto tenga un valor válido.

\* Con TRUE TRUE, el obj será asignado al campo (similar sin efecto)

\* Con FALSE TRUE, el obj no se asignará al campo y se mostrará el

\* mensaje "Invalid object value".

BINT46 #=casedrop ( valor -> F T // valor F )

:: ( valor )

DUP ( valor )

TestFuncion ( valor flag )

caseFALSE ( sale con: % FALSE )

( valor )

DROP ( )

AlertaFuncCond ( )

FalseTrue ( F T )

;

\* Fin del Message handler del campo:

DROPFALSE

;

\* CONVIERTE ob A GROB PEQUEÑO USANDO MINIFUENTE

\* Intenta convertir a grob con FLASHPTR EQW3GROBmini

\* Si no es posible hace lo siguiente:

\* Si ob es un grob, no hace nada.

\* Si ob es una cadena, hace $>grobCR

\* Si ob es de otro tipo, hace FLASHPTR FSTR11 $>grobCR

NULLNAME ob>grobmini ( ob -> grob )

:: ( ob )

BEGIN

:: DUP ( ob ob )

FLASHPTR EQW3GROBmini ( ob ext grob #0 // ob ob #1/2 )

BINT0 #=casedrop

:: UNROT2DROP TRUE ; ( OBS: SALE CON grob T ) ( SALE DE BUCLE )

( ob ob #1/2 )

BINT1 #=case

:: GARBAGE DROPFALSE ; ( OBS: SALE CON ob F ) ( REPITE BUCLE )

( ob ob )

TYPEGROB? ( ob flag )

caseTRUE

( ob )

DUPTYPECSTR? ( ob flag )

?SKIP

FLASHPTR FSTR11

( $ )

$>grobCR ( grob )

TRUE ( grob T ) ( SALE DE BUCLE )

;

UNTIL ( OBS: NECESITA TRUE PARA SALIR DE BUCLE BEGIN UNTIL )

( grob )

;

\* Si en la pila se encuentra xNOVAL, abre el EQW.

\* Si hay otro objeto, lo edita en EQW (de ser posible)

NULLNAME EditaEnEQW ( ob -> ob' T // F )

:: DUP ( ob ob )

' xNOVAL ( ob ob xNOVAL )

EQUAL ( ob flag )

ITE

:: DROP ( )

FLASHPTR EQW3 ( ob' T // F ) ( abre EQW )

;

FLASHPTR EQW3Edit ( ob' T // F ) ( edita ob en EQW )

;

\* Busca una función f(X) en el FILER que cumpla el test: TestFuncion

\* Inicia en el directorio HOME

\* Se puede desplazar por todos lados

\* No permite selección múltiple

\* Solo retorna una función que cumpla condición seguido de TRUE

\* Si se cancela, sólo retorna FALSE.

NULLNAME Filer\_Home\_S1\_NoDir\_FuncCond ( -> ob T // F )

::

{ # 2933 # 2E48 # 2AB8 # 2614 # 2A96 } ( Filer\_Tipos: %, id, symb, Z, Direct )

NULL{} ( Filer\_RutaInicial: NULL{} = Directorio HOME )

{

\* HAY 6 TECLAS DEL MENU

\* La 1º es para ver el objeto seleccionado

{ # DF25 ( Nombre: # DF25 "VER" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT3 ( Acción: BINT3= Ver objeto )

}

\* Esta lista es para quitar la tecla +/- "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

BINT28 ( Tecla\_Atajo: BINT28 = #1C = Tecla +/- "CHECK" )

}

\* Esta lista es para quitar la tecla ALPHA + ENTER "CHECK"

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

TakeOver ( Programa\_Extra: TakeOver para función ya definida )

# 133 ( Tecla\_Atajo: # 10E = #100 + #E = ALPHA + Tecla izquierda "UPDIR" )

}

\* Esta lista no hace nada

{ NULL$ ( Nombre: NULL$ para que no se vea en pantalla )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT0 ( Acción: BINT0= Hace Beep )

}

\* Esta lista es para la tecla de menú F5 "CANCL"

{ # DF2C ( Nombre: # DF2C "CANCL" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT37 ( Acción: BINT37= Sale del filer dejando FALSE en la pila )

}

\* Esta lista es para la tecla de menú F6 "OK"

\* También es para asignarla a la tecla ENTER

{ # DF2D ( Nombre: # DF2D "OK" )

BINT0 ( Localización: BINT0= En cualquier lugar )

BINT17 ( Acción: BINT17= Prog Pers que pone Ruta, Objeto y Nombre )

:: ( Ruta Objeto Nombre )

DROPSWAPDROP ( Objeto )

DUPTYPERRP? ( Objeto flag ) ( Evita finalizar con Directorio )

caseDrpBadKy ( )

DUP ( Objeto Objeto )

TestFuncion ( Objeto flag )

NOTcasedrop

AlertaFuncCond ( )

( Objeto )

' TakeOver ( Objeto TakeOver )

; ( Programa\_Extra: Sale del filer, deja en la pila el Objeto y TRUE )

# 33 ( Tecla\_Atajo: BINT51 = #33 = Tecla ENTER )

}

}

FLASHPTR FILER\_MANAGERTYPE ( ob T // F // :0:{} )

TRUE

EQUAL ( ob T // F )

;

\* Realiza un test al objeto del nivel uno de la pila

\* Retorna TRUE si en la pila hay una función f(X)

\* Reales, Enteros, ids o symbs que contengan a la variable X

\* o a ninguna variable

NULLNAME TestFuncion ( ob -> flag )

::

DUPTYPEREAL?

case

DROPTRUE

( ob )

DUPTYPEZINT?

case

DROPTRUE

( ob )

DUPTYPESYMB?

OVER

TYPEIDNT?

OR ( ob flag )

NOTcase

DROPFALSE

( ob )

FLASHPTR LIDNText ( {vars} )

DUPNULL{}?

SWAP

{ ID X }

EQUAL

OR ( flag )

;

NULLNAME AlertaFuncCond ( -> )

::

"Escribe función f(X)"

FlashWarning

;

\* Mueve el enfoque al campo especificado

\* Actualiza visualización de campos (invierte píxeles).

\* Actualiza la zona de la ayuda.

NULLNAME ChangeFocus3 ( #c -> )

:: ( #c )

DUP ( #c #c )

ROMPTR 0B0 097 ( #c ) ( cambia el enfoque e invierte píxeles )

TRUESWAP\_ ( T #c )

ROMPTR 0B0 017 ( ) ( muestra ayuda del campo justo encima de menús )

;

Capítulo 39  
La Pantalla

Existen dos pantallas disponibles para el programador en System RPL:

- La pantalla gráfica (GBUFF), la cual es visible, por ejemplo, en las aplicaciones de trazado de gráficos (esta pantalla es conocida como PICT en User RPL).

- La pantalla de texto (ABUFF), la cual es visible en el entorno estándar, donde se muestran los objetos de la pila.

Cuando sea posible, la pantalla ABUFF debe ser usada, dejando a la pantalla gráfica sin cambios, porque suponemos que esta pantalla es un recurso del usuario, la cual no debería ser cambiada por programas.

39.1 Organización de la Pantalla

La HP 50g tiene tres objetos gráficos cuyo fin es servir de pantalla. Los siguientes comandos muestran cada uno de estos grobs:

**Comando Grob**

ABUFF Grob pantalla de texto (mostrar pila).

GBUFF Grob pantalla gráfica (PICT).

HARDBUFF Cualquiera de las dos anteriores (la que se encuentre activa).

HARDBUFF2 Etiquetas de menú (su tamaño es de 131x8)

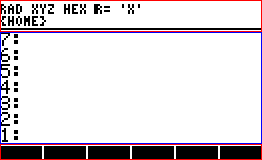
Algo que debemos observer es que los comandos de arriba retornan solo punteros a los grobs, de tal forma que si alteras el grob, la pantalla también será alterada automáticamente. La mayoría de veces este comportamiento es deseado, pero si no quieres eso, llama a TOTEMPOB después de usar cualquiera de los comandos de arriba para hacer una copia en la memoria temporal. Ver la sección 25.1.4 para más información sobre la memoria temporal y las referencias a objetos.

Las pantallas ABUFF y GBUFF pueden ampliarse y se puede observar a través de estas. En cambio, la pantalla que muestra las etiquetas de menú tiene un tamaño fijo de 131x8 píxeles.

El comando TOADISP activa la pantalla de texto (la vuelve visible), y el comando TOGDISP activa a la pantalla gráfica.

La pantalla de texto es dividida en tres regiones. Las áreas de la pantalla son numeradas como uno, dos y tres. En muchos comandos encontrarás la parte “DA”, la cual significa “Display Area”. La siguiente figura muestra cada una de estas tres áreas en la HP 50g.

|  |  |  |
| --- | --- | --- |
| DA 1 | Altura 16 | Filas 0-15 |
| DA 2 | Altura 56  (40 en HP 49g) | Filas 16-71  (16-55 en hp49g) |
| DA 3 | Altura 8 | Filas 72-79  (56-63 en hp49g) |



El area 2 ocupa las filas 16 a 71 (altura 56) y está dividida en dos areas: 2a y 2b.

El área 2a corresponde a la zona donde se muestra la pila.

El área 2b corresponde a la zona donde se muestra la línea de edición.

39.2 Preparando la Pantalla

Dos comandos establecen control sobre la pantalla de texto: RECLAIMDISP y ClrDA1IsStat.

El primero hace lo siguiente:

• Establece a ABUFF como la pantalla activa.

• Limpia toda la pantalla ABUFF.

• Redimensiona ABUFF a su tamaño por defecto (131x80 píxeles en la HP 50g)

Este comando puede parecerse al comando CLLCD de User RPL, aunque este último sólo limpia ABUFF sin cambiar su tamaño.

El comando ClrDA1IsStat es opcional, pero la mayoría de las veces debería ser usado. Este comando suspende la presentación del reloj temporalmente. La mayoría de los programas que usan la pantalla para mostrar algo no desean tener un reloj mostrado en la pantalla.

Cuando no es necesario mostrar las etiquetas de menú, usa el comando TURNMENUOFF para esconder HARDBUFF2 y observar en toda la pantalla a ABUFF o GBUFF.

Para volver visibles nuevamente a las etiquetas de menú usa el comando TURNMENUON.

Para más detalles del menu, lee el capítulo 40.

La plantilla sugerida para una aplicación que usa la pantalla de texto es:

::

ClrDA1IsStat ( suspende reloj temporalmente )

RECLAIMDISP ( fija como actual, limpia y redimensiona a ABUFF )

TURNMENUOFF ( oculta el menú ) ( opcional )

\* aplicación aquí

\* aplicación aquí

\* aplicación aquí

ClrDAsOK ( redibuja la pantalla )

\* O también puedes usar:

SetDAsTemp ( congela toda la pantalla )

;

39.3 Controlando el Refrescar Pantalla

En algunos programas, es deseable que después de que una aplicación termine, la pantalla no sea redibujada, sino que permanezca congelada, de tal manera que el usuario pueda ver algunos resultados o mensajes. En User RPL esto se logra con el comando FREEZE.

En otras ocasiones, es deseable que la pantalla regrese a la normalidad.

En System RPL, varios comandos sirven para estos propósitos. Los más usados son los siguientes:

**Comando Acción**

SetDA1Temp Congela DA1.

SetDA2OKTemp Congela DA2.

SetDA3Temp Congela DA3.

SetDA12Temp Congela DA1 y DA2.

SetDAsTemp Congela toda la pantalla.

ClrDA1OK Redibuja DA1.

ClrDA2OK Redibuja DA2.

ClrDA3OK Redibuja DA3.

ClrDAsOK Redibuja toda la pantalla.

39.4 Limpiando la Pantalla

Los siguientes comandos limpian HARDBUFF, todo el grob o una parte. Recuerda que HARDBUFF se refiere al grob actualmente mostrado (ABUFF o GBUFF). Estos commandos no toman argumentos, excepto por el comando BLANKIT. En la sección de referencia de abajo puedes ver más detalles del uso de estos comandos y también otros relacionados a limpiar la pantalla.

**Comando Acción**

CLEARVDISP Limpia HARDBUFF sin cambiar su tamaño.

BlankDA1 Limpia DA1.

BlankDA2 Limpia DA1.

BlankDA12 Limpia DA1 y DA2.

Clr16 Limpia DA1 (limpia filas 0-15 de la ventana actual).

Clr8 Limpia primera parte de DA1 (limpia filas 0-7 de la ventana actual).

Clr8-15 Limpia segunda parte de DA1 (limpia filas 8-15 de la ventana actual).

CLCD10 Limpia DA1 y DA2.

HARDBUFF debe tener un ancho exacto de 131.

CLEARLCD Limpia HARDBUFF sin cambiar su tamaño y también quita las etiquetas de menu de la pantalla.

HARDBUFF debe tener un ancho exacto de 131.

BLANKIT ( #FilaInic #Filas 🡺 )

Limpia #Filas de HARDBUFF, empezando por #FilaInic.

39.5 Mostrando Texto

Hay dos fuentes en la HP 50g: la “fuente de sistema” y la “minifuente”.

Ambas fuentes pueden ser cambiadas por el usuario, pero solo es posible acceder a dos fuentes al mismo tiempo.

La altura de los caracteres de la fuente de sistema puede variar (6, 7, u 8 píxeles), pero el ancho es constante e igual a 6.

El tamaño de los caracteres de la minifuente es de 4x6 píxeles y es fijo.

Para mostrar texto en la pantalla puedes usar los comandos de la sección 39.6.11.

Si el flag 72 está activado (Stack:mini font), el texto se mostrará en minifuente.

Si el flag 72 está desactivado (Stack:current fnt), el texto se mostrará con fuente de sistema.

En la HP 50g siempre se podrá ver por lo menos a nueve líneas.

Para mostrar texto en la pantalla también puedes usar los comandos del capítulo 15, tales como: $>grob, GROB! o XYGROBDISP. Para más información sobre estos comandos y otros relacionados a grobs, puedes revisar el capítulo 15.

Para mostrar un mensaje dentro de un rectángulo en el centro de la pantalla puedes usar el comando FlashWarning. Este comando hace beep y luego muestra la cadena. El usuario debe presionar CANCL u OK para continuar.

39.6 Referencia

39.6.1 Organización de la Pantalla

**Direcc. Nombre Descripción**

26166 TOADISP ( 🡺 )

Establece la pantalla de texto (ABUFF) como la activa.

Equivale al comando TEXT de User RPL.

2616B TOGDISP ( 🡺 )

Establece la pantalla gráfica (GBUFF) como la activa.

25FA4 ABUFF ( 🡺 ABUFF )

Retorna el grob texto a la pila.

26076 GBUFF ( 🡺 GBUFF )

Retorna el grob gráfico a la pila.

En la HP49 y HP50 la dirección en extable para ExitAction! es la misma, pero esto debe ser un bug.

2608F HARDBUFF ( 🡺 ABUFF/GBUFF )

Retorna el grob actualmente mostrado a la pila.

26094 HARDBUFF2 ( 🡺 menugrob )

Retorna el grob de los menús a la pila.

Es un grob de dimensiones 131x8.

25EDE HARDHEIGHT ( 🡺 #h )

Retorna la altura de HARDBUFF.

Hace HARDBUFF GROBDIM DROP

25ED5 GBUFFGROBDIM ( 🡺 #h #w )

Retorna las dimensiones de GBUFF.

Hace GBUFF luego GROBDIM

39.6.2 Preparando la Pantalla

**Direcc. Nombre Descripción**

25EF4 RECLAIMDISP ( 🡺 )

Establece a ABUFF como la pantalla activa.

Establece a ABUFF como un grob en blanco de 131x80 (131x64 en la HP49g) y mueve la ventana a x=#0, y=#0.

26418 PTR 26418 ( 🡺 )

Si HARDBUFF es mayor que el tamaño de la pantalla, fija HARDBUFF a dicho tamaño (131x80 ó 131x64 en la HP49g).

2EE7D ClrDA1IsStat ( 🡺 )

Suspende temporalmente la visualización del reloj.

2EEAC SetDA1IsStat ( 🡺 )

Hace visible al reloj.

2EEAB DA1IsStatus? ( 🡺 flag )

Retorna FALSE si se ha ocultado el reloj con ClrDA1IsStat.

2EEFD MENUOFF? ( 🡺 flag )

Retorna TRUE si el menú es invisible.

2F034 TURNMENUOFF ( 🡺 )

Quita de la pantalla las etiquetas de menú.

También aumenta filas a ABUFF/GBUFF si es necesario para forzar que su altura sea por lo menos de 80.

También fuerza a que la posición ‘y1’ de la ventana actual sea como máximo de h-80

No usar cuando el grob actual tiene un ancho igual a cero.

**Direcc. Nombre Descripción**

2F031 TURNMENUON ( 🡺 )

Hace visibles las etiquetas de menú en la pantalla.

2F353 LINECHANGE ( # 🡺 )

Si # es BINT63, el menú será invisible.

Si # es BINT55, el menú será visible.

26247 GetHeader ( 🡺 # )

Retorna el número de lineas visibles en el header (0-2).

Usado por el comando HEADER🡪 de User RPL.

26283 SetHeader ( # 🡺 )

Establece el número de líneas visibles del header (0-2).

Usado por el comando 🡪HEADER de User RPL.

26099 HEIGHTENGROB ( GBUFF #filas 🡺 )

( ABUFF #filas 🡺 )

Aumenta #filas a GBUFF o ABUFF en la parte inferior. Para esto, el gráfico debe tener un ancho diferente a cero.

39.6.3 Refrescando Inmediatamente la Pantalla

**Direcc. Nombre Descripción**

2EF67 SysDisplay ( 🡺 )

Redibuja todas las areas de la pantalla.

Lo hace inmediatamente, sin esperar que el comando actual termine.

2C305 DispStatus ( 🡺 )

Establece a ABUFF como la pantalla activa y muestra ahora el área de estado (DA1).

2C311 ?DispStatus ( 🡺 )

Llama a DispStatus si es necesario.

091002 FLASHPTR 002 091 ( 🡺 )

Establece a ABUFF como la pantalla activa y muestra ahora el área de la pila (DA2a).

2C341 ?DispStack ( 🡺 )

Llama a FLASHPTR 002 091 si es necesario.

2F19E DispCommandLine ( 🡺 )

Establece a ABUFF como la pantalla activa y redibuja la línea de comandos ahora.

2F19F ?DispCommandLine ( 🡺 )

Llama a DispCommandLine si es necesario.

2EE5A DispEditLine ( 🡺 )

Sólo llama a DispCommandLine.

2DFF4 DispMenu.1 ( 🡺 )

Dibuja el menú ahora.

2DFE0 DispMenu ( 🡺 )

:: DispMenu.1 SetDA3Valid ;

2DFCC ?DispMenu ( 🡺 )

Redibuja el menú ahora si ninguna tecla está esperando en el buffer.

Aun mejor es esto:

:: DA3OK?NOTIT ?DispMenu ;

**Direcc. Nombre Descripción**

2C2F9 DispStsBound ( 🡺 )

Muestra una línea horizontal negra de ancho 131 en la fila 14.

Muestra una línea horizontal blanca de ancho 131 en la fila 15.

Estas líneas se muestran en HARDBUFF.

Corresponden a la separación entre el header y el área de la pila, cuando el flag 73 está desactivado (Edit: current font).

2A7F7 DispTimeReq? ( 🡺 flag )

¿Es requerido mostrar el tiempo? Verifica el flag 40 y algo más.

39.6.4 Controlando el Refrescar Pantalla

**Direcc. Nombre Descripción**

2EE8D ClrDA1OK ( 🡺 )

Redibuja DA1

2EE8E ClrDA2aOK ( 🡺 )

Redibuja DA2a

2EE8F ClrDA2bOK ( 🡺 )

Redibuja DA2b

2EE90 ClrDA2OK ( 🡺 )

Redibuja DA2

2EE6E ClrDA3OK ( 🡺 )

Redibuja DA3

2EE6D ClrDAsOK ( 🡺 )

Redibuja toda la pantalla.

2EE62 DA1OK? ( 🡺 flag )

2EE82 (DA2aOK?) ( 🡺 flag )

2EE84 (DA2bOK?) ( 🡺 flag )

2EE63 DA3OK? ( 🡺 flag )

2EE66 DA2aLess1OK? ( 🡺 flag )

2BF3A DA1OK?NOTIT ( 🡺 )

Hace DA1OK?, luego NOT\_IT.

2BF53 DA2aOK?NOTIT ( 🡺 )

Hace DA2aOK?, luego NOT\_IT.

2BF6C DA2bOK?NOTIT ( 🡺 )

Hace DA2bOK?, luego NOT\_IT.

2BF85 DA3OK?NOTIT ( 🡺 )

Hace DA3OK?, luego NOT\_IT.

2EE69 SetDA1Temp ( 🡺 )

Congela DA1.

2EE8A SetDA2aTemp ( 🡺 )

Congela DA2a.

2EE6A SetDA2bTemp ( 🡺 )

Congela DA2b.

2EEA7 ClrDA2bTemp ( 🡺 )

2EEA6 (DA2bTemp?) ( 🡺 flag )

2F37A SetDA2OKTemp ( 🡺 )

Congela DA2.

2EE6B SetDA3Temp ( 🡺 )

Congela DA3.

2EE71 SetDA12Temp ( 🡺 )

Congela DA1 y DA2.

**Direcc. Nombre Descripción**

2EE64 SetDAsTemp ( 🡺 )

Congela toda la pantalla.

2EEA3 (SetDA2aTempF) ( 🡺 )

2EEA5 SetDA2bTempF ( 🡺 )

2EEA9 (SetDA3TempF) ( 🡺 )

2EE67 SetDA1Valid ( 🡺 )

2EF98 SetDA2aValid ( 🡺 )

2EE68 SetDA2bValid ( 🡺 )

2EE91 SetDA2Valid ( 🡺 )

2EF99 SetDA3Valid ( 🡺 )

2EE97 (SetDA1ValidF) ( 🡺 )

2EEA0 SetDA3ValidF ( 🡺 )

2EE78 SetDA1Bad ( 🡺 )

2EE74 ClrDA1Bad ( 🡺 )\*\*\*

2EEB0 DA1Bad? ( 🡺 flag )

2EE79 SetDA2aBad ( 🡺 )

2EE75 ClrDA2aBad ( 🡺 )\*\*\*

2EEB1 DA2aBad? ( 🡺 flag )

2EE7A SetDA2bBad ( 🡺 )

2EEB3 ClrDA2bBad ( 🡺 )\*\*\*

2EEB2 DA2bBad? ( 🡺 flag )

2EE7B SetDA3Bad ( 🡺 )

2EEB5 ClrDA3Bad ( 🡺 )\*\*\*

2EEB4 DA3Bad? ( 🡺 flag )

2EE72 SetDA1NoCh ( 🡺 )

2EEBA (DA1NoCh?) ( 🡺 flag )

2EE93 SetDA2NoCh ( 🡺 )

2EE73 SetDA2aNoCh ( 🡺 )

2EEB9 (DA2aNoCh?) ( 🡺 flag )

2EE76 SetDA2bNoCh ( 🡺 )

2EE81 ClrDA2bNoCh ( 🡺 )

2EEB7 DA2bNoCh? ( 🡺 flag )

2EE77 SetDA3NoCh ( 🡺 )

2EEB6 (ClrDA3NoCh) ( 🡺 )

2EE6F SetDA12NoCh ( 🡺 )

2EE70 SetDA13NoCh ( 🡺 )

2EE94 SetDA23NoCh ( 🡺 )

2EE65 SetDA12a3NCh ( 🡺 )

2F379 SetDA123NoCh ( 🡺 )

2EE7C SetDAsNoCh ( 🡺 )

2EE6C SetDA2aEcho ( 🡺 )

2EEAE SetNoRollDA2 ( 🡺 )

2EEAF ClrNoRollDA2 ( 🡺 )

2EEAD (NoRollDA2?) ( 🡺 flag )

2EE7F SetDA2bIsEdL ( 🡺 )

2EE80 ClrDA2bIsEdL ( 🡺 )\*\*\*

2EE7E DA2bIsEdL? ( 🡺 flag )

**Direcc. Nombre Descripción**

25EA8 Ck&Freeze ( % 🡺 )

Si % es 1, congela DA1.

Si % es 2, congela DA2.

Si % es 3, congela DA1 y DA2.

Si % es 4, congela DA3.

Si % es 5, congela DA1 y DA3.

Si % es 6, congela DA2 y DA3.

Si % es 7 o mayor, 0 o menor, congela DA1, DA2 y DA3.

Llama a

Equivale al comando FREEZE de User RPL.

39.6.5 Limpiando la pantalla

**Direcc. Nombre Descripción**

25E7E BLANKIT ( #FilaInic #Filas 🡺 )

Limpia #Filas de HARDBUFF empezando por #FilaInic.

Las columnas limpiadas serán 0-130.

2EF5E BlankDA1 ( 🡺 )

Limpia el área de estado de HARDBUFF.

Es decir, limpia las filas 0-15 usando BLANKIT.

2F31C BlankDA2a ( 🡺 )

Limpia una parte del área DA2a de HARDBUFF.

Es decir, limpia las filas 16-55 usando BLANKIT.

Si existe línea de edición, limpia una parte menor de la pantalla.

2F31B BlankDA2 ( 🡺 )

Limpia las áreas DA2a y DA2b de HARDBUFF.

Es decir, limpia las filas 16-71 usando BLANKIT.

2EE5C BlankDA12 ( 🡺 )

Limpia las áreas DA1 y DA2 de HARDBUFF.

Es decir, limpia las filas 0-71 usando BLANKIT.

2EED4 Clr8 ( 🡺 )

Limpia las primeras ocho líneas del área de estado de HARDBUFF, pero respecto a la ventana actualmente mostrada.

Es decir, limpia las filas 0-7 de la ventana actual usando

:: TOP8 GROB!ZERODRP ;

2EED5 Clr8-15 ( 🡺 )

Limpia las segundas ocho líneas del área de estado de HARDBUFF, pero respecto a la ventana actualmente mostrada.

Es decir, limpia las filas 8-15 de la ventana actual usando

:: Rows8-15 GROB!ZERODRP ;

2F15E Clr16 ( 🡺 )

Limpia el área de estado de HARDBUFF, pero respecto a la ventana actualmente mostrada.

Es decir, limpia las filas 0-15 de la ventana actual usando

:: TOP16 GROB!ZERODRP ;

261C0 CLCD10 ( 🡺 )

Limpia el area de estado y el área stack de HARDBUFF, pero a lo ancho de todo el grob.

Es decir, limpia las filas 0-71 de la ventana actual, a través de todas las columnas del grob.

HARDBUFF debe tener un ancho exacto de 131.

**Direcc. Nombre Descripción**

2EF05 DOCLLCD ( 🡺 )

Limpia ABUFF sin cambiar su tamaño.

Equivale al comando CLLCD de User RPL.

2EEF9 DOERASE ( 🡺 )

Limpia GBUFF sin cambiar su tamaño.

Equivale al comando ERASE de User RPL.

26021 CLEARVDISP ( 🡺 )

Limpia HARDBUFF sin cambiar su tamaño.

261C5 CLEARLCD ( 🡺 )

Limpia HARDBUFF (sólo las primeras 72 filas) sin cambiar su tamaño y también quita las etiquetas de menu de la pantalla.

HARDBUFF debe tener un ancho exacto de 131 y una altura mínima de 72.

39.6.6 Anuncios y Modos de Control

**Direcc. Nombre Descripción**

2613E SetLeftAnn ( 🡺 )

Muestra el anuncio de shift izquierdo.

2603A ClrLeftAnn ( 🡺 )

Borra el anuncio de shift izquierdo.

26148 SetRightAnn ( 🡺 )

Muestra el anuncio de shift derecho.

2603F ClrRightAnn ( 🡺 )

Borra el anuncio de shift derecho.

26139 SetAlphaAnn ( 🡺 )

Muestra el anuncio de ALPHA.

26035 ClrAlphaAnn ( 🡺 )

Borra el anuncio de ALPHA.

25EE9 LockAlpha ( 🡺 )

Asegura el teclado alfabético.

25F08 UnLockAlpha ( 🡺 )

Remueve el teclado alfabético.

25E6C 1A/LockA ( 🡺 )

Si el flag 60 está activado ([α] locks Alpha), asegura el teclado alfabético. Si el flag 60 está desactivado ([α][α] locks), pone el teclado alfabético sólo para la próxima tecla a presionar.

25738 (TOGLOWERCASE) ( 🡺 )

Alterna la calculadora entre mayúsculas y minúsculas.

2572B (SETLOWERCASE) ( 🡺 )

Pone la calculadora en modo minúsculas.

25730 (CLRLOWERCASE) ( 🡺 )

Pone la calculadora en modo mayúsculas.

25726 (LOWERCASE?) ( 🡺 )

Retorna TRUE si la calculadora está en modo minúsculas.

2649F (ClrBusyAnn) ( 🡺 )

Borra el anuncio de maquina ocupada.

26143 SetPrgmEntry ( 🡺 )

Activa el modo de entrada programación.

264F4 (ClrPrgmEntry) ( 🡺 )

Desactiva el modo de entrada programación.

**Direcc. Nombre Descripción**

2610C PrgmEntry? ( 🡺 flag )

Retorna TRUE si esta activado el modo de entrada programación.

25EBE Do1st/2nd+: ( 🡺 :: <ob1> ; (modo PRG) )

( 🡺 :: <ob2> <rest> ; (no modo PRG) )

Si la calculadora está en modo programación, sólo ejecuta el siguiente objeto del runstream y pasa por alto el resto.

Si la calculadora no está en modo programación, pasa por alto el siguiente objeto del runstream y ejecuta el resto.

25719 SetAlgEntry ( 🡺 )

Activa el modo de entrada algebraica.

2571E ClrAlgEntry ( 🡺 )

Desactiva el modo de entrada algebraica.

256EA AlgEntry? ( 🡺 flag )

Retorna TRUE si esta activado el modo de entrada algebraica.

25EDF ImmedEntry? ( 🡺 flag )

Retorna TRUE si la calculadora está en modo inmediato (modos de entrada algebraica y programación desactivados).

25EFC SetKeysNS ( 🡺 )

Quita teclado alfabético, shifts izq. y der. y actualiza ejecutor.

25E74 ?ClrAlg ( 🡺 )

Si está activado el modo de entrada algebraica, entonces lo desactiva.

25E75 ?ClrAlgSetPr ( 🡺 )

Activa el modo de entrada programación y desactiva el modo de entrada algebraica.

39.6.7 Coordenadas de la Ventana Visible

**Direcc. Nombre Descripción**

26198 WINDOWXY ( #y1 #x1 🡺 )

Fija las coordenadas de la esquina superior izquierda de la ventana.

2617F WINDOWCORNER ( 🡺 #y1 #x1 )

Retorna las coordenadas de la esquina superior izquierda de la ventana.

2F352 LEFTCOL ( 🡺 #x1 )

Retorna la coordenada ‘x’ de la columna del extremo izquierdo de la ventana.

2F36B RIGHTCOL ( 🡺 #x1+130 )

Retorna la coordenada ‘x’ de la columna del extremo derecho de la ventana.

2F385 TOPROW ( 🡺 #y1 )

Retorna la coordenada ‘y’ de la fila del extremo superior de la ventana.

2F31D BOTROW ( 🡺 #y1+71 )

( 🡺 #y1+79 )

Retorna la coordenada ‘y’ de la fila del extremo inferior de la ventana.

Si el menú es visible, retorna #y1+71

Si el menú es invisible, retorna #y1+79

**Direcc. Nombre Descripción**

2EED6 HBUFF\_X\_Y ( 🡺 HBgrob #x1 #y1 )

Retorna HARDBUFF y las coordenadas de la esquina superior izquierda de la ventana.

2F384 TOP8 ( 🡺 HBgrob #x1 #y1 #x1+131 #y1+8 )

Retorna las coordenadas de la primera línea de estado.

#x1, #y1 son coordenadas de la esquina superior izquierda de la ventana.

2F36C Rows8-15 ( 🡺 HBgrob #x1 #y1+8 #x1+131 #y1+16 )

Retorna las coordenadas de la segunda línea de estado.

#x1, #y1 son coordenadas de la esquina superior izquierda de la ventana.

2F383 TOP16 ( 🡺 HBgrob #x1 #y1 #x1+131 #y1+16 )

Retorna las coordenadas del área de estado.

#x1, #y1 son coordenadas de la esquina superior izquierda de la ventana.

2F038 (Save16) ( 🡺 grob131x16 )

Retorna area de estado en la ventana actual.

Hace :: TOP16 SUBGROB ;

2F3B6 (Restore16) ( grob 🡺 )

Pone grob en HARDBUFF en la esquina superior izquierda de la ventana.

Hace :: HBUFF\_X\_Y GROB! ;

39.6.8 Moviendo la Ventana

**Direcc. Nombre Descripción**

26193 WINDOWUP ( 🡺 )

Mueve la pantalla un pixel hacia arriba.

26184 WINDOWDOWN ( 🡺 )

Mueve la pantalla un pixel hacia abajo.

26189 WINDOWLEFT ( 🡺 )

Mueve la pantalla un pixel hacia la izquierda.

2618E WINDOWRIGHT ( 🡺 )

Mueve la pantalla un pixel hacia la derecha.

2F370 SCROLLUP ( 🡺 )

Mueve la pantalla un pixel hacia arriba. Y la sigue moviendo si la tecla ARRIBA se mantiene presionada.

2F36D SCROLLDOWN ( 🡺 )

Mueve la pantalla un pixel hacia abajo. Y la sigue moviendo si la tecla ABAJO se mantiene presionada.

2F36E SCROLLLEFT ( 🡺 )

Mueve la pantalla un pixel hacia la izquierda. Y la sigue moviendo si la tecla IZQUIERDA se mantiene presionada.

2F36F SCROLLRIGHT ( 🡺 )

Mueve la pantalla un pixel hacia la derecha. Y la sigue moviendo si la tecla DERECHA se mantiene presionada.

2F34A JUMPTOP ( 🡺 )

Salta a la parte alta de la pantalla, fijando #y como cero.

2F347 JUMPBOT ( 🡺 )

Salta a la parte más baja de la pantalla.

2F348 JUMPLEFT ( 🡺 )

Salta al extremo izquierdo de la pantalla, fijando #x como cero.

**Direcc. Nombre Descripción**

2F349 JUMPRIGHT ( 🡺 )

Salta al extremo derecho de la pantalla.

39.6.9 Mostrando Objetos

**Direcc. Nombre Descripción**

2F21D ViewObject ( ob 🡺 )

Muestra ob en la pantalla.

Es posible la alternancia TEXT/GRAPH.

Equivale al comando SCROLL de User RPL.

2F21E ViewStrObject ( flag $ 🡺 F )

( flag ob 🡺 F )

Si en la pila hay una cadena que no es demasiado larga, se insertan los saltos de línea necesarios para que cada renglón tenga como máximo 33 caracteres.

Si en la pila hay un objeto que no es cadena, este es convertido a cadena con ^FSTR11

Finalmente ejecuta un POL, en el que podremos desplazarnos a través de la cadena con las teclas de dirección.

Si flag es TRUE, es posible la alternancia TEXT/GRAPH.

2F21F ViewGrobObject ( flag grob 🡺 F )

( flag ob 🡺 F )

Si en la pila hay un objeto que no es grob, este es convertido a grob con: %0 FLASHPTR 001 10B

Finalmente ejecuta un POL, en el que podremos desplazarnos a través del grob con las teclas de dirección.

Si flag es TRUE, es posible la alternancia TEXT/GRAPH.

0C1007 ˆSCROLLext ( ob 🡺 )

Si el flag 100 está activado (Step by step on), muestra ob con el comando ViewObject.

Si el flag 100 está desactivado (Step by step off), sólo quita el objeto de la pila y no hace nada más.

2EF61 WINDOW# ( #x #y 🡺 )

Muestra GBUFF empezando en las coordenadas dadas.

Si las coordenadas no son correctas, genera el error: “Argumento: valor incorr”.

Previamente, cambia las dimensiones de GBUFF si es pequeño. Fuerza a que tenga un ancho mínimo de 131 y una altura mínima de 80.

39.6.10 Fuentes y Altura de Pantalla

**Direcc. Nombre Descripción**

078002 FLASHPTR 002 078 ( 🡺 font )

Retorna la fuente por defecto de altura 8 píxeles.

Equivale al comando FONT8 de User RPL.

079002 FLASHPTR 002 079 ( 🡺 font )

Retorna la fuente por defecto de altura 7 píxeles.

Equivale al comando FONT7 de User RPL.

07A002 FLASHPTR 002 07A ( 🡺 font )

Retorna la fuente por defecto de altura 6 píxeles.

Equivale al comando FONT6 de User RPL.

**Direcc. Nombre Descripción**

2621A FONT> ( 🡺 font )

Retorna la actual fuente del sistema.

Usado por el comando FONT🡪 de User RPL.

2F1A9 EDITF ( font 🡺 ... font' T )

( font 🡺 ... F )

Permite editar a una fuente.

2F1A7 CHARSEDIT ( 🡺 )

Abre la aplicación que muestra la colección de todos los caracteres de la calculadora de la fuente actual.

Si no existe línea de edición, se podrán editar los caracteres.

Si existe línea de edición, se podrán colocar los caracteres en la línea de comandos.

2F1A8 EditFont ( font flag prog 🡺 ... font' T )

( font flag prog 🡺 ... F )

Permite ver o editar a una fuente.

Si flag es TRUE, será posible la edición.

prog es un comando o programa de la forma:

chr #1/#2 🡪 ... T/F

#1 si se presionó ECHO1

#2 si se presionó ECHO

TRUE finalizará el POL.

25F15 >FONT ( font 🡺 )

Fija la nueva fuente del sistema.

Equivale al comando 🡪FONT de User RPL.

2625B MINIFONT> ( 🡺 minifont )

Retorna la actual minifuente.

Usado por el comando MINIFONT🡪 de User RPL.

2620B >MINIFONT ( minifont 🡺 )

Fija la nueva minifuente.

Equivale al comando 🡪MINIFONT de User RPL.

578001 FLASHPTR 001 578 ( 🡺 80/64 )

Retorna altura de la pantalla en píxeles.

En HP 50g y HP 49g+ retorna siempre 80.

26288 StackLineHeight ( 🡺 # )

Retorna la altura en píxeles del área donde se muestra la pila.

En la HP 50g:

- Retorna 58 cuando el header tiene 2 líneas (56 con fuente 8).

- Retorna 64 cuando el header tiene 1 línea (65 con fuente 7).

- Retorna 72 cuando el header no es visible.

2623D GetFontHeight ( 🡺 # )

Retorna la altura de la fuente del sistema (6, 7 u 8).

26242 GetFontStkHeight ( 🡺 # )

Retorna la altura de la fuente usada al mostrar la pila.

Si el flag 72 está activado (Stack:mini font), retorna la altura de la minifuente (6).

Si el flag 72 está desactivado (Stack:current fnt), retorna la altura de la fuente del sistema (6, 7 u 8).

aka: StackFontHeight

06F004 ^FontBrowser ( 🡺 font T )

( 🡺 F )

Abre el Filer para explorar en busca de fuentes de sistema.

39.6.11 Mostrando Texto en la Pantalla

**Direcc. Nombre Descripción**

25F16 DISP\_LINE ( $ #línea flag 🡺 )

Muestra la cadena a partir de la línea (1-11) especificada.

Si flag es TRUE, se pueden ver varias líneas de la cadena (cuando hay saltos de línea), de lo contrario se verá una sola línea.

Si flag 72 está activado (Stack:mini font), estarán disponibles 11 líneas.

Si flag 72 está desactivado (Stack:current fnt), el número de líneas disponibles será 11 (FONT6), 10 (FONT7) ó 9 (FONT8).

Si alguna línea es muy larga, al final de esta se verán puntos.

25EB4 DODISP ( **ob** %línea 🡺 )

Muestra un objeto en la pantalla a partir de la línea indicada.

Si el objeto no es una cadena, este será convertido a cadena con ^FSTR6, mostrándose sólo nueve líneas.

Si hay saltos de línea, se pueden ver varias líneas.

Si flag 72 está activado (Stack:mini font), estarán disponibles 11 líneas.

Si flag 72 está desactivado (Stack:current fnt), el número de líneas disponibles será 11 (FONT6), 10 (FONT7) ó 9 (FONT8).

Si alguna línea es muy larga, al final de esta se verán puntos.

Si %línea es 0 o negativo, se mostrará a partir de la línea 1.

Equivale al comando DISP de User RPL.

25FB3 DISPN ( $ #línea 🡺 )

Muestra la cadena en la línea especificada.

Se verá en una sola línea.

Si flag 72 está activado (Stack:mini font), estarán disponibles 13 líneas.

Si flag 72 está desactivado (Stack:current fnt), el número de líneas disponibles será 13 (FONT6), 11 (FONT7) o 10 (FONT8).

Si la cadena es muy larga, al final de esta no se verán puntos.

En este comando, #línea debe tener un valor correcto. De lo contrario, se produce un crash.

aka: BIGDISPN

25EBC Disp5x7 ( $ #líneaInicial #MaxLíneas 🡺 )

Muestra la cadena en líneas multiples (si hay saltos de línea), empezando en la línea #líneaInicial y no usando más de #MaxLíneas líneas.

Si flag 72 está activado (Stack:mini font), estarán disponibles 13 líneas.

Si flag 72 está desactivado (Stack:current fnt), el número de líneas disponibles será 13 (FONT6), 11 (FONT7) o 10 (FONT8).

Si alguna línea es muy larga, al final de esta se verán puntos.

Si #líneaInicial es muy grande , no hay problemas.

**Direcc. Nombre Descripción**

26260 nDISPSTACK ( $ #AlturaPixeles #PixelInicial flag flag’ 🡺 )

Muestra la cadena en la pantalla a partir de la fila #PixelInicial y con altura #AlturaPixeles.

Si flag es TRUE, se pueden ver varias líneas de la cadena (cuando hay saltos de línea), de lo contrario se verá una sola línea.

Si flag’ es TRUE, las filas sobrantes se limpiarán cuando el texto no cubra toda la altura especificada.

25FB8 DISPROW1 ( $ 🡺 )

Muestra la cadena sólo en la primera línea y con fuente de sistema o minifuente de acuerdo al estado del flag 72.

Si la cadena es muy larga, al final de esta no se verán puntos.

aka: DISP@01, BIGDISPROW1

25FBD DISPROW2 ( $ 🡺 )

aka: DISP@09, BIGDISPROW2

25FC2 DISPROW3 ( $ 🡺 )

aka: DISP@17, BIGDISPROW3

25FC7 DISPROW4 ( $ 🡺 )

aka: DISP@25, BIGDISPROW4

25FCC DISPROW5 ( $ 🡺 )

261F7 DISPROW6 ( $ 🡺 )

25FD1 DISPROW7 ( $ 🡺 )

25FD6 DISPROW8 ( $ 🡺 )

25FDB DISPROW9 ( $ 🡺 )

25FE0 DISPROW10 ( $ 🡺 )

Si el flag 72 está desactivado (Stack:current fnt), la altura de la fuente de sistema actual es 8 y el menú es visible, entonces no se verá esta línea en la HP 50g.

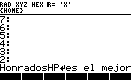
25FE5 DISPLASTROW ( $ 🡺 )

Muestra la cadena casi al final.

Muestra la cadena con fuente de sistema o minifuente de acuerdo al estado del flag 72.

Si el flag 72 está activado (Stack:mini font), la muestra en la línea 11.

Si el flag 72 está desactivado (Stack:current fnt), la muestra en la línea 11, 10 ó 9, según la altura de la fuente (6, 7 u 8 respectivamente).



**Direcc. Nombre Descripción**

25FEA DISPLASTROWBUT1 ( $ 🡺 )

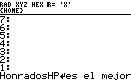
Muestra la cadena casi al final (una línea más abajo que al usar DISPLASTROW).

Muestra la cadena con fuente de sistema o minifuente de acuerdo al estado del flag 72.

Si el flag 72 está activado (Stack:mini font), la muestra en la línea 12.

Si el flag 72 está desactivado (Stack:current fnt), la muestra en la línea 12, 11 ó 10, según la altura de la fuente (6, 7 u 8 respectivamente).

Cuando la altura de la fuente sea 7 u 8, la cadena se verá, sólo si el menú es invisible.

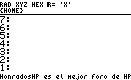


2EEFF DispCoord1 ( $ 🡺 )

Muestra la cadena en HARDBUFF2 usando la minifuente.

Observa que es puesta en HARDBUFF2. Por lo tanto, esta cadena no se verá cuando el menú se ha hecho invisible (por ejemplo, al usar TURNMENUOFF).

Puedes usar este comando cuando la pantalla activa sea ABUFF o GBUFF.



2F32B DISPCOORD2 ( $ 🡺 )

Llama a DispCoord1 para mostrar la cadena.

Luego espera a que el usuario presione una tecla.

Cuando el usuario presiona cualquier tecla, se vuelve a mostrar el menú.

Finalmente, llama a InitEdModes.

0C8002 ^DISPROW1\_plus\_ ( $ 🡺 )

Primero, limpia las filas 0-7 de la ventana actual con Clr8.

Luego, muestra la cadena en una línea con su esquina superior izquierda en la posición (#x1,#y1) (esquina superior izquierda de la ventana actual).

Muestra la cadena con fuente de sistema o minifuente de acuerdo al estado del flag 72.

Puede mostrar en ABUFF y también en GBUFF (con ancho mayor o igual a 131).

25EAB DISPROW1\* ( $ 🡺 )

Hace lo mismo que ^DISPROW1\_plus\_, pues sólo llama a este.

**Direcc. Nombre Descripción**

0C9002 ^DISPROW2\_plus\_ ( $ 🡺 )

Primero, limpia las filas 0-7 de la ventana actual con

Clr8-15.

Luego, muestra la cadena en una línea con su esquina superior izquierda en la posición (#x1,#y1+8).

Muestra la cadena con fuente de sistema o minifuente de acuerdo al estado del flag 72.

Puede mostrar en ABUFF y también en GBUFF (con ancho mayor o igual a 131).

25EAC DISPROW2\* ( $ 🡺 )

Hace lo mismo que ^DISPROW2\_plus\_, pues sólo llama a este.

25EAD DISPSTATUS2 ( $ 🡺 )

Muestra el mensaje en el área de estado (en la ventana actual) usando las 2 líneas.

Para esto separa la cadena en dos partes.

La primera línea es mostrada con ^DISPROW1\_plus\_

El resto de la cadena es mostrada con ^DISPROW2\_plus\_

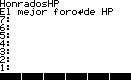
Puede mostrar en ABUFF y también en GBUFF.

38C00 (DoPrompt) ( $ 🡺 )

Hace DISPSTATUS2 y luego congela el area de estado (sólo congela cuando la pantalla actual es ABUFF).

Por ejemplo, si la pantalla actual es ABUFF y la cadena es:

"HonradosHP\0AEl mejor foro\0Ade HP", se mostrará:



25F12 sstDISP ( $ 🡺 )

( romptr 🡺 )

Muestra el objeto en las dos primeras filas con DISPSTATUS2 y luego congela la línea de estado.

25ED4 FlashMsg ( $ 🡺 )

Muestra el mensaje en el área de estado (de la ventana actual) usando DISPSTATUS2, durante casi 1 segundo. Luego devuelve el área de estado a su valor anterior.

Ni siquiera congelando la pantalla después, es posible seguir viendo el mensaje.

39.6.12 Mostrando Texto en el centro de la pantalla: DoMsgBox

**Direcc. Nombre Descripción**

2EE61 FlashWarning ( $ 🡺 )

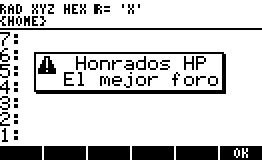
( # 🡺 )

Muestra una cadena en una caja de mensaje y hace beep.

Si en la pila hay un bint, este se convierte a cadena usando su mensaje de error correspondiente.

El usuario puede continuar, solo si presiona la tecla de menu OK, ENTER u ON.

La cadena se muestra con fuente de sistema o minifuente de acuerdo al estado del flag 72.



02E002 ˆDoAlert ( $ 🡺 )

( # 🡺 )

Similar al comando FlashWarning, pero congela la pantalla.

Con el comando FlashWarning, la pantalla era redibujada luego de que el usuario decidía continuar.

2EE60 DoWarning ( $ 🡺 )

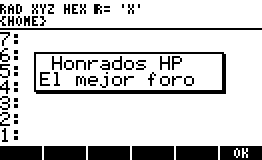
Establece a ABUFF como la pantalla activa y luego llama al comando ˆDoAlert.

007002 ˆCk&DoMsgBox ( $ 🡺 )

( # 🡺 )

Parecido al comando FlashWarning, pero no muestra el grob de alerta y no hace beep.

Al igual que el comando FlashWarning, la pantalla es redibujada luego de que el usuario decide continuar.



0000B1 ˜DoMsgBox ( $/# ob ob' grob menu 🡺 T )

( $/# ob ob' MINUSONE menu 🡺 T )

Con este comando puedes mostrar un mensaje como al usar FlashWarning, pero de manera personalizada. Puedes colocar el menú y el grob que desees. Si no deseas ningún grob, pon MINUSONE, (como al usar ˆCk&DoMsgBox).

ob y ob’ son dos objetos cualesquiera.

El menu es un objeto tal como se describe en el capítulo 40.

Este comando es llamado por FlashWarning, ˆDoAlert y ˆCk&DoMsgBox.

**Direcc. Nombre Descripción**

0040B1 ˜MsgBoxMenu ( 🡺 menu )

Retorna el menu por defecto que se puede usar al llamar al comando ˆCk&DoMsgBox para mostrar mensajes personalizados. Por ejemplo, el siguiente código hace lo mismo que el comando FlashWarning sobre la cadena:

:: "HonradosHP\0AEl mejor foro" ( $ )

ZEROZERO ( $ #0 #0 )

ROMPTR grobAlertIcon ( $ #0 #0 grob )

' ROMPTR MsgBoxMenu ( $ #0 #0 grob menu )

ERRBEEP ( $ #0 #0 grob menu ) ( beep )

ROMPTR DoMsgBox ( T )

DROP ( )

;

39.7 Ejemplos

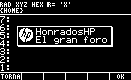
**Ejemplo 1 DoMsgBox**

El siguiente programa muestra un mensaje personalizado con el comando

ROMPTR DoMsgBox

Puedes poner el grob que desees y también cualquier menú.

En las figuras de la derecha se muestra el resultado.

NULLNAME MensajePersonalizado ( -> )

::

"HonradosHP\0AEl gran foro"

ZEROZERO

GROB 00042 E0000E00000B10CD70EDF1ECF1F813FA537B637DA3B5A3E7C3E7F1CBF08B700B00

' :: NoExitAction

{

{ "TORNA" :: TakeOver ABUFF INVGROB DROP ; }

NullMenuKey

NullMenuKey

NullMenuKey

NullMenuKey

{ "OK" :: TakeOver FLASHPTR DoMKeyOK ; }

}

;

ROMPTR DoMsgBox

DROP

;

**Ejemplo 2 ScrollGrob**

Con el siguiente programa podrás ver un grob usando toda la pantalla y sin el menú.

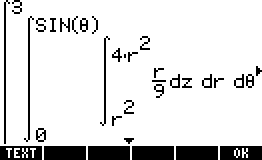
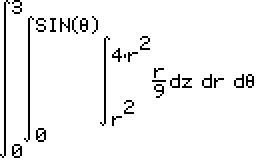
Con las teclas de dirección podrás desplazarte por el grob, incluso manteniendo presionada la tecla.

Con shift izquierdo más una tecla de dirección podrás desplazarte por el grob, una pantalla hacia arriba, hacia abajo, hacia la derecha o hacia la izquierda.

Con shift derecho más una tecla de dirección podrás desplazarte hacia un extremo del grob.

Con la tecla ENTER o la tecla ON puedes salir del programa.

Se usó el comando FLASHPTR 001 578, el cual retorna 80 en la hp 50g y hp49g+ y retorna 64 en la hp 49g. En versiones antiguas de ROM (1.24 y anteriores) de la hp 49g este comando no está, por lo que debes cambiarlo por otro comando o programa si deseas usarlo ahí.

Comando SCROLL de User RPL Comando FULLSCROLL de este ejemplo

xNAME FULLSCROLL ( grob -> )

::

CK1 ( ob ) ( se requiere un argumento )

DUPTYPEGROB? ( ob flag )

NcaseTYPEERR ( genera error: "Argumento incorrecto" )

( grob )

GBUFF ( grob GBUFF ) ( llama a GBUFF )

TOTEMPOB ( grob GROB ) ( hace una copia de GBUFF )

1LAMBIND ( grob ) ( guarda la copia de GBUFF en variable local )

GROB>GDISP ( ) ( el grob pasará a ser el nuevo GBUFF )

CHECKPICT ( ) ( fuerza a que GBUFF tenga ancho>=131 y altura >=80 )

TOGDISP ( ) ( vuelve a GBUFF como la pantalla activa )

TURNMENUOFF ( ) ( hace invisible al menú )

BEGIN

:: FALSE ( F )

WaitForKey ( F #ct #p )

BINT1 #=casedrop

:: ( F #ct )

BINT10 #=casedrop

SCROLLUP ( sale con F )

BINT15 #=casedrop

SCROLLDOWN ( sale con F )

BINT14 #=casedrop

SCROLLLEFT ( sale con F )

BINT16 #=casedrop

SCROLLRIGHT ( sale con F )

BINT47 #=casedrop

DROPTRUE ( sale con T )

BINT51 #=casedrop

DROPTRUE ( sale con T )

DropBadKey ( F )

;

BINT2 #=casedrop

:: ( F #ct )

BINT10 #=casedrop

:: ( F )

REPEATER

10

:: ( F F )

WINDOWCORNER ( F F #y #x )

SWAP ( F F #x #y )

FLASHPTR 001 578 ( F F #x #y 80 )

DIFF\_OR\_ZERO\_ ( F F #x #u[y-80] )

SWAP ( F F #u[y-80] #x )

WINDOWXY ( F F )

;

( F )

;

BINT15 #=casedrop

:: ( F )

REPEATER

15

:: ( F F )

WINDOWCORNER ( F F #y #x )

SWAP ( F F #x #y )

FLASHPTR 001 578 ( F F #x #y 80 )

#+ ( F F #x #y+80 )

HARDHEIGHT ( F F #x #y+80 #h )

FLASHPTR 001 578 ( F F #x #y+80 #h 80 )

#- ( F F #x #y+80 #h-80 )

#MIN ( F F #x #min[y+80,h-80] )

SWAP ( F F #min[y+80,h-80] #x )

WINDOWXY ( F F )

;

( F )

;

BINT14 #=casedrop

:: ( F )

REPEATER

14

:: ( F F )

WINDOWCORNER ( F F #y #x )

BINT131 ( F F #y #x 131 )

DIFF\_OR\_ZERO\_ ( F F #y #u[x-131] )

WINDOWXY ( F F )

;

( F )

;

BINT16 #=casedrop

:: ( F )

REPEATER

16

:: ( F F )

WINDOWCORNER ( F F #y #x )

BINT131 ( F F #y #x 131 )

#+ ( F F #y #x+131 )

HARDBUFF ( F F #y #x+131 GROB )

GROBDIMw ( F F #y #x+131 #w )

BINT131 ( F F #y #x+131 #w 131 )

#- ( F F #y #x+131 #w-131 )

#MIN ( F F #y #min[#x+131,w-131] )

WINDOWXY ( F F )

;

( F )

;

DropBadKey ( F )

;

BINT3 #=casedrop

:: ( F #ct )

BINT10 #=casedrop

JUMPTOP ( sale con F )

BINT15 #=casedrop

JUMPBOT ( sale con F )

BINT14 #=casedrop

JUMPLEFT ( sale con F )

BINT16 #=casedrop

JUMPRIGHT ( sale con F )

BINT47 #=casedrop

TurnOff ( sale con F )

DropBadKey ( F )

;

2DropBadKey ( F )

; ( T/F )

UNTIL

( )

TOADISP ( ) ( vuelve a ABUFF como la pantalla activa )

1GETABND ( grob ) ( retorna el contenido de 1LAM y abandona entorno temporal )

GROB>GDISP ( ) ( restaura el valor guardado de GBUFF )

;

**Ejemplo 3 ScrollGrobDelta**

Este programa es similar al anterior. De esta manera, podrás ver un grob usando toda la pantalla y sin el menú.

Las diferencias son:

- Al presionar una tecla de dirección puedes desplazarte de 7 píxeles en 7 píxeles (más rápido que en el programa anterior donde el desplazamiento era de 1 píxel en 1 píxel).

- Sólo funciona en HP 50g y HP 49g+ (pantalla de 131x80).

- El gráfico es mostrado en la pantalla de texto (el otro se mostraba en la pantalla gráfica).

DEFINE LAM\_GROB 7GETLAM

DEFINE LAM\_#h 6GETLAM

DEFINE LAM\_#w 5GETLAM

DEFINE LAM\_#x 4GETLAM

DEFINE LAM\_#y 3GETLAM

DEFINE LAM\_#d 1GETLAM

DEFINE STO\_#x 4PUTLAM

DEFINE STO\_#y 3PUTLAM

\* Muestra un grob usando toda la pantalla (sin menú).

\* Sólo para HP 50g y HP 49g+

xNAME FULLSCROLLD ( GROB -> )

::

CK1 ( ob ) ( se requiere un argumento )

DUPTYPEGROB? ( ob flag )

NcaseTYPEERR ( genera error: "Argumento incorrecto" )

( GROB )

DUPGROBDIM ( GROB #h #w )

ZEROZERO ( GROB #h #w #0 #0 )

' ::

LAM\_GROB ( GROB )

LAM\_#x ( GROB #x1 )

LAM\_#y ( GROB #x1 #y1 )

OVER BINT131 #+ ( GROB #x1 #y1 #x2 )

OVER BINT80 #+ ( GROB #x1 #y1 #x2 #y2 )

SUBGROB ( GROB' )

ABUFF ( GROB' ABUFF )

ZEROZERO ( GROB' ABUFF #0 #0 )

GROB! ( )

;

( GROB #h #w #0 #0 prog )

BINT7 ( GROB #h #w #0 #0 prog #d )

\* #d es el paso (en píxeles) al desplazarte. Aquí es 7. Puedes cambiarlo.

' NULLLAM BINT7 NDUPN DOBIND ( Crea entorno temporal con 7 LAMS )

\* 7LAM=GROB 6LAM=#h 5LAM=#w 4LAM=#x 3LAM=#y 2LAM=programa 1LAM=#d

TOADISP ( ) ( vuelve a ABUFF como la pantalla activa )

ClrDA1IsStat ( ) ( suspende la presentación del reloj temporalmente )

CLEARVDISP ( ) ( Limpia HARDBUFF sin cambiar su tamaño )

BINT63 LINECHANGE ( ) ( Quita de la pantalla las etiquetas de menú )

2GETEVAL ( ) ( Evalúa 2LAM ) ( Redibuja la pantalla )

BEGIN

:: WaitForKey

BINT1 #=casedrop

:: BINT15 #=casedrop ( ABAJO )

:: REPEATER

BINT15

:: LAM\_#y LAM\_#d #+

LAM\_#h BINT80 DIFF\_OR\_ZERO\_

#MIN ( # )

STO\_#y ( )

2GETEVAL ( )

;

FALSE

;

BINT10 #=casedrop ( ARRIBA )

:: REPEATER

BINT10

:: LAM\_#y LAM\_#d DIFF\_OR\_ZERO\_ ( # )

STO\_#y ( )

2GETEVAL ( )

;

FALSE

;

BINT16 #=casedrop ( DERECHA )

:: REPEATER

BINT16

:: LAM\_#x LAM\_#d #+

LAM\_#w BINT131 DIFF\_OR\_ZERO\_

#MIN ( # )

STO\_#x ( )

2GETEVAL ( )

;

FALSE

;

BINT14 #=casedrop ( IZQUIERDA )

:: REPEATER

BINT14

:: LAM\_#x LAM\_#d DIFF\_OR\_ZERO\_ ( # )

STO\_#x ( )

2GETEVAL ( )

;

FALSE

;

BINT47 #=casedrop ( CANCL )

TRUE

BINT51 #=casedrop ( ENTER )

TRUE

DropBadKey FALSE

;

BINT2 #=casedrop

:: BINT15 #=casedrop ( PAGINA ABAJO )

:: REPEATER

BINT15

:: LAM\_#y BINT80 #+

LAM\_#h BINT80 DIFF\_OR\_ZERO\_

#MIN ( # )

STO\_#y ( )

2GETEVAL ( )

;

FALSE

;

BINT10 #=casedrop ( PAGINA ARRIBA )

:: REPEATER

BINT10

:: LAM\_#y BINT80 DIFF\_OR\_ZERO\_ ( # )

STO\_#y ( )

2GETEVAL ( )

;

FALSE

;

BINT16 #=casedrop ( PAGINA DERECHA )

:: REPEATER

BINT16

:: LAM\_#x BINT131 #+

LAM\_#w BINT131 DIFF\_OR\_ZERO\_

#MIN ( # )

STO\_#x ( )

2GETEVAL ( )

;

FALSE

;

BINT14 #=casedrop ( PAGINA IZQUIERDA )

:: REPEATER

BINT14

:: LAM\_#x BINT131 DIFF\_OR\_ZERO\_ ( # )

STO\_#x ( )

2GETEVAL ( )

;

FALSE

;

DropBadKey FALSE

;

BINT3 #=casedrop

:: BINT14 #=casedrop ( TODO IZQUIERDA )

:: BINT0 ( #0 )

STO\_#x ( )

2GETEVAL ( )

FALSE ( F )

;

BINT16 #=casedrop ( TODO DERECHA )

:: LAM\_#w BINT131 DIFF\_OR\_ZERO\_ ( # )

STO\_#x ( )

2GETEVAL ( )

FALSE ( F )

;

BINT10 #=casedrop ( TODO ARRIBA )

:: BINT0 ( #0 )

STO\_#y ( )

2GETEVAL ( )

FALSE ( F )

;

BINT15 #=casedrop ( TODO ABAJO )

:: LAM\_#h BINT80 DIFF\_OR\_ZERO\_ ( # )

STO\_#y ( )

2GETEVAL ( )

FALSE ( F )

;

BINT47 #=casedrop ( APAGAR )

:: TurnOff ( )

FALSE ( F )

;

DropBadKey FALSE

;

2DropBadKey FALSE

;

UNTIL

( )

ABND ( ) ( Destruye el entorno temporal creado )

;

Capítulo 40  
El Menú

La línea del menú es dividida en seis partes, una por cada tecla. Cada etiqueta de menú tiene una altura de 8 píxeles y un ancho de 21 píxeles. Las columnas iniciales para cada etiqueta de menu en HARBDUFF2 son:

**Hex Dec Softkey**

0 0 F1

16 22 F2

2C 44 F3

42 66 F4

58 88 F5

6E 110 F6

El comando DispMenu.1 redibuja el menu actual.

El comando DispMenu redibuja el menu actual y luego llama al comando SetDA3Valid para congelar el área del menú (display area 3).

Los siguientes comandos convierten varios tipos de objetos a etiquetas de menú y las muestran en la columna especificada.

**Comando Pila y acción**

Str>Menu ( #col $ 🡺 )

Muestra una etiqueta de menú estándar.

Id>Menu ( #col id 🡺 )

Muestra una etiqueta de menu estándar o de tipo directorio de acuerdo al contenido del id.

Grob>Menu ( #col grob 🡺 )

Muestra un grob 21x8 como etiqueta de menú.

Seco>Menu ( #col :: 🡺 )

Evalua el programa y usa el resultado para mostrar una etiqueta de menú.

El programa debe retornar una cadena, id o grob 21x8.

Los siguientes comandos convierten cadenas a los cuatro tipos diferentes de grobs de etiquetas de menú.

**Comando Pila y acción**

MakeStdLabel ( $ 🡺 grob )

Crea etiqueta de menú estándar.

Por ejemplo, si la cadena es "12345", retorna 

MakeBoxLabel ( $ 🡺 grob )

Crea etiqueta de menú con un cuadrado.

Por ejemplo, si la cadena es "12345", retorna 

MakeDirLabel ( $ 🡺 grob )

Crea etiqueta de menú tipo directorio.

Por ejemplo, si la cadena es "12345", retorna 

MakeInvLabel ( $ 🡺 grob )

Crea etiqueta de menú tipo cuadro inverso.

Por ejemplo, si la cadena es "12345", retorna 

40.1 Formato de Menú

Un menú puede ser:

**TIPO A**. Una lista de la forma:

{ MenuKey1

MenuKey2

...

MenuKeyN

}

El menú tendrá las propiedades por defecto.

**TIPO B**. Un programa que retorne una lista ( 🡪 {} ).

:: < Código >

{ MenuKey1

MenuKey2

...

MenuKeyN

}

;

< Código > es ejecutado cuando el menu pasa a ser el menú actual (por ejemplo al ejecutar los comandos InitMenu, DoMenuKey o StartMenu) y aquí puedes cambiar las propiedades del menú para que ya no tengan sus valores por defecto. En la siguiente sección se explican en detalle las propiedades de un menú.

**TIPO C**. Un programa que retorne un programa ( 🡪 Programa ).

:: < Código >

' ProgMenuData

;

ProgMenuData es un programa que debe retornar una lista de la forma:

{ MenuKey1

MenuKey2

...

MenuKeyn

}

**TIPO D**. Un programa que retorne un programa ( 🡪 Programa ).

:: < Código >

' ProgLoadMKeys

;

ProgLoadMKeys es un programa que no retorna nada. Además quita el primer nivel de la pila de retornos (ya no se cargarán los MenuKeys de la manera por defecto). Además crea y carga los MenuKeys con el comando LoadTouchTbl de la siguiente manera:

:: RDROP ( ) ( Quita 1º nivel de pila de retornos )

... ( MenuKey1 .. MenuKeyN #n )

LoadTouchTbl ( ) ( Carga los MenuKeys en TOUCHTAB )

;

Cada MenuKey es una lista con dos elementos LabelObj y KeyProc.

MenuKey puede ser un objeto de algunas de estas formas:

• { LabelObj #SystemFlag }

• { LabelObj id/lam/tag } ( por ejemplo: TAG 2 ID NOMB7 )

• { LabelObj ObjetoUnidad }

• { LabelObj KeyProcNS }

• { LabelObj { KeyProcNS } }

• { LabelObj { KeyProcNS KeyProcLS } }

• { LabelObj { KeyProcNS KeyProcLS KeyProcRS } }

• NullMenuKey

• KeyObj

**LabelObj** es el **primer elemento de la lista**. Es el objeto que será mostrado como etiqueta de menú. Si no cambias el valor por defecto de la propiedad LabelDef, este objeto puede ser de cualquier tipo, pero deberás tener en cuenta lo siguiente:

Si LabelObj es un programa con TakeOver como su primer elemento, sólo puede retornar una cadena, un grob o un bint.

Si LabelObj es una cadena, sólo serán mostrados los 5 primeros caracteres.

Si LabelObj es un grob, deberá tener un tamaño exacto de 21x8 para que se muestre directamente como etiqueta de menú.

Si LabelObj es un bint, se mostrará el mensaje de error correspondiente como etiqueta de menú.

Si LabelObj es un id que corresponde a una variable del directorio actual o de más arriba y este id corresponde a un directorio, la etiqueta será de tipo directorio. De lo contrario, será de tipo estándar.

Si LabelObj es un objeto etiquetado, sólo puede contener a un grob, cadena, programa, id, bint o entero.

**KeyProc** es el **segundo elemento de la lista**. Especifica lo que se realizará cuando el usuario presione una tecla de menú.

KeyProcNS indica la acción al presionar una tecla de menú (NS = No Shift).

KeyProcLS indica la acción al presionar shift izquierdo + tecla de menú (LS = Left Shift).

KeyProcRS indica la acción al presionar shift derecho + tecla de menú (RS = Right Shift).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **KeyProc:** | **bint** | **Id/lam** | **tag** | **unidad** |
| Tecla de menú | Cambia estado de flag de sistema | Evalua contenido | Evalua contenido | Multiplica por la unidad |
| Shift izquierdo + tecla de menú | Activa el flag de sistema | Guarda en el nombre | Guarda en el puerto | Convierte hacia la unidad |
| Shift derecho + tecla de menú | Desactiva el flag de sistema | Llama al contenido | Llama al contenido | Divide entre la unidad |

Al presionar una tecla de menú, KeyProc será ejecutado por un ejecutor especial que tomará las acciones apropiadas de acuerdo al tipo de objeto que sea KeyProc.

Si KeyProc es un programa con TakeOver como su primer comando, se hará caso omiso al ejecutor normal.

40.2 Propiedades de un Menú

El sistema de menus de la HP 50g tiene una flexibilidad impresionante. Más allá de las acciones normales, un menu tiene muchas propiedades las cuales definen la apariencia de las etiquetas, las acciones que se tomarán cuando se presionen las teclas, las acciones que se tomarán cuando el directorio actual cambie, las acciones a ejecutarse cuando un nuevo menú es instalado, etc.

Las propiedades que tiene todo menú son:

**Propiedad Tipo y Descripción**

MenuDef El menu actual.

Puede ser de alguno de los 4 tipos mostrados.

MenuData Si menu es una lista, MenuData es la misma lista.

Si menú es un programa, Menudata es el objeto que resulta al evaluarlo, pudiendo ser una lista u otro programa (ProgMenuData o ProgLoadMKeys).

MenuRow Es un bint. Es el índice de la 1º tecla de menú en la actual página del menú.

Debe ser 1 para la primera página, 7 para la segunda página, 13 para la tercera y asi sucesivamente.

**Propiedad Tipo y Descripción**

LabelDef Se encarga de dibujar las etiquetas de menú en la pantalla.

Es un programa de la forma: ( #col ob 🡪 )

Por ejemplo, el menu LIBS (Shift derecho + 2) usa el siguiente programa para hacer que todas las etiquetas de menú sean del tipo directorio.

:: ( #col ob ) ( en este menú ob es una cadena, )

( nombre de biblioteca o puerto )

DUPNULL$? ( #col ob flag )

ITE

MakeStdLabel

MakeDirLabel

( #col grob )

Grob>Menu ( )

;

MenuKeysNS Acción a realizarse cuando se presiona una tecla de menú.

Es un programa de la forma ( KeyProc 🡪 ?? )

Su valor por defecto es StdMenuKeyNS.

MenuKeysLS Acción a realizarse cuando se presiona shift izquierdo + una tecla de menú.

Es un programa de la forma ( KeyProc 🡪 ?? )

Su valor por defecto es StdMenuKeyLS.

MenuKeysRS Acción a realizarse cuando se presiona shift derecho + una tecla de menú.

Es un programa de la forma ( KeyProc 🡪 ?? )

MenuRowAct Acción a realizarse cuando cambia la fila del menu actual.

Su valor por defecto es SetDA12NoCh.

ExitAction Acción a realizarse cuando el menú cambia (cuando este menú dejará de ser el menú actual).

Su valor por defecto es SaveLastMenu. De esta manera se guarda el menú actual como LastMenu.

TrackAct Acción a realizarse cuando el contexto (directorio actual) cambia.

Su valor por defecto es NOP.

ReviewKey Acción a realizarse cuando es presionada la tecla ShiftDerecho + Abajo.

Su valor por defecto muestra en la pantalla el contenido de cada tecla de menú, de la siguiente manera:

:: ( )

DOCLLCD ( ) ( Limpia ABUFF sin cambiar su tamaño )

BINT7 ( #7 )

ONE\_DO

INDEX@ ( #i ) ( nº de tecla de menú: entre 1 y 6 inclusive )

DUP ( #i #i )

GETPROC ( #i ob )

>Review$ ( $ )

INDEX@ ( $ #i )

BIGDISPN ( ) ( Muestra $ en línea #i. Se verá en 1 sola línea )

LOOP

SetDA12Temp ( ) ( Congela las áreas DA1 y DA2 )

SetDA3NoCh ( ) ( no se cambiará el área DA3 )

;

**Propiedad Tipo y Descripción**

BadMenu? Es un flag. No debes cambiar su valor. Must the menu be be redrawn?

Rebuild? Es un flag. Si su valor es TRUE, entonces un cambio en el número de teclas de menú o en su posición relativa es mostrado inmediatamente cuando ocurre.

Su valor por defecto es FALSE.

En el menú VAR y en el menú LIB esta propiedad es fijada como TRUE.

Track? Es un flag. Si su valor es TRUE, entonces el objeto guardado en TrackAct se ejecutará cada vez que el directorio actual cambie.

Su valor por defecto es FALSE.

En el menú VAR y en el menú CST esta propiedad es fijada como TRUE.

Ejemplo de menu tipo B: CST.

Ejemplo de menu tipo C: LIB.

Ejemplo de menu tipo D: VAR.

40.3 Referencia

40.3.1 Propiedades de Menú

**Direcc. Nombre Descripción**

04A41 GETDF ( #menukey 🡺 ob )

Consigue la definición de una etiqueta de menú.

#menukey = #1...#6

ob puede ser cadena, grob21x8, id, bint, programa u objeto etiquetado.

04A0B GETPROC ( #menukey 🡺 KeyProc )

Consigue la definición de una tecla de menú.

Con #menukey = #1..#6, consigue el KeyProc.

Con #menukey = #7, consigue el ejecutor.

275FD MenuKey ( 🡺 ob )

Run Stream:

( #menukey 🡺 )

Con #menukey = #1..#6, consigue el KeyProc.

Con #menukey = #7, consigue el ejecutor.

25845 MenuDef@ ( 🡺 menu )

Llama a la definición del menú actual (MenuDef).

Menu es una lista, un programa, un rompointer o un flashpointer.

25840 (MenuDef!) ( menu 🡺 )

Guarda en MenuDef.

260BC MenuRow@ ( 🡺 #mrow )

Llama al valor guardado en el parámetro MenuRow.

Retorna 1 para la primera página, 7 para la segunda página, 13 para la tercera y asi sucesivamente.

260B7 MenuRow! ( #mrow 🡺 )

Guarda # como nuevo valor de la propiedad MenuRow.

2590D LastMenuDef@ ( 🡺 menu )

Llama a la definición del último menú (MenuDef).

Menu es una lista un programa, un rompointer o un flashpointer.

25908 LastMenuDef! ( menu 🡺 )

Fija la definición del último menú (MenuDef).

260AD LastMenuRow@ ( 🡺 #mrow )

Llama al valor del parámetro MenuRow del último menú.

260A8 LastMenuRow! ( #mrow 🡺 )

Fija el valor del parámetro MenuRow del último menú.

25EFB SaveLastMenu ( 🡺 )

La fila y la definición ( MenuRow y MenuDef ) del menú actual son guardadas como la fila y la definición del ultimo menú.

Hace :: MenuRow@ LastMenuRow! MenuDef@ LastMenuDef! ;

25890 MenuKeyNS@ ( 🡺 ob )

Retorna el objeto guardado como propiedad MenuKeyNS.

2588B MenuKeyNS! ( ob 🡺 )

Guarda ob como nuevo valor de la propiedad MenuKeyNS.

2589A DoMenuKeyNS ( ... KeyProc 🡺 ... )

Ejecuta el objeto guardado como propiedad MenuKeyNS.

**Direcc. Nombre Descripción**

25F03 StdMenuKeyNS ( ... KeyProc 🡺 ... )

Es el contenido de MenuKeyNS para un menú estándar.

258A4 (MenuKeyLS@) ( 🡺 ob )

Retorna el objeto guardado como propiedad MenuKeyLS.

2589F MenuKeyLS! ( ob 🡺 )

Guarda ob como nuevo valor de la propiedad MenuKeyLS.

258AE (DoMenuKeyLS) ( ... KeyProc 🡺 ... )

Ejecuta el objeto guardado como propiedad MenuKeyLS.

25F02 StdMenuKeyLS ( ... KeyProc 🡺 ... )

Es el contenido de MenuKeyLS para un menú estándar.

258B8 (MenuKeyRS@) ( 🡺 ob )

Retorna el objeto guardado como propiedad MenuKeyRS.

258B3 MenuKeyRS! ( ob 🡺 )

Guarda ob como nuevo valor de la propiedad MenuKeyRS.

258C2 (DoMenuKeyRS) ( ... KeyProc 🡺 ... )

Ejecuta el objeto guardado como propiedad MenuKeyRS.

25868 (MenuRowAct@) ( 🡺 ob )

Retorna el objeto guardado como propiedad MenuRowAct.

25863 MenuRowAct! ( ob 🡺 )

Guarda ob como nuevo valor de la propiedad MenuRowAct.

25872 (DoMenuRowAct) ( 🡺 )

Ejecuta el objeto guardado como propiedad MenuRowAct.

258F4 (MenuExitAct@) ( 🡺 ob )

Retorna el objeto guardado como propiedad ExitAction.

258EF (MenuExitAct!) ( ob 🡺 )

Guarda ob como nuevo valor de la propiedad ExitAction.

258FE (DoMenuExit) ( 🡺 )

Ejecuta el objeto guardado como propiedad ExitAction.

25EEF NoExitAction ( 🡺 )

Guarda NOP como nuevo valor de la propiedad ExitAction.

De esta manera, si el menú actual es abandonado, este menú no será LastMenu.

258E0 (TrackAct@) ( 🡺 ob )

Retorna el objeto guardado como propiedad TrackAct.

258DB (TrackAct! ) ( ob 🡺 )

Guarda ob como nuevo valor de la propiedad TrackAct.

258EA (DoTrack) ( 🡺 )

Ejecuta el objeto guardado como propiedad TrackAct.

257FC (SetTrack) ( 🡺 )

Fija la propiedad Track? como TRUE.

25801 (ClrTrack) ( 🡺 )

Fija la propiedad Track? como FALSE.

257F7 (Track?) ( 🡺 flag )

Retorna el valor de la propiedad Track?

258CC (ReviewKey@) ( 🡺 ob )

Retorna el objeto guardado como propiedad ReviewKey.

258C7 ReviewKey! ( ob 🡺 )

Guarda ob como nuevo valor de la propiedad ReviewKey.

258D6 DoReview ( 🡺 )

Ejecuta el objeto guardado como propiedad ReviewKey.

**Direcc. Nombre Descripción**

2580E SetRebuild ( 🡺 )

Fija la propiedad Rebuild? como TRUE.

25813 (ClrRebuild) ( 🡺 )

Fija la propiedad Rebuild? como FALSE.

25809 (Rebuild?) ( 🡺 flag )

Retorna el valor de la propiedad Rebuild?

25EE2 InitTrack: Run Stream:

( ob 🡺 )

Guarda ob como nuevo valor de la propiedad TrackAct.

Fija la propiedad Track? como TRUE.

Fija el directorio actual como el directorio LastContext.

El menú VAR lo usa para fijar la 1º fila al cambiar el directorio.

El menú CST lo usa para reiniciarse al cambiar el directorio.

40.3.2 Creando Menús

**Direcc. Nombre Descripción**

275C6 TakeOver ( 🡺 )

Pasa por alto el ejecutor por defecto de teclas de menú.

Si es el primer comando en un programa, el programa puede ser usado en modo de edición.

Si es el primer comando en un programa que es la definición de una etiqueta de menú, el programa es evaluado para conseguir el objeto de etiqueta de menú (mayormente un grob).

Al crear menús, normalmente se usa así:

:: TakeOver ' menu DoMenuKey ;

275EE Modifier ( 🡺 )

:: TakeOver ;

27620 MenuMaker Run Stream:

( ob 🡺 )

Cita al siguiente objeto del runstream y también llama al comando TakeOver.

Este comando hace: :: TakeOver 'R ;

Normalmente se usa así:

:: MenuMaker menu DoMenuKey ;

25EC6 DoMenuKey ( menu 🡺 )

Convierte el objeto de la pila en el nuevo menú actual y lo muestra con MenuRow=#1

menu debe estar en el formato descrito en la sección 40.1

Si es un programa, al evaluarse se sobreescriben las propiedades por defecto del menú.

Hace: :: SetDA12NoCh InitMenu ;

25EE0 InitMenu ( menu 🡺 )

Convierte el objeto de la pila en el nuevo menú actual y lo muestra con MenuRow=#1

Llama a StartMenu

**Direcc. Nombre Descripción**

25F00 StartMenu ( menu #menurow 🡺 )

Convierte el objeto de la pila en el nuevo menú actual.

#menurow es el índice de la 1º tecla de menú en la actual página del menú. Debe ser 1, 7, 13 y asi sucesivamente.

Primero ejecuta la propiedad ExitAction del menú actual (el que está a punto de abandonarse), luego fija las propiedades por defecto para el nuevo menú, y si este es un programa, entonces lo evalúa sobrescribiendo las propiedades por defecto y guarda las definiciones de teclas de menú llamando a SetThisRow.

27FED NullMenuKey ( 🡺 {} )

Con este comando puedes poner un MenuKey vacío en la lista del menú.

Equivale a: { NULL$ DoBadKey }

25EE1 InitMenu% ( %menu.pag 🡺 )

( %0 🡺 )

Si el argumento es 0, se instalará el último menú.

Si menu está entre 1 y 177, se instala un menú predefinido.

Si menú es mayor a 255, se instala un menú de una biblioteca.

pag es el número de página: 0.01, 0.02, 0.03, etc

25EFE SetThisRow ( 🡺 )

Crea un nuevo TOUCHTAB. Lo usan menús de tipos A, B y C.

25EE8 LoadTouchTbl ( MenuKey1 .. MenuKeyN #n 🡺 )

Crea un nuevo TOUCHTAB desde los MenuKeys.

#n está entre 0 y 6.

40.3.3 Mostrando el Menú

**Direcc. Nombre Descripción**

2EF66 SysMenuCheck ( 🡺 )

Hace lo siguiente de acuerdo a las propiedades del menú:

Si DA3NoCh?, entonces no hace nada.

Si Track? es TRUE, ejecuta TrackAct.

Si Rebuild? es TRUE, actualiza el menú con SetThisRow.

2DFCC ?DispMenu ( 🡺 )

Redibuja el menú ahora si ninguna tecla está esperando en el buffer.

Aun mejor es esto: :: DA3OK?NOTIT ?DispMenu ;

2DFF4 DispMenu.1 ( 🡺 )

Muestra el menu inmediatamente.

2DFE0 DispMenu ( 🡺 )

:: DispMenu.1 SetDAsValid ;

Redibuja el menu actual y luego llama a SetDA3Valid para congelar el área del menú (display area 3).

40.3.4 Mostrando Etiquetas de Menú y Propiedad LabelDef

**Direcc. Nombre Descripción**

2E0D5 Grob>Menu ( #col grob 🡺 )

Muestra el grob como una etiqueta de menú.

2E0F3 Str>Menu ( #col $ 🡺 )

Muestra la cadena como una etiqueta de menú (estándar).

2E11B Id>Menu ( #col id 🡺 )

Muestra el id como una etiqueta de menú.

Si el id corresponde a una variable del directorio actual o de más arriba y este id corresponde a un directorio, la etiqueta será de tipo directorio. De lo contrario, será de tipo estándar.

2E107 Seco>Menu ( #col prog 🡺 )

Hace EVAL luego DoLabel

2587C (LabelDef@) ( 🡺 ob )

Retorna el objeto guardado como propiedad LabelDef.

25877 LabelDef! ( ob 🡺 )

Guarda ob como nuevo valor de la propiedad LabelDef.

2E094 (StdLabelDef) ( #col ob 🡺 )

Ejecuta la acción por defecto de la propiedad LabelDef.

25886 DoLabel ( #col ob 🡺 )

Ejecuta el objeto guardado como propiedad LabelDef.

De esta manera muestra una etiqueta de menú.

Si LabelDef tiene su valor por defecto, entonces ob sólo puede ser grob, cadena, id, bint, etiquetado (que contenga a entero, id, grob, cadena, programa, id o bint) o programa que retorne cualquiera de los objetos anteriores. Si ob es de otro tipo, genera el error “Argumento incorrecto”.

08E007 ˆWRITEMENU ( $6 $5 $4 $3 $2 $1 🡺 )

Muestra las seis cadenas como etiquetas de menú (estándar).

40.3.5 Comandos Generales

**Direcc. Nombre Descripción**

25EA6 CheckMenuRow ( #n 🡺 #n #MenuRow )

Cambia el valor de MenuRow y retorna su nuevo valor.

#n es el número de elementos del menú.

Si MenuRow era menor o igual que #n, no lo cambia.

Si MenuRow era igual a #FFFFB, retorna el mayor número menor o igual que #n (y que sea multiplo de seis más uno).

Si MenuRow era mayor que #n (y diferente a #FFFFB), retorna #1.

25EFD SetSomeRow ( #i 🡺 )

Muestra el menú actual a partir de #i filas hacia adelante.

Para ir a la siguiente página del menú, escribe:

BINT6 SetSomeRow

Para ir a la página anterior, escribe:

# FFFFA SetSomeRow

25EC9 DoNextRow ( 🡺 )

Muestra la siguiente página del menú.

25ECB DoPrevRow ( 🡺 )

Muestra la anterior página del menú.

**Direcc. Nombre Descripción**

25EC3 DoFirstRow ( 🡺 )

Muestra la primera página del menú.

2F15B CLEARMENU ( 🡺 )

Limpia todo HARDBUFF2.

Este comando no hace invisible al menú. Por lo tanto, es diferente al comando TURNMENUOFF.

25854 (MenuData@) ( 🡺 {}/prog )

Si menu es una lista, MenuData es la misma lista.

Si menú es un programa, Menudata es el objeto que resulta al evaluarlo, pudiendo ser una lista u otro programa (ProgMenuData o ProgLoadMKeys).

2584F (MenuData!) ( {}/prog 🡺 )

2585E (GetMenuData) ( 🡺 {} )

Si el menú es del tipo A, retorna lista con MenuKeys.

Si el menú es del tipo B, retorna lista con MenuKeys.

Si el menú es del tipo C, evalúa ProgMenuData para retornar lista con MenuKeys.

No usarlo con menús del tipo D.

3EA01 (ID\_CST) ( 🡺 ob )

Evaluates ID CST.

2C2C0 nCustomMenu ( 🡺 {} )

Guarda propiedades del menú CST y retorna sus MenuKeys.

Para instalar el menu CST:

:: ' nCustomMenu InitMenu ;

25EFF SolvMenuInit ( 🡺 )

Fija algunas propiedades: MenuKeyNS/LS/RS, ReviewKey, LabelDef y ExitAction necesarias para el menú Solver.

25EDA GetMenu% ( 🡺 %menu.pag )

( 🡺 %0 )

Equivale al comando RCLMENU de User RPL.

40.3.6 Evaluando contenido del Menú

**Direcc. Nombre Descripción**

0000A3 ROMPTR 0A3 000 ( ... 🡺 ... )

Equivale a oprimir shift derecho + F1

0010A3 ROMPTR 0A3 001 ( ... 🡺 ... )

Equivale a oprimir shift derecho + F2

0020A3 ROMPTR 0A3 002 ( ... 🡺 ... )

Equivale a oprimir shift derecho + F3

0030A3 ROMPTR 0A3 003 ( ... 🡺 ... )

Equivale a oprimir shift derecho + F4

0040A3 ROMPTR 0A3 004 ( ... 🡺 ... )

Equivale a oprimir shift derecho + F5

0050A3 ROMPTR 0A3 005 ( ... 🡺 ... )

Equivale a oprimir shift derecho + F6

Equivale a oprimir una tecla de menú: ( ... 🡪 ... )

:: MenuKey BINT1 DoMenuKeyNS ;

:: MenuKey BINT2 DoMenuKeyNS ;

:: MenuKey BINT3 DoMenuKeyNS ;

:: MenuKey BINT4 DoMenuKeyNS ;

:: MenuKey BINT5 DoMenuKeyNS ;

:: MenuKey BINT6 DoMenuKeyNS ;

Equivale a oprimir shift izquierdo + una tecla de menú: ( ... 🡪 ... )

:: MenuKey BINT1 DoMenuKeyLS\_ ;

:: MenuKey BINT2 DoMenuKeyLS\_ ;

:: MenuKey BINT3 DoMenuKeyLS\_ ;

:: MenuKey BINT4 DoMenuKeyLS\_ ;

:: MenuKey BINT5 DoMenuKeyLS\_ ;

:: MenuKey BINT6 DoMenuKeyLS\_ ;

Equivale a oprimir shift derecho + una tecla de menú: ( ... 🡪 ... )

:: MenuKey BINT1 DoMenuKeyRS\_ ;

:: MenuKey BINT2 DoMenuKeyRS\_ ;

:: MenuKey BINT3 DoMenuKeyRS\_ ;

:: MenuKey BINT4 DoMenuKeyRS\_ ;

:: MenuKey BINT5 DoMenuKeyRS\_ ;

:: MenuKey BINT6 DoMenuKeyRS\_ ;

40.4 Ejemplos de Menús

**Ejemplo 1 Menú**

**Cuando KeyProc es un bint.**

Cuando KeyProc es un bint se puede cambiar el estado de un flag de sistema de la siguiente manera:

|  |  |
| --- | --- |
| **Tecla de menú** | **Acción** |
| Tecla de menú | Cambia estado de flag de sistema |
| Shift izquierdo + tecla de menú | Activa el flag de sistema |
| Shift derecho + tecla de menú | Desactiva el flag de sistema |

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME MenuFlags

:: CK0 ( ) ( No se requieren argumentos )

{

{ :: TakeOver "REL" BINT40 TestSysFlag Box/StdLabel ; BINT40 }

{ :: TakeOver BINT41 SysITE "24H" "AMPM" MakeBoxLabel ; BINT41 }

{ :: TakeOver BINT103 SysITE "COMP" "REAL" MakeBoxLabel ; BINT103 }

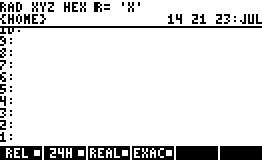
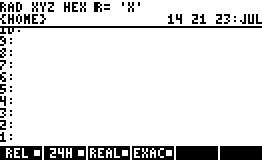
{ :: TakeOver BINT105 SysITE "APRO" "EXAC" MakeBoxLabel ; BINT105 }

}

BINT1 ( menu #mrow )

StartMenu ( )

;

**Ejemplo 2 Menú**

**Modificando las propiedades del menú VAR, para prevenir un cambio en el contenido de las variables.**

En este ejemplo se instala el menu VAR y luego se modifican las propiedades LabelDef y MenuKeyLS.

- La propiedad LabelDef es modificada para que las etiquetas de menú sean mostradas en modo inverso.

- La propiedad MenuKeyLS de manera que no se pueda cambiar el contenido de la variable al usar Shift derecho + tecla de menú. En lugar de eso, se mostrará un mensaje de alerta.

Nota que se protége el contenido de las variables de un cambio al presionar shift izquierdo + tecla de menú. No se protege la variable del uso del comando STO ni del cambio de su contenido a través del filer.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME MenuVarProt ( -> )

:: ( )

MenuMaker

:: ( )

ROMPTR 0A9 002 ( ) ( El menú VAR por defecto, menú tipo D )

' :: ( #col id/$ )

DUPTYPECSTR? ( #col id/$ flag )

NOT\_IT DECOMP$

( #col $ )

MakeInvLabel ( #col grob ) ( crea etiqueta de menú inversa )

Grob>Menu ( ) ( muestra etiqueta de menú )

;

LabelDef! ( ) ( fija la propiedad LabelDef )

' :: ( id/prog )

DUPTYPEIDNT? ( id/prog flag )

NOTcase

EVAL

( id )

DECOMP$ ( $ ) ( convierte a cadena sin comillas )

"\0A is protected" &$ ( $ )

FlashWarning ( )

;

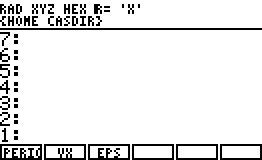
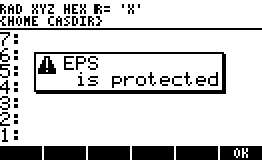
MenuKeyLS! ( ) ( fija la propiedad MenuKeyLS )

;

( prog ) ( menú tipo D )

DoMenuKey

;

**Ejemplo 3 Menú**

**Modificando las propiedades del menú VAR, para prevenir un cambio en el contenido de una variable que cumpla una condición.**

Este ejemplo es similar al anterior.

Pero ahora no se protegen a todas las variables. Sólo se protegen las variables que cumplan una condición (aquí la condición es que contenga a un objeto programa, aunque puedes cambiarla a cualquier otra condición).

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME MenuVarProt2 ( -> )

:: ( )

MenuMaker

:: ( )

ROMPTR 0A9 002 ( ) ( El menú VAR por defecto, menú tipo D )

' :: ( #col id/$ )

DUPTYPECSTR? ( #col id/$ flag )

case

Str>Menu ( sale con: )

( #col id )

DUP ( #col id id )

PROTECTED? ( #col id flag )

ITE

:: ID>$ ( #col $ )

MakeInvLabel ( #col grob ) ( crea etiqueta de menú inversa )

Grob>Menu ( ) ( muestra etiqueta de menú )

;

Id>Menu ( )

;

LabelDef! ( ) ( fija la propiedad LabelDef )

' :: ( id/prog )

DUPTYPEIDNT? ( id/prog flag )

NOTcase

EVAL

( id )

DUP

PROTECTED?

NOTcase

StdMenuKeyLS

DECOMP$ ( $ ) ( convierte a cadena sin comillas )

"\0Ais protected" &$ ( $ )

FlashWarning ( )

;

MenuKeyLS! ( ) ( fija la propiedad MenuKeyLS )

;

( prog ) ( menú tipo D )

DoMenuKey

;

\* Si id contiene a un objeto programa, entonces será variable protegida

\* Puedes cambiar este NULLNAME para proteger a una variable según

\* cualquier condición.

NULLNAME PROTECTED? ( id -> flag )

:: ( id )

@ ( ob T // F )

NOTcaseFALSE ( SALE CON: FALSE )

( ob )

TYPECOL? ( flag )

;

Cambiemos la condición para proteger a una variable.

Por ejemplo, si deseas proteger las variables de una lista ya conocida, puedes cambiar el NULLNAME PROTECTED? al siguiente:

\* Si id está en la lista, entonces será variable protegida

NULLNAME PROTECTED? ( id -> flag )

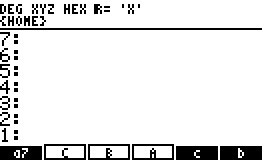
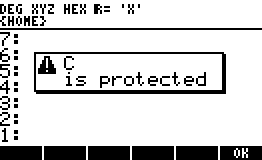
:: ( id )

{ ID A ID B ID C } ( id {id} )

FLASHPTR ListPos ( #pos/#0 )

#0<> ( flag )

;

**Ejemplo 3 Menú**

**Menú que se actualiza inmediatamente al variar el tamaño de la pila.**

En este ejemplo se fija la propiedad Rebuild como TRUE para actualizar el menú inmediatamente al producirse cambios en el número de elementos de la pila.

\* Este es un menú del tipo C.

\* El número de elementos de este menú es igual al tamaño de la pila,

\* siendo el tamaño máximo del menú igual a 9.

\* Al presionar la tecla NIV 1, se copia a la pila el obj del nivel 1

\* Al presionar la tecla NIV 2, se copia a la pila el obj del nivel 2

\* ....

\* Al presionar la tecla NIV 9, se copia a la pila el obj del nivel 9

\* En este ejemplo se puede apreciar el uso de la propiedad Rebuild.

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME MenuNPICK ( -> )

:: ( )

MenuMaker

:: ( ) ( fija propiedad Rebuild como TRUE )

SetRebuild ( )

' :: DEPTH ( #depth )

BINT9 ( #depth #9 )

#MIN ( # )

DUP#0=csedrp ( # flag )

NULL{} ( SALE CON: {} )

{ { "NIV 1" DUP }

{ "NIV 2" OVER }

{ "NIV 3" 3PICK }

{ "NIV 4" 4PICK }

{ "NIV 5" 5PICK }

{ "NIV 6" 6PICK }

{ "NIV 7" 7PICK }

{ "NIV 8" 8PICK }

{ "NIV 9" 9PICK\_ }

} ( # {{}} )

ONE ( # {{}} #1 )

ROT ( {{}} #1 # )

SUBCOMP ( {{}}' )

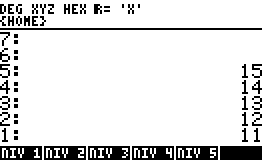
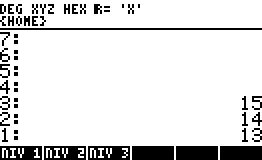
;

;

( prog ) ( menú tipo D )

DoMenuKey

;

Capítulo 41  
El Editor

Las calculadoras HP49 y HP50 tienen un editor ya incorporado, el cual es mucho más rápido y eficaz que el de la HP48. Sin embargo, este es un editor de propósito general, y podría ser útil para aplicaciones específicas agregando algunas características a tus programas sin tener que escribir un editor completo desde cero. De esta manera, podrás usar comandos para manipular el editor desde programas.

41.1 Terminología

Los siguientes términos aparecerán con frecuencia en este capítulo.

**Término Significado**

Linea de Edición La cadena que está siendo editada actualmente. También es llamada “Buffer” y “Línea de comandos”. En los diagramas de pila, usaremos $buf para mencionar a esta cadena.

Posición del Cursor La posición del cursor en la línea de edición. Representada por un bint. En los diagramas de pila, nos referimos a este bint como #cpos.

Línea Actual La línea actual en el editor. Es una subcadena comprendida desde después del SALTO DE LINEA antes del cursor hasta el siguiente SALTO DE LINEA.

Ventana del Editor Cuando el texto que se está editando es muy largo y/o muy ancho, la pantalla de la HP 50g muestra sólo una parte del texto: la ventana. Cuando el cursor es movido, a veces la ventana debe moverse también para mostrar la nueva posición del cursor.

Selección Una region en el buffer puede ser seleccionada cuando el marcador inicial y el marcador final están activos. La subcadena seleccionada es llamada $sel en los diagramas de pila.

Inicio de Palabra El comienzo de una palabra, una posición en una cadena donde el carácter anterior es ESPACIO o SALTO DE LINEA, y el carácter que se encuentra después no es un carácter en blanco. Varios comandos tratan con la posición del inicio de palabra. Esta posición es llamada #ws en los diagramas de pila de abajo.

Caracteres Invisibles La HP 50g puede mostrar texto en diferentes fuentes y estilos. Para alternar entre diferentes fuentes y estilos, marcadores especiales son insertados dentro del texto para indicar un cambio en la fuente o en el estilo. Estos marcadores son secuencias de tres caracteres los cuales no son visibles, pero si cuentan en la longitud de la cadena o para definir la posición del cursor. Algunos comandos que manejan al editor son concientes de estas cadenas y hacen cálculos complicados para cortar y pegar texto con estos atributos especiales. Esta es la razón por la cual estos comandos son más lentos de lo que podrían ser. Si tu no usas fuentes y estilos, no deberías preocuparte sobre esto.

41.2 Referencia

41.2.1 Estado

**Direcc. Nombre Descripción**

257A2 EditLExists? ( 🡺 flag )

Retorna TRUE si existe una línea de edición.

2EEED NoEditLine? ( 🡺 flag )

Retorna TRUE si no existe una línea de edición.

2F196 RCL\_CMD ( 🡺 $buf )

Retorna una copia de la línea de edición actual a la pila.

2EEEB EDITLINE$ ( 🡺 $buf )

Retorna una copia de la línea de edición actual a la pila.

Hace lo mismo que RCL\_CMD, pues sólo llama a este.

2F197 RCL\_CMD2 ( 🡺 $buf )

Similar a RCL\_CMD, pero si no hay suficiente memoria para copiar la línea de edición a la pila, este comando moverá la línea de edición actual a TEMPOB, mientras muestra en la línea de estado el mensaje

“Low Memory Condition

Please Wait…”.

De hecho, esto borrará la actual linea de edición.

2EF87 RCL\_CMD\_POS ( 🡺 # )

Retorna la posición actual del cursor.

26585 CURSOR@ ( 🡺 # )

Retorna la posición actual del cursor.

26594 (CURSOR\_PART) ( 🡺 # )

Retorna la fila (línea) donde se encuentra el cursor.

2F158 (ChrAtCur) ( 🡺 chr )

Retorna el carácter que está detrás del cursor.

Si el cursor está al final de la cadena, retorna el carácter número cero.

2EEEA CURSOR\_END? ( 🡺 flag )

Retorna TRUE, si el cursor está al final de una línea o al final de la cadena, o si debajo del cursor está el carácter cero.

264CC FIRSTC@ ( 🡺 # )

Retorna la columna del extremo izquierdo de la ventana.

26030 CURSOR\_OFF ( 🡺 # )

Retorna la posición actual del cursor relativa al extremo izquierdo de la ventana. Es decir, el número de caracteres de la línea actual entre el cursor y el extremo izquierdo de la línea actual (columna relativa al extremo izquierdo, menos uno).

2EF91 CAL\_CURS\_POS ( #línea #col 🡺 #c )

Calcula la posición en la actual línea de edición a partir de la línea y el número de columna especificadas. El resultado puede ser usado por el comando STO\_CURS\_POS para mover el cursor a esa ubicación. Si #line es mayor que el número de líneas en la línea de edición, halla la posición en la última línea.

**Direcc. Nombre Descripción**

2EF90 CAL\_CURS\_POS\_VIS ( #c'+1 🡺 #c )

#c' es la posición del cursor ignorando los caracteres invisibles.

#c es la posición del cursor al tomar en cuenta los caracteres invisibles.

Este comando es usado por el comando STO\_CURS\_POS\_VIS para mover el cursor a esa localización.

2F199 RCL\_CMD\_MODE ( 🡺 $data $buf )

Llama a una cadena $data la cual contiene el estado actual del editor (como la posición del cursor y la posición de la ventana).

El resultado puede ser usado por el comando STO\_CMD\_MODE para restaurar el estado de la línea de edición cuando se salió temporalmente del editor con HALT o cuando se llama a un programa que debe temporalmente cambiar el estado actual del editor.

2F198 STO\_CMD\_MODE ( $data $buf 🡺 )

Reinicia el editor en el estado en el que se dejó al llamar a RCL\_CMD\_MODE

41.2.2 Insertando Texto

**Direcc. Nombre Descripción**

25795 INSERT\_MODE ( 🡺 )

Activa el modo INSERT. En este modo, los nuevos caracteres escritos no se sobrescriben sobre los antiguos.

2579A REPLACE\_MODE ( 🡺 )

Activa el modo REPLACE. En este modo, los nuevos caracteres escritos se sobrescriben sobre los antiguos.

2577F (TOGGLE\_I/R) ( 🡺 )

Alterna la calculadora entre los modos INSERT y REPLACE.

25790 INSERT? ( 🡺 flag )

Retorna TRUE si está activado el modo INSERT.

Retorna FALSE si está activado el modo INSERT.

2EF74 CMD\_PLUS ( $/chr 🡺 )

Si está activo el modo INSERT, inserta la cadena en la posición actual del cursor en la línea de edición.

Si está activo el modo REPLACE, reemplaza la cadena a partir de la posición actual del cursor y sobrescribiendo los caracteres antiguos.

El cursor se moverá al final de esta cadena puesta.

Si no había línea de edición, se abre el editor y el cursor se posicionará al final de esta cadena.

**Direcc. Nombre Descripción**

2F194 CMD\_PLUS2 ( $ 🡺 )

Reemplaza la línea de edición completa con la cadena que está en la pila.

Si no hay suficiente memoria para copiar la cadena que está en el nivel uno de la pila, mueve la cadena afuera de TEMPOB, mientras muestra en la línea de estado el mensaje

“Low Memory Condition

Please Wait…”.

Debes asegurarte que la cadena no esté referenciada de ninguna manera.

Si no había línea de edición, se abre el editor y el cursor se posicionará al final de la nueva cadena.

2EF97 InsertEcho ( $/chr 🡺 )

Inserta la cadena en la posición del cursor sin importar el modo en el que se encuentra la calculadora.

El cursor se posicionará al final de la cadena insertada.

2EEE4 Echo$Key ( $/chr 🡺 )

Hace lo mismo que CMD\_PLUS, pues sólo llama a este.

2F11C Echo$NoChr00 ( $/chr 🡺 )

Hace lo mismo que CMD\_PLUS, pues sólo llama a este.

25EC1 DoDelim Run Stream:

( $/chr 🡺 )

Hace CMD\_PLUS con el siguiente objeto del runstream.

25EC2 DoDelims Run Stream:

( $/chr 🡺 )

Hace CMD\_PLUS con el siguiente objeto del runstream.

Luego, el cursor se mueve un carácter hacia atrás.

41.2.3 Borrando Texto

**Direcc. Nombre Descripción**

2EF82 CMD\_DEL ( 🡺 )

Borra el carácter que está debajo del cursor en el editor.

Si mantienes presionada la tecla BACKSPACE mientras este comando es ejecutado, la HP 50g repetirá esta acción mientras se mantenga presionada dicha tecla.

Equivale a apretar LS+BACKSPACE en el editor.

2EF81 CMD\_DROP ( 🡺 )

Borra el carácter que está antes del cursor en el editor.

Si mantienes presionada la tecla BACKSPACE mientras este comando es ejecutado, la HP 50g repetirá esta acción mientras se mantenga presionada dicha tecla.

Equivale a apretar BACKSPACE en el editor.

2EF95 DEL\_CMD ( 🡺 )

Limpia la línea de edición y finaliza el editor.

2EEE7 InitEdLine ( 🡺 )

Hace lo mismo que DEL\_CMD, pues sólo llama a este.

2F2F0 DO<Del ( 🡺 )

Borra los caracteres que están a la izquierda del cursor hasta el anterior comienzo de palabra.

Equivale a apretar 🡪DEL en el menú EDIT.

**Direcc. Nombre Descripción**

2F2F1 DO>Del ( 🡺 )

Borra los caracteres que están a la derecha del cursor hasta el comienzo de la siguiente palabra.

Equivale a apretar DEL🡪 en el menú EDIT.

2F2F9 DODEL.L ( 🡺 )

Borra todos los caracteres en la línea actual.

Si la línea actual ya está vacía, quita la línea borrando el SALTO de LINEA.

Equivale a apretar DEL\_L en el menú EDIT.

2F2DD DoFarBS ( 🡺 )

Borra desde el comienzo de la línea actual hasta la posición actual del cursor.

Equivale a apretar RS+🡪DEL en el menú EDIT.

2F2DE DoFarDel ( 🡺 )

Borra desde la posición actual del cursor hasta el final de la línea actual.

Equivale a apretar RS+DEL🡪 en el menú EDIT.

41.2.4 Moviendo el Cursor

**Direcc. Nombre Descripción**

2EF8B STO\_CURS\_POS ( # 🡺 )

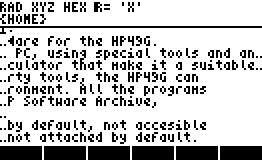
Pone el cursor en la posición especificada.

Si es necesario, también mueve la ventana del editor para asegurarse que el cursor sea visible.

Si es necesario mover la ventana del editor horizontalmente, este comando pone el cursor casi a la izquierda de la ventana y muestra tanto texto a la derecha del cursor como sea posible, como se muestra en la figura. Sin embargo, si el cursor es también visible cuando el extremo izquierdo de la ventana es la columna cero, esta posición de la ventana es la elegida.

#0 para poner el cursor al inicio de la cadena.

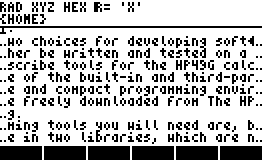
MINUSONE para poner el cursor al final de la cadena.



**Direcc. Nombre Descripción**

2EF8C STO\_CURS\_POS2 ( # 🡺 )

Similar a STO\_CURS\_POS, pero pone el cursor casi en el extremo derecho de la ventana como se indica en la figura.



2EF8D STO\_CURS\_POS3 ( # 🡺 )

Este comando es parecido a STO\_CURS\_POS, pero no verifica las secuencias de cambios de estilo o de fuente.

Mientras que al usar STO\_CURS\_POS la posición del cursor es corregida de tal manera que a la izquierda del cursor no hay caracteres invisibles, con este comando puedes posicionar el cursor en cualquier lugar, incluso entre caracteres invisibles.

2EF8E STO\_CURS\_POS4 ( # 🡺 )

Con respecto al posicionamiento de la ventana del editor se comporta como STO\_CURS\_POS2, pero con respecto a los caracteres invisibles se comporta como STO\_CURS\_POS3

2EF8F STO\_CURS\_POS\_VIS ( #c'+1 🡺 )

Mueve el cursor a la posición especificada.

#c' es la posición del cursor ignorando los caracteres invisibles.

Equivale a hacer

:: CAL\_CURS\_POS\_VIS STO\_CURS\_POS ;

2F378 SetCursor ( #posición 🡺 )

( {#fila #col} 🡺 )

Mueve el cursor a la posición indicada. Usado por el comando InputLine para fijar la posición inicial del cursor.

2EF7C CMD\_NXT ( 🡺 )

Mueve el cursor al carácter siguiente.

El cursor no debe estar al final de la cadena.

Equivale a apretar la tecla FLECHA DERECHA en el editor.

2EF7B CMD\_BAK ( 🡺 )

Mueve el cursor al carácter anterior.

El cursor no debe estar al inicio de la cadena.

Equivale a apretar la tecla FLECHA IZQUIERDA en el editor.

2EF80 CMD\_DOWN ( 🡺 )

Mueve el cursor a la siguiente línea.

El cursor no debe estar en la última línea.

Equivale a apretar la tecla FLECHA ABAJO en el editor.

2EF7F CMD\_UP ( 🡺 )

Mueve el cursor a la anterior línea.

El cursor no debe estar en la primera línea.

Equivale a apretar la tecla FLECHA ARRIBA en el editor.

2EF7D CMD\_DEB\_LINE ( 🡺 )

Mueve el cursor al inicio de la línea actual.

Equivale a apretar la tecla RS+IZQUIERDA en el editor.

**Direcc. Nombre Descripción**

2EF7E CMD\_END\_LINE ( 🡺 )

Mueve el cursor al final de la línea actual.

Equivale a apretar la tecla RS+DERECHA en el editor.

2EF7A CMD\_PAGED ( 🡺 )

Mueve el cursor una página hacia abajo.

Equivale a apretar la tecla LS+ABAJO en el editor.

2EF77 CMD\_PAGEL ( 🡺 )

Mueve el cursor una página hacia la izquierda.

Equivale a apretar la tecla LS+IZQUIERDA en el editor.

2EF78 CMD\_PAGER ( 🡺 )

Mueve el cursor una página hacia la derecha.

Equivale a apretar la tecla LS+DERECHA en el editor.

2EF79 CMD\_PAGEU ( 🡺 )

Mueve el cursor una página hacia arriba.

Equivale a apretar la tecla LS+ARRIBA en el editor.

2F2EE DO<Skip ( 🡺 )

Mueve el cursor al inicio de la palabra.

Llama a STO\_CURS\_POS2

Equivale a apretar 🡪SKIP en el menú EDIT.

2F2EF DO>Skip ( 🡺 )

Mueve el cursor al inicio de la siguiente palabra.

Llama a STO\_CURS\_POS

Equivale a apretar 🡪SKIP en el menú EDIT.

2F2E4 DO>BEG ( 🡺 )

Mueve el cursor al inicio de la selección (si está activa) o al inicio de la línea de edición (si la selección no está activa).

Equivale a apretar 🡪BEG en el menú EDIT.

2F2E5 DO>END ( 🡺 )

Mueve el cursor al final de la selección (si está activa).

Si no hay selección activa, entonces el cursor no se mueve.

Equivale a apretar 🡪END en el menú EDIT.

2F2E6 GOTOLABEL ( 🡺 )

Abre un cuadro de selección CHOOSE con las etiquetas en la actual línea de edición (las etiquetas son líneas cuyo primer carácter es \*).

Equivale a apretar LABEL en el menú EDIT/GOTO.

41.2.5 Seleccionar, Cortar y Pegar, el Portapapeles

**Direcc. Nombre Descripción**

2EF83 CMD\_STO\_DEBUT ( # 🡺 )

Establece la posición del marcador inicial.

Equivale a apretar RS+BEGIN.

2EF84 CMD\_STO\_FIN ( # 🡺 )

Establece la posición del marcador final.

Equivale a apretar RS+END.

2EF85 RCL\_CMD\_DEB ( 🡺 # )

( 🡺 #0 )

Llama a la posición del marcador inicial.

Si no hay selección, retorna ZERO.

**Direcc. Nombre Descripción**

2EF86 RCL\_CMD\_FIN ( 🡺 # )

( 🡺 #0 )

Llama a la posición del marcador final.

Si no hay selección, retorna ZERO.

2F2DC ClearSelection ( 🡺 )

Deselecciona el texto seleccionado sin cambiar el contenido del editor. Para esto, establece ambos marcadores a ZERO.

2EF93 VERIF\_SELECTION ( 🡺 flag )

Retorna TRUE cuando el marcador final no es ZERO, lo cual indica que la selección está activa.

Debes usar este comando como una verificación antes de hacer algo con la selección.

2EF8A CMD\_COPY ( 🡺 )

Si hay selección activa, copia la cadena seleccionada.

Si no hay selección activa, sólo hace beep.

Equivale a apretar RS+COPY cuando hay una selección activa.

2EF88 CMD\_CUT ( 🡺 )

En realidad este comando no corta una cadena.

Este comando quita del texto la cadena que es la selección actual, establece los marcadores inicial y final a cero y posiciona el cursor en el lugar donde estaba la selección.

Si no hay una selección activa, este comando no hace nada.

Una verdadera acción de cortar debería ser:

:: CMD\_COPY CMD\_CUT ;

2F2FA CMD\_COPY.SBR ( 🡺 $ )

Pone la selección como una cadena en la pila.

Este comando respeta las fuentes y estilos.

Si no usas fuentes y estilos, lo siguiente hace algo similar:

:: RCL\_CMD

RCL\_CMD\_DEB #1+ RCL\_CMD\_FIN

SUB$

;

2EF94 PASTE.EXT ( $ 🡺 )

Si está activo el modo INSERT, inserta la cadena en la posición actual del cursor en la línea de edición.

Si está activo el modo REPLACE, reemplaza la cadena a partir de la posición actual del cursor y sobrescribiendo los caracteres antiguos.

El cursor se moverá al final de esta cadena puesta.

Si no había línea de edición, se abre el editor y el cursor se posicionará al final de esta cadena.

Este comando respeta las fuentes y estilos.

Si no usas fuentes y estilos, es más rápido usar el comando CMD\_PLUS

2F2E1 SELECT.LINE ( 🡺 )

Selecciona toda la línea actual.

También mueve el cursor al inicio de la línea actual.

La selección no incluye al carácter SALTO DE LINEA que está al final de la línea.

2F2E2 SELECT.LINEEND ( 🡺 )

Selecciona toda la línea actual.

También mueve el cursor al final de la línea actual.

La selección no incluye al carácter SALTO DE LINEA que está al final de la línea.

**Direcc. Nombre Descripción**

2A085 (Clipboard!) ( $ 🡺 )

Guarda la cadena en el portapapeles.

2A095 (Clipboard@) ( 🡺 $ )

Retorna el contenido del portapapeles en la pila.

2A0A5 (Clipboard0) ( 🡺 )

Limpia el portapapeles.

2A0B5 (Clipboard?) ( 🡺 flag )

Retorna TRUE si hay algo en el portapapeles.

41.2.6 Buscar y Reemplazar

**Direcc. Nombre Descripción**

2F2F2 FindStrInCmd ( $buscada 🡺 $buscada #inicio #fin T )

( $buscada 🡺 $buscada F )

Encuentra una cadena en la línea de edición, empezando desde la actual posición del cursor. La cadena buscada permanece en la pila. Si esta es encontrada, retorna las posiciones inicial y final del cursor correspondientes a los extremos de la cadena hallada en la línea de edición, seguidas por TRUE.

Este comando respeta el estado actual de la calculadora en cuanto a diferenciar mayúsculas de minúsculas.

2F2E8 DOFIND ( 🡺 )

Abre el formulario de entrada FIND (buscar).

Equivale a apretar FIND en el menú EDIT/SEARCH.

2F2EA DONEXT ( 🡺 )

Encuentra la siguiente ocurrencia de la cadena a buscar que se encuentre después del cursor. Selecciona la cadena encontrada y coloca el cursor al final de esta selección.

Este comando respeta el estado actual de la calculadora en cuanto a diferenciar mayúsculas de minúsculas.

Si no existe una cadena a buscar, genera el error “Nonexistent Find Pattern” sin salir del editor.

Si no encuentra cadena alguna, genera el error “Not Found” sin salir del editor.

Equivale a apretar NEXT en el menú EDIT/SEARCH.

2F2E9 DOREPL ( 🡺 )

Abre el formulario de entrada FIND REPLACE (buscar y reemplazar).

Equivale a apretar REPL en el menú EDIT /SEARCH.

2F2EB DOREPLACE ( 🡺 )

Si existe una selección, reemplaza dicha selección con la cadena de reemplazo.

Si no existe una selección, inserta la cadena de reemplazo en la posición actual del cursor.

En ambos casos mueve el cursor hacia el final de la cadena puesta.

Si no existe una cadena que reemplazará a la buscada, genera el error “Nonexistent Replace Pattern” sin salir del editor.

Equivale a apretar R en el menú EDIT/SEARCH.

2F2EC DOREPLACE/NEXT ( 🡺 )

Equivale a hacer DOREPLACE y luego DONEXT.

Equivale a apretar R/N en el menú EDIT/SEARCH.

**Direcc. Nombre Descripción**

2F2ED REPLACEALL ( 🡺 )

Primero hace DOREPLACE

Luego halla la cadena a buscar y la cambia por la cadena a rremplazar. Hace esto varias veces hasta el final de la línea de edición.

Si no existe una cadena a buscar, genera el error “Nonexistent Find Pattern” sin salir del editor.

Si no existe una cadena que reemplazará a la buscada, genera el error “Nonexistent Replace Pattern” sin salir del editor.

Equivale a apretar ALL en el menú EDIT/SEARCH.

2F2FC REPLACEALLNOSCREEN ( 🡺 )

Similar a REPLACEALL, pero más rápido, pues no actualiza la pantalla tras cada reemplazo. Sólo actualiza la pantalla al final.

Equivale a apretar RS+ALL en el menú EDIT/SEARCH.

2A0C5 (FindPattern!) ( $ 🡺 )

Fija la cadena como aquella que se buscará.

2A0D5 (FindPattern@) ( 🡺 $ )

Retorna la cadena a buscar.

Usar este comando sólo cuando exista dicha cadena.

2A0E5 (FindPattern0) ( 🡺 )

Borra la cadena a buscar.

2A0F5 (FindPattern?) ( 🡺 flag )

Retorna TRUE si existe una cadena a buscar.

2A105 (ReplacePattern!) ( $ 🡺 )

Fija la cadena como aquella que reemplazará a la buscada.

2A115 (ReplacePattern@) ( 🡺 $ )

Retorna la cadena que reemplazará a la buscada.

Usar este comando sólo cuando exista dicha cadena.

2A125 (ReplacePattern0) ( 🡺 )

Borra la cadena que reemplazará a la buscada.

2A135 (ReplacePattern?) ( 🡺 flag )

Retorna TRUE si existe una cadena a buscar.

25772 (SetCaseSensitive) ( 🡺 )

Activa el modo de búsqueda que no diferencia mayúsculas de minúsculas.

25772 (ClrCaseSensitive) ( 🡺 )

Activa el modo de búsqueda que diferencia mayúsculas de minúsculas.

2576D (CaseSensitive?) ( 🡺 )

Retorna TRUE si está activo el modo de búsqueda que no diferencia mayúsculas de minúsculas.

41.2.7 Evaluación

**Direcc. Nombre Descripción**

2F2F8 EXEC\_CMD ( cmd algflag 🡺 obsel )

Ejecuta un comando sobre la selección actual de la línea de edición. Toma dos argumentos: el comando a ejecutar y un flag el cual nos indica como compilar la selección antes de que el comando es aplicado. Si el flag es TRUE y el modo algebraico está activado, será usado el compilador ALG y la etiqueta :: xEVAL del resultado será removida.

El resultado de la evaluación reemplaza a la selección actual.

Si hay más de un resultado, el último de estos reemplaza a la selección actual y los otros son colocados en la pila.

0B954 (RunInNewContext) ( ob 🡺 )

Guarda la actual interfaz de usuario, evalua ob y restaura la interfaz de usuario. Puede ser usado para ejecutar aplicaciones desde dentro de otra aplicación.

2F2DF EditSelect ( 🡺 )

Edita la selección actual usando ViewLevel1

Abre algún editor sólo con la selección. Al presionar ENTER solo el ultimo objeto es insertado en el entorno previo.

Si no hay selección, genera el error “Can´t Find selection” sin salir del editor.

Hace:

:: CK0NOLASTWD ' ViewLevel1 TRUE EXEC\_CMD ;

Equivale a apretar EDIT en el menú EDIT.

2F2FB EVAL.SELECTION ( 🡺 )

Evalua la selección actual y la reemplaza con el resultado de la evaluación.

Hace

:: CK0NOLASTWD ' EVAL FALSE EXEC\_CMD ;

Equivale a apretar EXEC en el menú EDIT.

2F2E3 EVAL.LINE ( 🡺 )

Evalua la linea actual y la reemplaza con el resultado de la evaluación. Similar a EVAL.SELECTION, pero sin necesidad de seleccionar la línea primero.

Hace

:: SELECT.LINE ' EVAL FALSE EXEC\_CMD ;

41.2.8 Empezando el Editor

**Direcc. Nombre Descripción**

2EEE9 EditString ( $ 🡺 )

Empieza la edición de la cadena cuando el programa actual termina. El cursor se posicionará al inicio de la cadena.

Este es el comando que debemos usar si un programa debe terminar con el editor activado.

Puedes usar antes el comando InitEdLine si deseas limpiar la línea de edición. De lo contrario, la cadena es insertada dentro de la línea de edición actual.

Todo el código que se encuentra después de este comando es ejecutado antes de que el control pase a manos de la aplicación editor.

Por ejemplo:

::

"CADENA A EDITAR" ( $ )

DUPLEN$ ( $ #n )

SWAP ( #n $ )

InitEdLine ( #n $ ) ( limpia línea de edición )

EditString ( #n ) ( Cadena a línea de edición )

STO\_CURS\_POS2 ( ) ( cursor al final )

"Empezando editor" ( $ )

FlashWarning ( ) ( muestra antes de editar )

;

2F1AD CharEdit ( ob 🡺 ob' T )

( ob 🡺 F )

Edita el objeto.

Cuando el usuario cancela, solo FALSE es retornado.

De lo contrario, el objeto cambiado y TRUE son retornados.

2EEE5 EditLevel1 ( ob 🡺 ob' )

Edita el objeto del nivel 1.

Llama a CharEdit

Equivale al comando EDIT de User RPL.

2F1AE ObEdit ( ob 🡺 ob' T )

( ob 🡺 F )

Edita el objeto.

Cuando el usuario cancela, solo FALSE es retornado.

De lo contrario, el objeto cambiado y TRUE son retornados.

2F19A ViewLevel1 ( ob 🡺 ob' )

Edita el objeto del nivel 1.

Llama a ObEdit

Equivale al comando EDITB de User RPL.

2F1AF AlgObEdit ( ob 🡺 ob' )

Este comando se usa para editar un objeto en lugar de ViewLevel1 cuando la calculadora está en modo algebraico. Este comando no llama a STARTED y EXITED.

2B2F2 (DoLevel1:) ( ob 🡺 ob' )

Evalua el siguinte objeto del runstream, el cual usualmente es un comando de edición como ObEdit o CharEdit

Cuando la evaluación retorna FALSE, el objeto original (que fue guardado en una variable temporal) es restaurado a la pila. Cuando la evaluación retorna TRUE, el TRUE es removido de la pila.

**Direcc. Nombre Descripción**

012004 ˆEQW3StartEdit ( ob 🡺 ext #0 )

( ob 🡺 ob #1 )

( ob 🡺 ob #2 )

Si el objeto se puede editar en el escritor de ecuaciones retorna ext y #0 (cuando el objeto es simbólico, id, lam, entero, real, complejo, algunos rompointers y caracteres).

Retorna #1 cuando necesita hacerse una recolección de basura (con GARBAGE).

Retorna #2 cuando el objeto no es apto para editarse en el escritor de ecuaciones.

011004 ˆEQW3Edit ( ob 🡺 symb/id/Z/%/C% T )

( ob 🡺 F )

( ob 🡺 ob' T )

Abre el editor de ecuaciones para editar la expresión.

Si sales con ENTER, retorna la nueva expresión y TRUE

Si sales con CANCEL, retorna sólo FALSE

Si el objeto no es apto para editarse en el escritor de ecuaciones, llama a CharEdit

073004 FLASHPTR 4 73 ( ob 🡺 symb/id/Z/%/C% T )

( ob 🡺 ob F )

Abre el editor de ecuaciones para editar la expresión.

Si sales con ENTER, retorna la nueva expresión y TRUE

Si sales con CANCEL, retorna la antigua expresión y FALSE

Si el objeto no es apto para editarse en el escritor de ecuaciones, genera el error “Argumento incorrecto”

Usado por el comando EQW de User RPL.

010004 ^EQW3 ( 🡺 symb/id/Z/%/C% T )

( 🡺 F )

Abre el editor de ecuaciones para crear una nueva expresión..

Si sales con ENTER, retorna la nueva expresión y TRUE.

Si sales con CANCEL, retorna sólo FALSE.

2F192 DoNewEqw ( 🡺 symb/id/Z/%/C% )

( 🡺 )

Abre el editor de ecuaciones para crear una nueva expresión.

Si sales con ENTER, retorna la nueva expresión.

Si sales con CANCEL, no retorna nada.

Llama a FLASHPTR EQW3

41.2.9 Miscelánea

**Direcc. Nombre Descripción**

2F2F3 GET.W-> ( 🡺 # )

Retorna la posición del siguiente inicio de palabra que esté a la derecha del cursor. Una palabra es una subcadena que no tiene caracteres ESPACIO ni SALTO DE LINEA.

2F2F4 GET.W<- ( #1 🡺 #' )

Retorna la posición del siguiente inicio de palabra que esté a la izquierda del cursor.

25ED2 EditMenu ( 🡺 prog )

Retorna el menú del editor. Este es un programa de dos elementos, el primero es un programa y el segundo una lista.

El resultado depende del estado del flag 95 (rpn o algebraico).

**Direcc. Nombre Descripción**

2EF73 ?Space/Go> ( 🡺 )

Inserta un carácter ESPACIO cuando no lo hay a la izquierda ni a la derecha del cursor.

Si es necesario mueve el cursor una posición a la derecha para que a la izquierda del cursor haya un carácter ESPACIO.

Pero si el cursor está al inicio de una línea, no hace nada.

Puedes usar este comando si quieres asegurarte que alguna palabra que vayas a insertar, esté separada de la palabra que la precede por al menos un espacio.

2EF76 AddLeadingSpace ( $ 🡺 $' )

Agrega un carácter ESPACIO al inicio de la cadena del nivel 1 de la pila, sólo si esta cadena no empieza con espacio y si el cursor en el editor está después de un carácter que no es espacio. Por ejemplo:

:: FALSE ( F )

"DUP" ( F $ )

AddLeadingSpace ( F $' )

AddTrailingSpace ( F $'' )

SWAPDROP ( $'' )

InsertEcho ( )

;

Inserta DUP en el editor y se asegura que esta cadena insertada esté rodeada de espacios.

2EF75 AddTrailingSpace ( flag $ 🡺 flag $' )

Si flag es TRUE, no hace nada.

Si flag es FALSE, agrega una carácter ESPACIO al final de la cadena, sólo si el carácter final de esta cadena no era espacio.

26238 GetFontCmdHeight ( 🡺 # )

Retorna la altura de la fuente usada al editar.

Si el flag 73 está activado (Edit:mini font), retorna la altura de la minifuente (6).

Si el flag 73 está desactivado (Edit:current fnt), retorna la altura de la fuente del sistema (6, 7 u 8).

2EF9A CommandLineHeight ( 🡺 #pix )

Retorna el número de píxeles de alto ocupado por la parte visible de la línea de edición.

Si quieres hallar el número de líneas visibles en la actual línea de edición puedes usar:

:: CommandLineHeight GetFontCmdHeight #/ SWAPDROP ;

2F2DB DOTEXTINFO ( 🡺 )

Muestra una pantalla con información sobre la línea de edición actual.

Equivale a apretar INFO en el menú EDIT del editor.

**Direcc. Nombre Descripción**

2F2F7 PUT\_STYLE ( # 🡺 )

Cambia el estilo, cuando hay una línea de edición activa.

Si hay una selección activa, cambia el estilo de texto en la selección.

Si no hay selección activa, cambia el estilo del texto tipeado a continuación.

Si el bint es 1, alterna a negrita.

Si el bint es 2, alterna a cursiva.

Si el bint es 3, alterna a subrayada.

Si el bint es 4, alterna a inversa.

Cuando hay una selección activa, PUT\_STYLE no hace ABND para su entorno temporal de manera que se debería de corregir ese bug.

2F2F5 PUT\_FONTE ( # 🡺 )

Cambia la fuente en un punto. Funciona de manera similar al comando PUT\_STYLE

2F2E7 SELECT.FONT ( 🡺 )

Muestra una caja de selección para escoger una fuente.

Equivale a apretar FONT en el menú EDIT/STYLE.

2F2F6 GET\_CUR\_FONT.EXT ( 🡺 # )

Retorna un bint que representa a la fuente usada por el caracter que se encuentra debajo del cursor.

2EF96 NO\_AFFCMD ( 🡺 )

Hace que la próxima vez que se sea llamado el comando CMD\_PLUS no se actualice la pantalla.

Puedes usarlo por velocidad, si quieres hacer más inserciones antes de que el usuario necesite verlas.

2EF92 XLINE\_SIZE? ( ob 🡺 flag )

Retorna TRUE si se cumplen las dos condiciones.

• El cursor se encuentra en una línea que no sea la última.

• El cursor se encuentra al final de la línea o más a la derecha.

Nota que este comando toma un objeto arbitrario de la pila. Por lo tanto, debes poner algo antes de llamar al comando.

27F47 <DelKey ( 🡺 {} )

Retorna el menukey correspondiente a 🡪DEL del menu EDIT.

27F9A >DelKey ( 🡺 {} )

Retorna el menukey correspondiente a DEL🡪 del menu EDIT.

27EAF <SkipKey ( 🡺 {} )

Retorna el menukey correspondiente a 🡪SKIP del menu EDIT.

27EFB >SkipKey ( 🡺 {} )

Retorna el menukey correspondiente a SKIP🡪 del menu EDIT.

2F19B OngoingText? ( 🡺 flag )

Si no existe línea de edición, retorna TRUE

Si existe una línea de edición pequeña o una en la que se han hecho pocas modificaciones, retona TRUE

Si existe una línea de edición grande y en la que se han hecho no pocas modificaciones, pregunta al usuario SI o NO. Retorna TRUE o FALSE respectivamente.

**Direcc. Nombre Descripción**

2EEE8 InitEdModes ( 🡺 )

Restablece algunos modos de la calculadora a su valor por defecto. Son los siguientes:

Pone la calculadora en modo inmediato (para esto desactiva los modos de entrada algebraica y programación).

Pone la calculadora en modo mayúsculas.

Remueve el teclado alfabético.

Activa el modo INSERT. En este modo, los nuevos caracteres escritos no se sobrescriben sobre los antiguos.

2EEE6 InitEd&Modes ( 🡺 )

:: InitEdLine InitEdModes ;

25636 HISTON? ( 🡺 flag )

Retorna TRUE si está activado el modo CMD (si se muestran las 4 últimas entradas confirmadas al apretar la tecla CMD).

2563B PTR 2563B ( 🡺 )

Activa el modo CMD.

0BCB2 PTR 0BCB2 ( 🡺 )

Desactiva el modo CMD.

2F326 CMDSTO ( $ 🡺 )

Agrega la cadena a la lista de las 4 últimas entradas confirmadas, accessibles por medio de la tecla CMD.

2F05E SaveLastEdit ( $ 🡺 )

Si el modo CMD está activado, llama a CMDSTO

Si el modo CMD está desactivado, sólo borra la cadena.

41.3 Ejemplos Editor

**Ejemplo 1. Editor**

Selecciona la línea actual y la copia al portapapeles.

::

TakeOver ( )

CMD\_END\_LINE ( ) ( Posiciona cursor al final de línea )

RCL\_CMD\_POS ( # ) ( Llama posición del cursor )

CMD\_STO\_FIN ( ) ( Fija posición de marcador final )

CMD\_DEB\_LINE ( ) ( Posiciona cursor al inicio de línea )

RCL\_CMD\_POS ( # ) ( Llama posición del cursor )

CMD\_STO\_DEBUT ( ) ( Fija posición de marcador inicial )

CMD\_COPY ( ) ( Copia al portapapeles )

;

Esto podría hacerse más corto usando el comando SELECT.LINE

El siguiente código es equivalente al de arriba.

::

TakeOver ( )

SELECT.LINE ( ) ( Selecciona la línea actual )

CMD\_COPY ( ) ( Copia al portapapeles )

;

**Ejemplo 2. Editor**

Inserta una plantilla ":: ;" en una línea única y posiciona el cursor entre "::" y ";"

::

TakeOver ( )

":: ;" ( $ )

CMD\_PLUS ( ) ( Inserta o reemplaza en la posición actual del cursor )

( Además coloca el cursor al final de esa cadena )

CMD\_BAK ( ) ( Mueve el cursor al carácter anterior )

CMD\_BAK ( ) ( Mueve el cursor al carácter anterior )

;

**Ejemplo 3. Editor**

Inserta una plantilla ":: ;" en tres líneas y posiciona el cursor en la segunda línea y en la segunda columna de dicha línea.

::

TakeOver ( )

"::\0A\0A;" ( $ )

CMD\_PLUS ( ) ( Inserta o reemplaza en la posición actual del cursor )

( Además coloca el cursor al final de esa cadena )

CMD\_UP ( ) ( Mueve el cursor a la anterior línea )

SPACE$ ( $ ) ( espacio )

CMD\_PLUS ( ) ( Coloca espacio y pone cursor al final de este espacio )

;

**Ejemplo 4. Editor**

Busca la siguiente etiqueta y va ahí. Las etiquetas son líneas que empiezan con "\*"

::

TakeOver ( )

"\0A\*" ( $find ) ( newline followed by star)

FindStrInCmd ( $find #inic #final T \\ $find F )

IT

:: ( $find #inic #final )

DROP ( $find #inic )

#1+ ( $find #inic+1 )

STO\_CURS\_POS ( $find ) ( Fija posición del cursor )

;

( $find )

DROP ( )

;

**Ejemplo 5. Editor**

Si deseas crear un editor que autocomplete palabras, será necesario conseguir el fragmento de palabra entre el inicio de la palabra y el cursor.

El siguiente programa retorna en el nivel 1 de la pila al fragmento entre el inicio de la palabra y el cursor.

::

TakeOver

RCL\_CMD ( $ ) ( Llama a la línea de edición )

RCL\_CMD\_POS ( $ #c ) ( Llama posición del cursor )

DUP ( $ #c #c )

GET.W<- ( $ #c #InicPalab ) ( Inicio de palabra a izq de cursor )

#1+SWAP ( $ #InicPalab+1 #c )

SUB$ ( $' ) ( Consigue una subcadena )

;

**Ejemplo 6. Editor**

Cambia la sangría de la línea actual a 4 espacios.

Las líneas que sólo tienen caracteres espacio no son modificadas.

Las líneas que empiezan con "\*" no son modificadas.

::

TakeOver ( )

BINT4 ( # )

Blank$ ( "\_" ) ( Crea una cadena con espacios )

CMD\_DEB\_LINE ( "\_" ) ( Mueve el cursor al inicio de la línea actual )

RCL\_CMD ( "\_" $ ) ( Retorna copia de la línea de edición )

RCL\_CMD\_POS ( "\_" $ #c ) ( Retorna la posición actual del cursor )

#1+ ( "\_" $ #c+1 )

SUB$1# ( "\_" #ASCII ) ( Retorna su código ASCII )

BINT42 ( "\_" #ASCII 42 ) ( código ASCII para \* )

OVER#=case

:: ( "\_" #ASCII ) ( línea empieza con \* )

2DROP ( )

CMD\_DOWN ( ) ( Mueve el cursor a la siguiente línea )

;

( "\_" #ASCII )

BINT32 ( "\_" #ASCII 32 )

#= ( "\_" flag )

IT

:: ( "\_" ) ( línea empieza con ESPACIO )

CMD\_END\_LINE ( "\_" ) ( Posiciona cursor al final de línea )

RCL\_CMD\_POS ( "\_" #fin ) ( Llama posición del cursor )

CMD\_DEB\_LINE ( "\_" #fin ) ( Posiciona cursor al inicio de línea )

DO>Skip ( "\_" #fin ) ( Mueve el cursor al inicio de la sgte palabra )

RCL\_CMD\_POS ( "\_" #fin #InicPalabra ) ( Llama posición del cursor )

#>ITE

DoFarBS ( "\_" ) ( Borra desde el comienzo de la línea actual )

( hasta la posición actual del cursor )

DROPRDROP ( ) ( Si el cursor ya está en la sgte línea, finaliza )

;

( "\_" )

CMD\_PLUS ( ) ( Inserta espacios )

CMD\_DEB\_LINE ( ) ( Posiciona cursor al inicio de línea )

CMD\_DOWN ( ) ( Mueve el cursor a la siguiente línea )

;

**Ejemplo 7. Editor en InputLine**

Para usar los programas que manejan el editor, debes asignarlos a una tecla o ponerlas en un menú. Nota que cada comando que escribes necesita un TakeOver como su primer elemento o de lo contrario, el programa no se ejecutará en el editor.

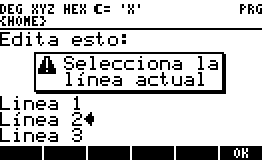
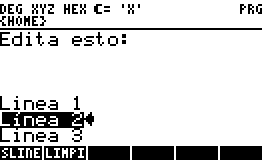
Aquí hay un ejemplo simple para el entorno InputLine el cual define un menú inicial con dos teclas de menú.

Con F1 se seleccionará toda la línea actual.

Con F2 se borra toda la línea de edición.

Con Shift Derecho más F1 o F2, puedes ver una ayuda sobre lo que hace la tecla de menú correspondiente.

Para más información sobre el comando InputLine, ver el capítulo 32.

xNAME MenúInputL

:: CK0 ( ) ( No se requieren argumentos )

"Edita esto:" ( $ ) ( $cabec )

"Linea 1\0ALínea 2" ( $ $ ) ( $inicial )

BINT0 ( ... ) ( CursPos: cursor al final )

BINT0 ( ... ) ( #Ins/Rep: modo actual )

BINT0 ( ... ) ( #ALG: modo actual )

BINT0 ( ... ) ( #alpha: modo actual )

{ { "SLINE"

{ :: TakeOver ( )

SELECT.LINE ( ) ( Selecciona toda la línea actual )

;

Modifier

:: TakeOver ( )

"Selecciona la línea actual" ( $ )

FlashWarning ( )

;

}

}

{ "LIMPI"

{ :: TakeOver ( )

DEL\_CMD ( ) ( Quita la línea de edición y finaliza editor )

NULL$ ( $ )

EditString ( ) ( Empieza la edición de la cadena )

;

Modifier

:: TakeOver ( )

"Límpia toda la línea de edición" ( $ )

FlashWarning ( )

;

}

}

}

( ... ) ( menú )

ONE ( ... ) ( fila inicial del menu )

TRUE ( ... ) ( CANCEL aborta inmediatamente la edición )

BINT0 ( ... ) ( retorna sólo cadena sin evaluar )

InputLine ( $ T // F )

;

Capítulo 42  
Trazado de Gráficos

Los comandos de este capítulo tratan con aspectos relacionados al trazado de gráficos. Hay varios comandos que tratan con la variable reservada PPAR, que contiene los parámetros usados en el trazado de gráficos. Esta variable es una lista con los siguientes parámetros:

{ (xmin, ymin) (xmax, ymax) indep resolucion ejes tipo dependiente}

Este es el significado de cada uno de los parámetros para la HP 50G.

|  |  |  |
| --- | --- | --- |
| **Parámetro** | **Descripción** | **Valor por** defecto |
| (xmin, ymin) | Un número complejo que representa las coordenadas de la esquina inferior izquierda en la ventana visible. | (−6.5,−3.9) |
| (xmax, ymax) | Un número complejo que representa las coordenadas de la esquina superior derecha en la ventana visible. | (6.5, 4.) |
| Indep  {Indep x1 x2} | La variable independiente o también una lista con dicha variable y su rango de trazado. | X |
| res | Resolución. Un número que representa el intervalo entre los puntos graficados. | 0 |
| Axes  C%  {C% {Ax% Ay%}}  {C% {Axh Ayh}} | Un número complejo que representa las coordenadas de intersección de los ejes. También puede ser una lista que además contenga a la distancia entre las marcas moostradas en los ejes. | (0, 0) |
| type | Es un comando User RPL que indica el tipo de gráfico que será trazado. | FUNCTION |
| Depend  {dep y1 y2}  {dep [%] y2} | Variable dependiente. | Y |

39.1 Referencia

39.1.1 Comandos Generales

**Direcc. Nombre Descripción**

2F162 CHECKPICT ( 🡺 )

Cambia las dimensiones de GBUFF, si este es pequeño.

Fuerza a que tenga un ancho mínimo de 131 y una altura mínima de 80.

2EF06 CKPICT ( xPICT 🡺 )

Verifica que xPICT (el comando, no un grob) se encuentre en el nivel 1 de la pila.

Si no es así, genera el error “Argumento incorrecto”.

2F258 PICTRCL ( xPICT 🡺 grob )

Retorna una copia de GBUFF a la pila.

Hace CKPICT, luego llama a GBUFF y luego TOTEMPOB

Equivale al comando RCL de User RPL cuando en la pila se encuentra xPICT.

2F105 GDISPCENTER ( 🡺 )

Primero, fuerza a que GBUFF tenga un ancho mínimo de 131 y una altura mínima de 80.

Luego, establece a GBUFF como la pantalla actual.

Finalmente, mueve la ventana al centro de GBUFF.

2EF60 DOGRAPHIC ( 🡺 )

Primero hace GDISPCENTER

Luego, hace TURNMENUOFF

Finalmente ejecuta un POL, en el que podremos desplazarnos a través de GBUFF con las teclas de dirección.

Las teclas disponibles son:

|  |  |
| --- | --- |
| **🡺,🡻,🡸,🡹** | Mueve un pixel |
| * con   **🡺,🡻,🡸,🡹** | Mueve a los extremos de GBUFF |
| TECLA CAMBIO DERECHA NARANJATECLA - ON | Apaga la calculadora |
| TECLA - ON | Sale del POL y establece a ABUFF como la pantalla actual |
| TECLA CAMBIO IZQUIERDA AZULTECLA - FLECHA IZQUIERDA | Sale del POL y ejecuta otro POL con un cursor y con menús de graficar funciones. |

Equivale al comando PVIEW de User RPL cuando en la pila hay una lista vacía.

2F297 XEQPURGEPICT ( xPICT 🡺 )

Si en el nivel uno de la pila se encuentra xPICT, borra GBUFF estableciendolo como un grob nulo. De lo contrario, genera el error “Argumento incorrecto”.

Equivale al comando PURGE de User RPL cuando en la pila se encuentra xPICT.

260A3 KILLGDISP ( 🡺 )

Borra GBUFF y lo establece como un grob nulo.

El mismo efecto que hacer PICT PURGE en User RPL.

Vea también el comando DOERASE

39.1.2 Variable PPAR

**Direcc. Nombre Descripción**

2F355 MAKEPVARS ( 🡺 {} )

Crea la variable PPAR en el directorio actual con sus valores por defecto y retorna su valor.

2F163 CHECKPVARS ( 🡺 {} )

Llama al contenido de PPAR del directorio actual.

Si PPAR no existe en el directorio actual, lo crea.

Si existe un PPAR inválido, genera el error “PPAR inválido”.

2F33D GETPARAM ( # 🡺 ob )

Extrae el objeto de orden # de la lista PPAR.

Primero, llama a CHECKPVARS

265001 FLASHPTR 1 265 ( ob #8-i 🡺 )

Guarda ob en la posición i de la variable PPAR.

Primero, si PPAR no existe en el directorio actual, lo crea.

Verifica que PPAR contenga una lista de 7 elementos. De lo contrario, genera el error “PPAR inválido”

312001 FLASHPTR 1 312 ( flag 🡺 )

Si el flag es TRUE, no hace nada.

Si el flag es FALSE, genera el error “PPAR inválido”.

39.1.3 Coordenadas de los extremos de GBUFF

**Direcc. Nombre Descripción**

2F0FF GETXMIN ( 🡺 % )

Llama a XMIN de la lista PPAR.

Primero, llama a CHECKPVARS

2F366 PUTXMIN ( % 🡺 )

Fija un nuevo valor para XMIN en la lista PPAR.

Primero, llama a CHECKPVARS

2F0FE GETXMAX ( 🡺 % )

Llama a XMAX de la lista PPAR.

Primero, llama a CHECKPVARS

2F365 PUTXMAX ( % 🡺 )

Fija un nuevo valor para XMAX en la lista PPAR.

Primero, llama a CHECKPVARS

2F100 GETYMIN ( 🡺 % )

Llama a YMIN de la lista PPAR.

Primero, llama a CHECKPVARS

2F368 PUTYMIN ( % 🡺 )

Fija un nuevo valor para YMIN en la lista PPAR.

Primero, llama a CHECKPVARS

2F10E GETYMAX ( 🡺 % )

Llama a YMAX de la lista PPAR.

Primero, llama a CHECKPVARS

2F367 PUTYMAX ( % 🡺 )

Fija un nuevo valor para YMAX en la lista PPAR.

Primero, llama a CHECKPVARS

**Direcc. Nombre Descripción**

2F107 GETPMIN&MAX ( 🡺 C% C% )

Retorna PMIN y PMAX en la pila.

Es decir, retorna (xmin, ymin) y (xmax, ymax)

Primero, si PPAR no existe en el directorio actual, lo crea.

Verifica que PPAR contenga una lista de 7 elementos y que sus 2 primeros elementos sean complejos. De lo contrario, genera el error “PPAR inválido”

39.1.4 Variable Independiente y Rango de Trazado

**Direcc. Nombre Descripción**

2F106 GETINDEP ( 🡺 id )

( 🡺 {id % %'} )

Retorna el tercer término de PPAR (la variable independiente o una lista con dicha variable y su rango de trazado)

Los números reales podrían estar en desorden.

Primero, llama a CHECKPVARS

2F0E8 INDEPVAR ( 🡺 id )

Retorna la variable independiente.

Primero, llama a CHECKPVARS

2EEF2 PUTINDEP ( id 🡺 )

( {id % %'} 🡺 )

Si en la pila está id, lo fija como la variable independiente respecto al trazado (tercer término de PPAR). Si existía un rango de trazado, este es borrado.

El argumento también podría ser una lista, pero en este caso no se verifican sus elementos.

Llama a FLASHPTR 1 265

Equivale al comando INDEP de User RPL cuando en la pila hay un id.

2EEF3 PUTINDEPLIST ( {id} 🡺 )

( {id % %'} 🡺 )

( {% %'} 🡺 )

Fija el valor que tendrá el tercer término de la variable PPAR, es decir, la variable independiente y su rango de trazado.

Si es {id}, fija la variable independiente, pero no cambia el rango de trazado.

Si es {% %'}, fija el rango de trazado, pero no cambia la variable independiente.

Si es {id % %'}, fija la nueva variable independiente y el nuevo rango de trazado.

% y %' pueden estar en desorden.

Llama a FLASHPTR 1 265

Equivale al comando INDEP de User RPL cuando en la pila hay una lista.

39.1.5 Resolución

**Direcc. Nombre Descripción**

2F10D GETRES ( 🡺 % )

( 🡺 hxs )

Retorna la resolución del trazado (cuarto elemento de PPAR).

Primero, llama a CHECKPVARS

2EEF4 PUTRES ( % 🡺 )

( hxs 🡺 )

Fija una nueva resolución del trazado.

El número real debe ser positivo, cero o menos uno.

Llama a FLASHPTR 1 265

Equivale al comando RES de User RPL.

39.1.6 Ejes y sus Etiquetas

**Direcc. Nombre Descripción**

2A4001 FLASHPTR 1 2A4 ( 🡺 C% )

( 🡺 {C% atick} )

( 🡺 {C% “xlabel” “ylabel”} )

( 🡺 {C% atick “xlabel” “ylabel”} )

Retorna el quinto elemento de PPAR.

Primero llama a CHECKPVARS

2A8001 FLASHPTR 1 2A8 ( 🡺 “xlabel” “ylabel” T )

( 🡺 F )

Llama a las etiquetas de los ejes, si estas existen en el quinto elemento de PPAR.

Primero llama a CHECKPVARS

254001 FLASHPTR 1 254 ( C% 🡺 )

( {C% atick} 🡺 )

( {C% “xlabel” “ylabel”} 🡺 )

( {C% atick “xlabel” “ylabel”} 🡺 )

Si en la pila está C%, lo fija como el punto de intersección de los ejes (guarda como quinto elemento de PPAR). Si existía atick y/o etiquetas de ejes, estas serán borradas.

Llama a FLASHPTR 1 265

Equivale al comando AXES de User RPL cuando en la pila hay un número complejo.

255001 FLASHPTR 1 255 ( {C%} 🡺 )

( {C% atick} 🡺 )

( {C% “xlabel” “ylabel”} 🡺 )

( {C% atick “xlabel” “ylabel”} 🡺 )

Si en la pila está C%, lo fija como el punto de intersección de los ejes. Pero no se cambiarán atick ni las etiquetas.

En los otros casos, cuando en la pila hay una lista, guarda esta lista como el quinto elemento de PPAR.

Llama a FLASHPTR 1 265

Equivale al comando AXES de User RPL cuando en la pila hay una lista.

**Direcc. Nombre Descripción**

0780AB ROMPTR 0AB 078 ( Z/%/hxs 🡺 )

( {Z/% Z'/%'} 🡺 )

( {hxs hxs'} 🡺 )

Guarda atick en el quinto elemento de PPAR.

Llama a FLASHPTR 1 265

Equivale al comando ATICK de User RPL.

2CA001 FLASHPTR 1 2CA ( 🡺 )

Dibuja los ejes horizontal y vertical. El punto de intersección de los ejes es un número complejo (quinto elemento de PPAR, y si este es una lista, su primer elemento).

La distancia entre las marcas de los ejes está dada por atick

Primero llama a CHECKPICT y CHECKPVARS

Equivale al comando DRAX de User RPL.

0A7001 FLASHPTR 1 A7 ( 🡺 )

Dibuja las etiquetas y sus valores extremos en ambos ejes.

Si en el quinto elemento de PPAR, no hay cadenas, entonces las etiquetas serán las variables independiente y dependiente.

Primero llama a CHECKPICT y CHECKPVARS

Equivale al comando LABEL de User RPL.

39.1.7 Tipo de Gráfico

**Direcc. Nombre Descripción**

2EEF5 GETPTYPE ( 🡺 name )

Retorna el tipo de gráfico (sexto elemento de PPAR).

Primero, llama a CHECKPVARS

2EEF6 PUTPTYPE ( name 🡺 )

Fija el nuevo tipo de gráfico (sexto elemento de PPAR).

Llama a FLASHPTR 1 265

2C37D PTYPE>PINFO ( name # 🡺 ob T )

( name # 🡺 F )

39.1.8 Variable Dependiente

**Direcc. Nombre Descripción**

2B3001 FLASHPTR 1 2B3 ( 🡺 id )

( 🡺 {id % %'} )

( 🡺 {id [%] %'} )

Retorna el séptimo elemento de PPAR (la variable dependiente o una lista con dicha variable y dos elementos más).

Primero, llama a CHECKPVARS

2E5001 FLASHPTR 1 2E5 ( 🡺 id )

Retorna la variable dependiente.

Primero, llama a CHECKPVARS

05B002 FLASHPTR 2 5B ( id 🡺 )

( {id % %'} 🡺 )

( {id [%] %'} 🡺 )

Si en la pila se encuentra un objeto como los indicados, no hace nada. De lo contrario, genera el error “PPAR inválido”.

39.1.9 Escala

**Direcc. Nombre Descripción**

2F33E GETSCALE ( 🡺 %Xscale %Yscale )

Llama a ambos parámetros de escala.

%Xscale es la distancia en unidades de usuario de dos puntos separados 10 píxeles en una recta horizontal.

Xscale=10•(Xmax-Xmin)/(ancho-1)

%Yscale es la distancia en unidades de usuario de dos puntos separados 10 píxeles en una recta vertical.

Yscale=10•(Ymax-Ymin)/(altura-1)

Llama a CHECKPVARS

2EEF1 PUTSCALE ( %Xscale %Yscale 🡺 )

Fija nuevos parámetros de escala (el centro no cambia).

Para esto cambia los dos primeros elementos de PPAR:

(xmin, ymin) y (xmax, ymax).

Primero, llama a GETPMIN&MAX

Los dos números reales deben ser positivos. De lo contrario, genera el error “Argumento: valor incorr”

Equivale al comando SCALE de User RPL.

2EEF8 HSCALE ( %Xfactor 🡺 )

Multiplica %Xscale por %Xfactor (el centro no cambia).

Para esto cambia los elementos xmin y xmax de PPAR.

Primero, llama a GETPMIN&MAX

El número real debe ser positivo. De lo contrario, genera el error “Argumento: valor incorr”

Equivale al comando SCALEW de User RPL.

2EEF7 VSCALE ( %Yfactor 🡺 )

Multiplica %Yscale por %Yfactor (el centro no cambia).

Para esto cambia los elementos ymin y ymax de PPAR.

Primero, llama a GETPMIN&MAX

El número real debe ser positivo. De lo contrario, genera el error “Argumento: valor incorr”

Equivale al comando SCALEH de User RPL.

2EEEF AUTOSCALE ( 🡺 )

Cambia los dos primeros eIementos de PPAR según el tipo de gráfico que se va a trazar.

Llama a CHECKPVARS

Equivale al comando AUTO de User RPL.

39.1.10 Variable EQ

**Direcc. Nombre Descripción**

25ECF EQUATION ( 🡺 ob T )

( 🡺 F )

Si EQ existe en el directorio actual, pone en la pila su contenido y TRUE.

Si no existe sólo pone FALSE.

**Direcc. Nombre Descripción**

2F339 GetEqN ( #i 🡺 ob T )

( #i 🡺 F )

Si EQ es un lista, cuyos elementos son objetos simbólicos:

• Si #i es válido, retorna la ecuación de lugar #i y TRUE

• Si #i no es válido, retorna FALSE

Si EQ no es una lista:

• Si #i es 1, retorna el contenido de EQ y TRUE

• Si #i es diferente de 1, retorna FALSE

25EB5 DORCLE ( 🡺 ob )

Llama al contenido de la variable EQ del directorio actual.

Si no existe, genera el error “Ecuación inexistente”.

Equivale al comando RCEQ de User RPL.

25EB6 DOSTOE ( ob 🡺 )

Guarda ob en la variable EQ en el directorio actual.

Equivale al comando STEQ de User RPL.

39.1.11 Coordenadas Relativas y Absolutas

**Direcc. Nombre Descripción**

2EF01 DOPX>C ( { hxs hxs' } 🡺 C% )

Convierte una lista de dos hxs a número complejo que representa sus coordenadas en el gráfico.

Llama a GETPMIN&MAX

Equivale al comando PX🡪C de User RPL.

2EF02 DOC>PX ( C% 🡺 { hxs hxs' } )

Convierte un número complejo a una lista con dos hxs.

Es la operación inversa de DOPX>C

Llama a GETPMIN&MAX

Equivale al comando C🡪PX de User RPL.

2F31F C%># ( C% 🡺 # #' )

Parecido al comando DOC>PX, pero retorna 2 bints.

Llama a GETPMIN&MAX

2CB001 FLASHPTR 1 2CB ( 🡺 )

Dibuja el gráfico correspondiente a la ecuación de EQ o a la formación en ΣDAT con los parámetros de PPAR (o VPAR o ΣPAR) EN GBUFF.

No dibuja ejes ni etiquetas, tampoco borra gráficos anteriores.

Se ve afectado por el flag 28 (gráficos secuenciales o simultáneos) y el flag 31 (puntos conectados o sólo puntos)

Primero llama a CHECKPICT, GDISPCENTER, TURNMENUOFF y CHECKPVARS

Equivale al comando DRAW de User RPL.

Parte IV

El CAS de

la HP 50g

Capítulo 43  
Introducción al CAS de la HP 50g

Una de las mayores innovaciones en la HP 49G fue el poderoso Computer Algebra System (CAS) que ya viene incorporado. El CAS de la HP49G principalmente se derivó de las bibliotecas ALG48 y ERABLE, originalmente escritas para las calculadoras HP48. Pero, a partir de la HP 49G, el CAS está completamente integrado al sistema operativo, de tal manera que usando User RPL se puede acceder a las funciones del CAS.

Un gran número de entradas soportadas dan acceso a los comandos internos del CAS (System RPL), permitiendonos escribir programas que tratan con objetos simbólicos, matrices simbólicas y números enteros de precisión infinita.

43.1 Problemas con Estos Capítulos

La versión inicial de la lista de referencia de comandos del CAS para este libro se deriva de la fuente de archivos de las bibliotecas ALG48 y ERABLE.

El problema con este enfoque es que en la fuente los comandos no están ordenados de acuerdo a su funcionalidad. Más bien, cada archivo fuente maneja una cierta área de comandos del CAS y muchas rutinas de diversa funcionalidad son incluidas dentro del mismo archivo.

Por esta razón hay varias ubicaciones donde por ejemplo comandos que manejan a objetos meta pueden ser encontrados en archivos diferentes. Incluso hay comandos similares (que al parecer hacen lo mismo) que se encuentran en diferentes archivos. Se ha hecho un importante esfuerzo para reordenar las entradas por su funcionalidad, pero damos sólo ha habido un éxito parcial. Un conocimiento más profundo del CAS y de sus componentes internos se necesita para completar este trabajo.

Una documentación completa del CAS también debería contener abundante material sobre la representación interna de los objetos del CAS, y muchos ejemplos de cómo utilizar estos comandos. Esperemos que Bernard Parisse halle un día el tiempo para hacer una documentación exhaustiva de la los comandos internos del CAS. Por el momento contamos con un una versión ligeramente editada de un documento que el proporcionó, que introduce algunos aspectos importantes del CAS.

43.2 Objetos Simbólicos

El CAS manipula escalares simbólicos y vectores o matrices de estos objetos. Escalares simbólicos tienen tres representaciones, que se muestran en la siguiente tabla con el objeto ‘2\*X’ como ejemplo:

|  |  |
| --- | --- |
| Representación | Escritura |
| User | Objeto simbólico único:  SYMBOL Z2 ID X x\* ; |
| Meta | El objeto simbólico descompuesto en la pila como un meta. Por ejemplo, '2\*X', se representa como cuatro objetos en los niveles 4 a 1 en la pila  Z2 ID X x\* BINT3 |
| Interna | Coeficientes del polinomio como una lista.  En el ejemplo, será:  { 2 0 }  con respecto a la lista de variables:  { X } |

La conversión de la representación user a la representación meta se hace con el comando ^SYMBINCOMP (un INNERCOMP generalizado que permite también manejar objetos no simbólicos como por ejemplo los números enteros).

La representación meta es utilizada para controlar las operaciones cuando la forma normal racional no es relevante. Es más eficiente que la representación simbólica porque no se tiene que descomponer y reconstruir a los objetos simbólicos a cada rato en las operaciones realizadas por las operaciones internas de un comando. Los comandos que manejan objetos meta son explicados detalladamente en el capítulo 12. Los comandos unarios y binarios fecuentemente tiene nombres cuyo prefijo es addt (por ejemplo addtSIN). Un ejemplo de una compleja rutina que maneja objetos meta es el comando CASCOMPEVAL, el cual hace un bucle similar a COMPEVAL pero con objetos meta en la pila en lugar de objetos simbólicos.

La representación interna se utiliza cuando la forma normal racional es importante. Este es el caso de la integración de fracciones racionales, simplificación racional, transformadas de Laplace, desarrollos en serie y operaciones similares. El primer paso para la conversión es encontrar la lista de variables con respecto a los cuales la expresión es racional. Por ejemplo:

es racional con respecto a { sin(x) cos(x) y }. Dado un objeto simbólico, una matriz simbólica o una lista con objetos simbólicos, el comando LVAR de User RPL o el comando LVARext de System RPL retorna la lista de variables. La conversión luego es hecha como un cociente de dos polinomios de varias variables con respecto a esta lista de variables, en ese orden.

La representación interna está dada por una lista, excepto para una constante, la cual puede no estar dentro de una lista.

Los enteros gaussianos son representados como un objeto programa con dos elementos:

:: parte\_imaginaria parte\_real ;

Tanto la parte real como la parte imaginaria deben de ser enteros.

Las raíces cuadradas son representadas como irrquads:

:: x<< a b c x>> ;

representa al simbólico: a+b\*sqrt(c)

Donde a, b y c deben de ser enteros.

Los polinomios son definidos como una lista de coeficientes. Cada coeficiente puede ser un polinomio, una constante (entero o entero gaussiano) o un irrquad. Las fracciones racionales construidas en base a estos polinomios son representadas como:

SYMBOL num deno x/

Donde num y deno son polinomios que son primos entre sí (en modo exacto).

El principal comando que convierte al formato interno es VXXLext. El principal comando que hace la operación inversa es R2SYM. Hay muchos otros comandos especializados para convertir entre las diferentes representaciones. Estas rutinas especializadas son más eficientes pero de más difícil uso.

Hay comandos que operan incluso sobre listas que representan a objetos racionales (QAdd, QSub, QDiv, QMul, QNeg, RPext), así como divisiones euclidianas con especializaciones, por ejemplo, para enteros o enteros gaussianos.

43.3 Ejemplos

La simplificación racional de un objeto simbólico puede ser codificado como:

:: ( symb )

FLASHPTR LVARext ( symb lvar ) ( Retorna lista de variables )

FLASHPTR VXXLext ( lvar n/d ) ( Convierte a representación interna )

FLASHPTR R2SYM ( symb ) ( Convierte a simbólico )

;

El producto escalar de dos vectores simbólico en "forma de lista":

:: ( VA VB )

INNERCOMP ( VA B1...Bn #n )

#1+ROLL ( B1...Bn VA )

INNERCOMP ( B1...Bn A1...An #n )

DUP#1= ( B1...Bn A1...An #n flag )

casedrop

FLASHPTR QMul ( Sale con: A1oB1 )

( B1...Bn A1...An #n )

get1 ( B1...Bn-1 A1...An #n Bn )

ROTSWAP ( B1...Bn-1 A1...An-1 #n An Bn )

FLASHPTR QMul ( B1...Bn-1 A1...An-1 #n AnoBn )

OVER ( B1...Bn-1 A1...An-1 #n AnoBn #n )

ONE\_DO (DO)

ROT ( B1...Bn-1 A1...An-2 #n DOT An-1 )

3PICK ( B1...Bn-1 A1...An-2 #n DOT An-1 #n )

#2+PICK ( B1...Bn-1 A1...An-2 #n DOT An-1 Bn-1 )

FLASHPTR QMul ( B1...Bn-1 A1...An-2 #n DOT An-1oBn-1 )

FLASHPTR QAdd ( B1...Bn-1 A1...An-2 #n DOT' )

LOOP

( B1...Bn-1 #n DOT )

OVER ( B1...Bn-1 #n DOT #n )

#1+UNROLL ( DOT B1...Bn-1 #n )

#1- ( DOT B1...Bn-1 #n-1 )

NDROP ( DOT )

;

Capítulo 44  
Verificar Tipos y Conversión

Los comandos de este capítulo son usados para verificar la presencia de los objetos especiales del CAS descritos en el capítulo anterior, y para la conversión entre estas diferentes clases de objetos.

44.1 Referencia

**Direcc. Nombre Descripción**

157006 ˆSYMBINCOMP ( symb 🡺 ob1 .. obN #n )

( ob 🡺 ob #1 )

( {} 🡺 {} #1 )

Descompone un objeto simbólico en un meta. Otros objetos son convertidos en metas de un objeto agregando #1 en la pila.

12A006 ˆ2SYMBINCOMP ( ob1 ob2 🡺 meta1 meta2 )

Hace ˆSYMBINCOMP para 2 objetos.

4D7006 ˆVXXLext ( ob Lvar 🡺 Lvar Q )

( ob nombre 🡺 Lvar Q )

Convierte ob a su representación interna.

ob puede ser de clase simbólica o matriz simbólica e incluso una lista con varios objetos.

Ejemplos:

‘X^2+5\*X+7’ { X } 🡺 { X } { Z1 Z5 Z7 }

‘13+14\*i’ { X } 🡺 { X } :: Z14 Z13 ;

‘7+5\*√11’ { X } 🡺 { X } :: Z7 Z5 Z11 ;

‘(3\*X+2)/(5\*X+7)’ { X } 🡺 { X } {Z3 Z2}/{Z5 Z7}

400006 ˆR2SYM ( lvar Q 🡺 ob )

Realiza la operación inversa al comando ˆVXXLext

Ejemplo:

{Z1 Z5 Z7} {X} 🡺 ‘X^2+5\*X+7’

4DB006 ˆVXXL0 ( ob 🡺 Q )

Conversion del objeto a su forma interna con respecto a Lvar (Lvar en LAM1).

4DD006 ˆVXXL2 ( Meta 🡺 Q )

Conversion del meta a su forma interna con respecto a Lvar (Lvar en LAM1).

4D8006 ˆMETALISTVXXL ( Meta 🡺 Meta )

Conversión de cada uno de los elementos del meta a su forma interna con respecto a Lvar

(Lvar en LAM1).

4DC006 ˆVXXL2NR ( Meta 🡺 Q )

Conversion del meta a su forma interna con respecto a Lvar

Este comando sólo llama a ˆVXXL2 pero antes activa temporalmente el flag 119 (Modo No riguroso) para permitir problemas con las raíces cuadradas

(Lvar en LAM1).

4DA006 ˆVXXL1ext ( n 🡺 Z )

Conversión de un objeto a su representación interna.

Para objetos que no dependen de ninguna variable.

**Direcc. Nombre Descripción**

4D9006 ˆVXXLFext ( n/d 🡺 Z1/Z2 )

Hace ˆVXXL1ext por separado al numerador y al denominador de una fracción.

167006 ˆTYPEIRRQ? ( ob 🡺 flag )

¿Es ob un irrquad?

Retorna TRUE si ob es un programa que tiene como primer elemento a x<<

168006 ˆDTYPEIRRQ? ( ob 🡺 ob flag )

Hace DUP, luego ˆTYPEIRRQ?

177006 ˆCKMATRIXELEM ( ob 🡺 ob )

Verifica que ob sea un elemento válido de una matriz interna. Es decir, verifica que sea de clase simbólica (id, lam ,symb o entero), real, complejo, real extendido, complejo extendido, unidad, lista o programa.

De lo contario, genera el error “Argumento incorrecto”

Para verificar que un objeto sea un elemento válido de una matriz simbólica en User RPL, debes usar otro comando, el comando ˆCKSYMREALCMP

18F006 ˆCKFPOLYext ( prog 🡺 ERROR )

( {} 🡺 {} )

( {} 🡺 ERROR )

( ob 🡺 ob )

Verifica que en la pila, no haya un programa. También verifica que en la pila no haya una lista que contenga algún programa o lista vacía (busca incluso dentro de listas interiores).

El error generado sería: “Argumento: valor incorr”

190006 ˆCK2FPOLY ( ob ob 🡺 ob ob )

Hace CKFPOLYext a dos objetos de la pila.

19E006 ˆCLEANIDLAM ( symb 🡺 id )

( symb 🡺 lam )

( symb 🡺 Z )

( symb 🡺 symb )

( ob 🡺 ob )

Si symb tiene un único objeto y ese objeto es un id, lam o entero, entonces retorna ese único elemento en la pila.

De lo contrario, no hace nada.

Capítulo 45  
Enteros

Este capítulo muestra los comandos que tratan con números enteros de precisión arbitraria, que son un nuevo tipo de objetos proporcionados por el CAS a partir de la HP 49g. Para una descripción breve de estos objetos, ve al capítulo 5.

Puedes observar que en este capítulo no hay comandos que realicen operaciones aritméticas básicas con enteros. Esto se debe a que un número entero también es una representación interna de objetos del CAS, por lo cual puedes usar los comandos que manipulan esta clase de objetos (tales como ˆQAdd, ˆQMul, etc). Estos comandos son listados en el capítulo 49 (polinomios).

45.1 Referencia

45.1.1 Enteros Ya Incorporados en ROM

**Direcc. Nombre Descripción**

2733F Z-9\_ ( 🡺 z-9 )

2734B Z-8\_ ( 🡺 z-8 )

27357 Z-7\_ ( 🡺 z-7 )

27363 Z-6\_ ( 🡺 z-6 )

2736F Z-5\_ ( 🡺 z-5 )

2737B Z-4\_ ( 🡺 z-4 )

27387 Z-3\_ ( 🡺 z-3 )

27393 Z-2\_ ( 🡺 z-2 )

2739F Z-1\_ ( 🡺 z-1 )

273AB Z0\_ ( 🡺 z0 )

273B6 Z1\_ ( 🡺 z1 )

273C2 Z2\_ ( 🡺 z2 )

273CE Z3\_ ( 🡺 z3 )

273DA Z4\_ ( 🡺 z4 )

273E6 Z5\_ ( 🡺 z5 )

273F2 Z6\_ ( 🡺 z6 )

273FE Z7\_ ( 🡺 z7 )

2740A Z8\_ ( 🡺 z8 )

27416 Z9\_ ( 🡺 z9 )

27422 Z10\_ ( 🡺 z10 )

2742F Z12\_ ( 🡺 z12 )

2743C Z24\_ ( 🡺 z24 )

27449 Z100\_ ( 🡺 z100 )

27516 Z0Z1\_ ( 🡺 z0 Z1 )

274A9 Z1Z0\_ ( 🡺 z1 Z0 )

aka: ZINT1\_0\_

2756C Z1Z1\_ ( 🡺 z1 Z1 )

2754B Z-1Z0\_ ( 🡺 z-1 Z0 )

27C70 Z0ONE\_ ( 🡺 z0 #1 )

45.1.2 Enteros Ya Incorporados con Manipulación de Pila

**Direcc. Nombre Descripción**

2E0006 ˆDROPZ0 ( ob 🡺 z0 )

2DF006 ˆDROPZ1 ( ob 🡺 z1 )

392006 ˆ2DROPZ0 ( 2 1 🡺 z0 )

3B3006 ˆNDROPZ0 ( obn...ob1 #n 🡺 z0 )

Reemplaza a un meta con Z0

3B4006 ˆNDROPZ1 ( obn...ob1 #n 🡺 z1 )

Reemplaza a un meta con Z1

45.1.3 Conversión

**Direcc. Nombre Descripción**

0EE006 ˆ#>Z ( # 🡺 Z )

Convierte un bint a entero zint.

0F5006 ˆR>Z ( % 🡺 Z )

Convierte un real a entero zint.

No usar este comando si el número real tiene parte decimal.

18D006 ˆR2Zext ( % 🡺 Z )

( % 🡺 %% )

Si el número real no tiene parte decimal, lo convierte a entero.

Si el número real tiene parte decimal y la calculadora está en modo aproximado (flag 105 activado), entonces lo convierte a real extendido y activa el flag 102 (no GCD).

Si el número real tiene parte decimal y la calculadora está en modo exacto (flag 105 desactivado), entonces

a) Si el flag 123 está activado (modo switch prohibido), genera el error "Mode switch not allowed here"

b) Si el flag 123 está desactivado, (modo switch permitido), entonces:

• Si el flag 120 está activado (modo silencioso encendido), activa los flags 105 y 102 sin preguntar y convierte el real a real extendido.

• Si el flag 120 está desactivado (modo silencioso apagado), pregunta al usuario si desea activar el modo aproximado (activar flag 105).

Si el usuario no quiere encenderlo, entonces genera el error “Mode switch cancelled”. Si el usuario acepta, activa los flags 105 y 102 y convierte el real a real extendido.

0ED006 ˆH>Z ( HXS 🡺 Z // Error )

Convierte un HXS a entero, si el hxs en base 16 no tiene como dígitos a letras.

En otro caso, genera el error “Argumento incorrecto”

Por ejemplo:

# 123456h 🡺 123456

# 123C56h 🡺 ERROR

0F2006 ˆS>Z ( $ 🡺 Z )

Convierte una cadena a entero. La cadena no debe tener punto decimal. Por ejemplo:

“-12345” 🡺 -12345

**Direcc. Nombre Descripción**

0F3006 ˆS>Z? ( $ 🡺 Z T )

( $ 🡺 $ F )

Si es posible, convierte cadena a entero y retorna TRUE

De lo contrario, agrega FALSE

Por ejemplo:

“-78” 🡺 -78 TRUE

“123.” 🡺 “123.” FALSE

184006 ˆCK1Z ( Z 🡺 Z )

( % 🡺 Z )

( $ 🡺 Z )

Si hay un entero en la pila, no hace nada.

Convierte un real a entero con ˆR>Z

Convierte una cadena a entero con ˆS>Z

Para otros objetos genera el error “Argumento incorrecto”

185006 ˆCK2Z ( ob ob' 🡺 Z Z' )

Hace ˆCK1Z, pero para dos objetos.

186006 ˆCK3Z ( ob ob' ob'' 🡺 Z Z' Z'' )

Hace ˆCK1Z, pero para tres objetos.

202006 ˆCK&CONVINT ( Z 🡺 Z )

( symb 🡺 :: Zim Zre' ; )

Convierte un zint a entero gausiano.

Por ejemplo:

'2+3\*i' 🡺 :: Z3 Z2 ;

Para otros objetos, genera el error “Argumento incorrecto”.

203006 ˆCK&CONV2INT ( ob ob' 🡺 ob'' ob''' )

Hace ˆCK1Z, pero para dos objetos.

205006 ˆCONVBACKINT ( Z 🡺 Z )

( C 🡺 symb )

( irrquad 🡺 symb )

Hace:

:: NULL{} SWAP FLASHPTR R2SYM ;

Se puede usar para convertir un entero gaussiano o un irrquad a su representación user.

Ejemplos:

:: Z3 Z2 ; 🡪 '2+3\*i'

:: << Z5 Z6 Z7 >> ; 🡪 '5+6\*√7'

204006 ˆCONVBACK2INT ( ob ob' 🡺 ob'' ob''' )

Hace ˆCONVBACKINT, pero para dos objetos.

0F4006 ˆZ>ZH ( Z 🡺 Z' )

Convierte un entero de base decimal a base hexadecimal.

18E006 ˆZ2Sext ( Z 🡺 '$Z' )

Convierte un entero a cadena. Luego, esta cadena es incrustada en un objeto simbólico.

Ejemplo:

Z3 🡺 '"3"'

45.1.4 Operaciones Generales con Enteros

**Direcc. Nombre Descripción**

101006 ˆZTrim ( Z 🡺 Z' )

Le quita a Z, nibbles sobrantes innecesarios.

Cuenta los nibbles requeridos para la representación de Z

Si estos son iguales a los usados, sale rápidamente.

De lo contrario, asigna a un nuevo objeto, copia los nibbles significativos de la mantisa y agrega el signo original.

Por ejemplo:

00085 🡺 85

102006 ˆZabs ( Z 🡺 |Z| )

Pone el valor absoluto de Z en la pila.

Si Z es positivo, no hace nada.

De lo contrario, cambia de signo a una copia del objeto.

50B006 ˆZABS ( Z 🡺 |Z| )

Pone el valor absoluto de Z en la pila.

Hace

:: FLASHPTR DupZIsNeg? NOT?SEMI FLASHPTR QNeg ;

0E0006 ˆZSQRT ( Z 🡺 Z' flag )

Calcula la parte entera de la raíz cuadrada de un entero POSITIVO o CERO.

Si es cuadrado perfecto, entonces retorna TRUE para indicar que el entero Z tiene raíz exacta.

Por ejemplo:

144 🡺 12 TRUE

145 🡺 12 FALSE

3D0006 ˆMod ( Z Zn 🡺 Z' )

Retorna el residuo de dividir Z entre Zn

Por ejemplo:

45 7 🡺 3

80 16 🡺 0

-75 -7 🡺 2

15 0 🡺 0

0DD006 ˆZmod ( Z1 Z2 🡺 Z' )

Retorna el residuo de dividir Z entre Zn

Parecido al comando MOD de User RPL.

Por ejemplo:

45 7 🡺 3

80 16 🡺 0

-75 -7 🡺 -5

15 0 🡺 15

105006 ˆZNMax ( Z1 Z2 🡺 Z )

Retorna el entero que tenga el mayor valor absoluto.

Por ejemplo:

45 7 🡺 45

-8 -6 🡺 -8

106006 ˆZNMin ( Z1 Z2 🡺 Z )

Retorna el entero que tenga el menor valor absoluto.

Por ejemplo:

45 7 🡺 7

-8 -6 🡺 -6

**Direcc. Nombre Descripción**

10D006 ˆZBits ( Z 🡺 Z #bits )

Retorna el número de bits usados en Z

10E006 ˆZBit? ( Z #bit 🡺 Z flag )

Comprueba si un bit está activado en Z

El contador empieza desde cero, a diferencia de ^ZBits

2B7006 ˆZGCDext ( Z Z' 🡺 Z'' )

Máximo común divisor de dos enteros.

2B8006 ˆZGcd ( Z Z' 🡺 Z'' )

Máximo común divisor de dos enteros.

Hace lo mismo que ZGCDext

0DE006 ˆZDIVext ( Z1 Z2 🡺 Zcoc Zdiv )

Retorna el cociente y el divisor de dividir Z1/Z2

20A006 ˆIEGCD ( x y 🡺 c a b )

Dados dos enteros x y y, retorna 3 enteros a, b, c de tal forma que ax+by=c.

Aquí c es el máximo común divisor de x e y

x, y son enteros o reales sin parte decimal.

Equivale al comando IEGCD de user RPL.

3D6006 ˆIEGCDext ( x y 🡺 c a b )

Llamado por el comando ^IEGCD

07C007 ˆ#FACT ( # 🡺 Z )

Calcula el factorial de un entero.

Funciona para los números #0 - #FFFFF, aunque desde algún número se generará el error: “Memoria insuficiente”.

576006 ˆfactzint ( Z 🡺 Z! )

Factorial de un entero de 0 a 9999.

215006 ˆPA2B2 ( p 🡺 a+bi )

El argumento puede ser 1, 2, o primo múltiplo de 4 más 1.

Retorna un complejo simbólico de manera que p=a²+b².

El argumento debe ser entero o real sin parte decimal.

Equivale al comando PA2B2 de user RPL.

45.1.5 Factorización de Enteros y Números Primos

**Direcc. Nombre Descripción**

0C9006 ˆZFactor ( Zs 🡺 {} )

Factoriza un entero largo.

Retorna una lista con los factores.

0CA006 ˆNFactor ( z 🡺 {} )

Factoriza un entero positivo largo.

0CB006 ˆNFactorSpc ( z 🡺 {} )

Semi-factoriza un entero largo positivo. Esta es la factorización regular con una prueba extra “sin esperanza”.

0CD006 ˆSFactor ( S 🡺 Lf )

Factors short integer. Pollard Rho, with the assumption that trial division has been done already. Thus any factor less than 4012009 is known to be a prime, for greater factors a primality test is used before calling the actual Pollard Rho. Pollard Rho does not find the factors in order of magnitude, thus the results will be sorted after full factorization has been achieved.

0CE006 ˆSPollard ( S 🡺 S1 S2 )

Factors short integer into 2 parts using Pollard Rho algorithm. Trial division and primality tests should be done prior to calling this subroutine, otherwise an eternal loop is risked. The random number generator is modeled after the user level RAND command, although the starting value is different.

0CF006 ˆBFactor ( N 🡺 Lf )

Factors long integer. Brent-Pollard, with the assumption that trial division has been done already. When a small factor is found SFactor is called to get full short factorization.

Since the factorization can potentially take a very long time, an execution time test is used to abort factoring very long integers (limit is 60s for each composite). The factors are sorted at exit.

0D0006 ˆBrentPow ( Za Z1 Z2 Zn #k 🡺 Z )

Modular \* + ˆ mod for Brent-Pollard factorization.

Output is Z1\*Z2+Za mod Zn repeated k times Note that k=0 and k=1 give the same result. Also Z16=Z2 makes no sense for k6=0.

All arguments are assumed to be positive. Za is assumed to be < 16. In some instances k can be a very high number, thus it might make sense to use Montgomery multiplication.

218006 ˆISPRIME ( Z 🡺 %1/%0 )

( % 🡺 %1/%0 )

Retorna %1 si el número es primo.

Equivale al comando ISPRIME? de User RPL.

0D1006 ˆZPrime? ( Z 🡺 flag )

Retorna TRUE si el entero es primo o uno.

0C7006 ˆPrime+ ( Z 🡺 Z' )

Retorna el siguiente primo ( Z' > Z ).

Equivale al comando PREVPRIME de User RPL.

0C8006 ˆPrime- ( Z 🡺 Z' )

Retorna el anterior primo ( Z' < Z ).

Equivale al comando NEXTPRIME de User RPL.

45.1.6 Enteros Gausianos

**Direcc. Nombre Descripción**

114007 ˆTYPEGAUSSINT? ( ob 🡺 flag )

Retorna TRUE si el objeto es un entero gausiano.

Cuidado: este comando sólo verifica que en la pila se encuentre un objeto programa no vacío cuyo primer elemento sea un entero.

Para verificar si en la pila se encuentra un objeto programa con exactamente 2 enteros, puedes usar:

NULLNAME TYPEGAUSS2INT? ( ob -> flag )

:: ( ob )

DUPTYPECOL? ( ob flag )

NOTcase

DROPFALSE

( ob )

INNERDUP ( ob1...obn #n #n )

#2= ( ob1...obn #n flag )

NOTcase

NDROPFALSE

( ob1 ob2 #2 )

DROP ( ob1 ob2 )

TYPEZINT? ( ob1 flag )

SWAP ( flag ob1 )

TYPEZINT? ( flag flag' )

AND ( flag'' )

;

115007 ˆDTYPEGAUSSINT? ( ob 🡺 ob flag )

Equivale a:

DUP FLASHPTR TYPEGAUSSINT?

116007 ˆDUPTYPEGAUSSINT? ( ob 🡺 ob flag )

Equivale a:

DUP FLASHPTR TYPEGAUSSINT?

187006 ˆCK1Cext ( Z 🡺 Z )

( C 🡺 C )

Verifica que en la pila se encuentre un entero o un entero gausiano. Si no es así, genera el error “Argumento incorrecto”.

Este comando verifica que en la pila se encuentre un entero o un objeto programa con dos elementos.

15D006 ˆCXRIext ( C 🡺 Zre Zim )

( Z 🡺 Z Z0 )

Retorna la parte real y la imaginaria de un entero gausiano.

519006 ˆCXIRext ( C 🡺 Zim Zre )

( Z 🡺 Z0 Z )

Retorna la parte imaginaria y la real de un entero gausiano.

4D6006 ˆSUMSQRext ( Z 🡺 Z C )

Retorna un entero gaussiano C que cumpla: |C|ˆ2=Z.

Z puede ser:

- El entero 2.

- Un entero múltiplo de 4 más 1 y primo a la vez.

Si no se cumplen las condiciones, puede generar los errores: “Negative integer”, “Z is not = 1 mod 4” o “Z is not prime”.

518006 ˆCNORMext ( C 🡺 |C|ˆ2 )

Retorna el modulo al cuadrado de un entero gausiano.

45.1.7 Tests con Enteros

**Direcc. Nombre Descripción**

265C1 Z= ( Z Z' 🡺 flag )

265C6 Z<> ( Z Z' 🡺 flag )

265BC Z< ( Z Z' 🡺 flag )

265D0 Z<= ( Z Z' 🡺 flag )

265B7 Z> ( Z Z' 🡺 flag )

265CB Z>= ( Z Z' 🡺 flag )

0F8006 ˆQIsZero? ( Z 🡺 flag )

( {} 🡺 T )

Retorna TRUE, si Z es cero.

0F7006 ˆDupQIsZero? ( Q 🡺 Q flag )

Hace DUP, luego ˆQIsZero?

0FA006 ˆZIsOne? ( Z 🡺 flag )

Retorna TRUE si Z es 1.

0F9006 ˆDupZIsOne? ( Z 🡺 Z flag )

Agrega TRUE si Z es 1.

109006 ˆDupZIsTwo? ( Z 🡺 Z flag )

Agrega TRUE si Z es 2.

0FC006 ˆZIsNeg? ( Z 🡺 flag )

Retorna TRUE si Z es un entero negativo.

0FB006 ˆDupZIsNeg? ( Z 🡺 Z flag )

Agrega TRUE si Z es un entero negativo.

10A006 ˆDupZIsEven? ( Z 🡺 Z flag )

Agrega TRUE si Z es par.

107006 ˆZNLT? ( Z1 Z2 🡺 flag )

Retorna TRUE si |Z1|<|Z2|.

19A006 ˆOBJINT? ( Z 🡺 Z T )

( # 🡺 Z T )

( % 🡺 Z T )

( % 🡺 % F )

( symb 🡺 Z T )

( symb 🡺 id/lam/symb F )

( ob 🡺 ob F )

¿Puede el objeto representar a un entero?

Para un entero, agrega TRUE

Para un bint, lo convierte a entero y agrega TRUE

Para un real sin parte decimal, lo convierte a entero y agrega TRUE

Para un real con parte decimal, agrega FALSE

Para un simbólico de un solo objeto (un entero), retorna ese entero y TRUE

Para un simbólico de un solo objeto (id o lam), retorna ese nombre y FALSE

Para otros simbólicos u otros objetos agrega FALSE

19C007 FLASHPTR 007 19C ( ob ob' 🡺 Z Z' )

Si los dos objetos pueden representar a enteros, los coloca en la pila como enteros (usa ˆOBJINT? para ambos).

De lo contrario, genera el error “Argumento incorrecto”.

**Direcc. Nombre Descripción**

19B006 ˆOBJPOSINT? ( Z 🡺 Z flag )

( # 🡺 Z flag )

( % 🡺 Z flag )

( % 🡺 % F )

( symb 🡺 Z flag )

( symb 🡺 id/lam/symb F )

( ob 🡺 ob F )

¿Puede el objeto representar a un entero mayor o igual a cero y menor o igual a diez?

La conversión de un real o de un simbólico es similar a la que hace el comando ˆOBJINT?

19C006 ˆCKINT>0 ( Z/#/%/symb 🡺 # TRUE )

( Z/#/%/symb 🡺 #0 FALSE )

( Z/#/%/symb 🡺 ERROR )

( ob 🡺 ERROR )

Si el objeto representa un entero entre 1 y 9999 inclusive, retorna un bint y TRUE

Si el objeto representa a cero, retorna el bint 0 y FALSE

De otro modo, genera un error.

Hace:

:: FLASHPTR OBJINT? NcaseTYPEERR FLASHPTR Z># DUP#0<> ;

198006 ˆMETAINT? ( Meta 🡺 Meta flag )

Agrega TRUE si en la pila hay un meta de 1 objeto y ese objeto es un entero.

Si el meta es de 1 objeto que puede convertirse en entero (con el comando ˆOBJINT?) entonces es convertido y se agrega TRUE

De lo contrario, se agrega FALSE

199006 ˆMETAPOSINT? ( Meta 🡺 Meta flag )

Agrega TRUE si en la pila hay un meta de 1 objeto y ese objeto es un entero entre cero y diez inclusive.

Si el meta es de 1 objeto que puede convertirse en entero (con el comando ˆOBJPOSINT?) de 0 a 10, entonces es convertido y se agrega TRUE

De lo contrario, se agrega FALSE

0CC006 ˆDupTypeS? ( Z 🡺 Z flag )

Agrega TRUE si |Z|<1x1015

Capítulo 46  
Matrices

Las matrices simbólicas son un nuevo objeto presente a partir de la HP 49G. A diferencia de los arreglos, las matrices simbólicas pueden contener objetos de diferentes tipos en su interior, incluso objetos simbólicos.

Las matrices también son objetos compuestos y podemos usar muchos de los comandos descritos en el capítulo 11 también para manipular matrices. La razón es obvia, la estructura de las matrices simbólicas es la misma que la de una lista de listas. Sólo cambia el prólogo.

En la calculadora, puedes crear una matriz simbólica con los delimitadores

MATRIX y ;

Por ejemplo:

A) Para crear una matriz de 1 dimensión con los enteros 7,8 y 9.

MATRIX Z7\_ Z8\_ Z9\_ ;

B) Para crear una matriz de 2 dimensiones de dos filas por tres columnas.

MATRIX

MATRIX Z4\_ Z5\_ Z6\_ ;

MATRIX Z7\_ Z8\_ Z9\_ ;

;

En el editor de Debug 4x es diferente. Para crear una matriz puedes hacer uso de los comandos TYPEMATRIX\_ y COMPN\_

**Direcc. Nombre Descripción**

03FF9 (TYPEMATRIX) ( 🡺 # )

Retorna el bint 9862d (2686h).

05331 (COMPN) ( ob1..obn #n #prólogo 🡺 comp )

Crea un objeto compuesto.

Por ejemplo:

:: 1. 2. 3. BINT3 # 2686 COMPN ;

crea una matriz simbólica de una dimensión.

Por ejemplo:

A) Para crear la matriz [ 7 8 9 ], escribe:

Z7\_ Z8\_ Z9\_ BINT3 TYPEMATRIX\_ COMPN\_

B) Para crear la matriz [ [ 4 5 6 ] [ 7 8 9 ] ], escribe:

Z4\_ Z5\_ Z6\_ BINT3 TYPEMATRIX\_ COMPN\_

Z7\_ Z8\_ Z9\_ BINT3 TYPEMATRIX\_ COMPN\_

BINT2 TYPEMATRIX\_ COMPN\_

También puedes usar los comandos ˆXEQ>ARRY y ˆLIST2MATRIX para crear una matriz simbólica.

**Direcc. Nombre Descripción**

17F006 ˆXEQ>ARRY ( %...%' {%el}/{%f,%c} 🡺 RealARRY )

( C%...C%' {%el}/{%f,%c} 🡺 CmpARRY )

( ob1...obn {%el}/{%f,%c} 🡺 MATRIX )

Crea una formación a partir de los elementos de la pila.

Si todos son reales, crea un arreglo real.

Si todos son complejos, crea un arreglo complejo.

De no cumplirse esas condiciones, crea una matriz simbólica.

Pero los objetos sólo puden ser reales, complejos y objetos de clase simbólica (enteros, symb, id o lam).

Usado por el comando **→ARRY** de User RPL.

17A006 ˆLIST2MATRIX ( {} 🡺 1DMATRIX )

( {{}} 🡺 2DMATRIX )

( {MATRIX} 🡺 2DMATRIX )

( ob 🡺 ob )

Convierte una lista a matriz simbólica.

Cada objeto de la matriz puede ser real, complejo, clase simbólica (id, lam, symb o entero), unidad o arreglo.

Si en la pila hay una lista vacía o una lista que contiene una lista vacía, genera el error “Dimensión inválida”.

Ninguno de los objetos de la matriz resultante debe ser lista o matriz simbólica (llama al comando ˆDIMS).

ob puede ser real, complejo, clase simbólica (id, lam, symb o entero), unidad, arreglo o matriz simbólica.

Por ejemplo:

A) Para crear la matriz [ 7 8 9 ], puedes hacer de una de estas formas:

Z7\_ Z8\_ Z9\_

{ %3 }

FLASHPTR XEQ>ARRY

{ Z7\_ Z8\_ Z9\_ }

FLASHPTR LIST2MATRIX

B) Para crear la matriz [ [ 4 5 6 ] [ 7 8 9 ] ], puedes hacer de una de estas formas:

Z4\_ Z5\_ Z6\_

Z7\_ Z8\_ Z9\_

{ %2 %3 }

FLASHPTR XEQ>ARRY

{ { Z4\_ Z5\_ Z6\_ }

{ Z7\_ Z8\_ Z9\_ } }

FLASHPTR LIST2MATRIX

Sin embargo, si deseas crear siempre una matriz de 2 dimensiones a partir de los elementos de la pila, la manera más rápida es usando el siguiente NULLNAME:

\* Crea una matriz simbólica de 2 dimensiones a partir de objetos

\* que estén en la pila.

\* En el nivel 2 debe estar el número de filas.

\* En el nivel 1 debe estar el número de columnas.

NULLNAME Crea\_2DMATRIX ( ob1...obk #f #c -> 2DMATRIX )

:: ( ob1...obk #f #c )

ZERO ( ob1...obk #f #c #0 )

ROT ( ob1...obk #c #0 #f )

ZERO\_DO (DO)

( ... #c MetaMatrix )

psh ( ... MetaMatrix ob1...obc #c )

DUP ( ... MetaMatrix ob1...obc #c #c )

OBJ>R\_ ( ... MetaMatrix ob1...obc #c )

TYPEMATRIX\_ ( ... MetaMatrix ob1...obc #c 9862 )

COMPN\_ ( ... MetaMatrix 1DMATRIX )

SWP1+ ( ... MetaMatrix' )

R>OBJ\_ ( ... MetaMatrix' #c )

OVER#2+UNROL

LOOP

( #c MetaMatrix )

reversym ( #c MetaMatrix' )

TYPEMATRIX\_ ( #c MetaMatrix' 2686 )

COMPN\_ ( #c 2DMATRIX )

SWAPDROP ( 2DMATRIX )

;

El siguiente NULLNAME crea una matriz de 1 dimensión:

\* Crea una matriz simbólica de 1 dimensión a partir de objetos

\* que estén en la pila.

\* En el nivel 1 debe estar el número de elementos.

NULLNAME Crea\_1DMATRIX ( ob1...obn #n -> 1DMATRIX ) ( M1...Mn #n -> 2DMATRIX )

:: ( ob1...obn #n )

TYPEMATRIX\_ ( ob1...obn #n 9862 )

COMPN\_ ( MATRIX )

;

Para conseguir un objeto desde una posición especificada, puedes usar el comando ˆPULLEL[S]

**Direcc. Nombre Descripción**

35B006 ˆPULLEL[S] ( RealARRY #ubic 🡺 RealARRY % )

( CmpARRY #ubic 🡺 CmpARRY C% )

( MATRIX #ubic 🡺 MATRIX ob )

Retorna el elemento de la posición especificada.

431001 FLASHPTR 001 431 ( RealArry %ubic 🡺 % )

( CmpArry %ubic 🡺 C% )

( MATRIX %ubic 🡺 ob )

( [[%]] {%i %j} 🡺 % )

( [[C%]] {%i %j} 🡺 C% )

( 2DMATRIX {%i %j} 🡺 ob )

Retorna el elemento de la posición especificada.

Equvale al comando GET de User RPL cuando en la pila hay una formación y una lista o número real con dos reales.

Sin embargo, a veces queremos obtener un objeto desde una matriz, teniendo la posición de fila y columna del elemento deseado como bints.

Para ello puedes usar alguno de estos NULLNAME. Son más rápidos que los 2 comandos descritos arriba.

\* Retorna un elemento de una matriz de 2 dimensiones ubicado

\* en la fila 'i' y en la columna 'j'. También pone TRUE.

\* Si el elemento no existe, sólo pone FALSE.

NULLNAME GetFrom2DMATRIX ( 2DMATRIX #i #j -> ob T // F )

:: ( 2DMATRIX #i #j )

UNROT ( #j 2DMATRIX #i )

NTHELCOMP ( #j 1DMATRIX T // #j F )

NOTcase

DROPFALSE

( #j 1DMATRIX )

SWAP ( 1DMATRIX #j )

NTHELCOMP ( ob T // F )

;

\* Retorna un elemento de una matriz de 2 dimensiones ubicado

\* en la fila 'i' y en la columna 'j'

NULLNAME GetFrom2DMATRIX ( 2DMATRIX #i #j -> ob )

:: ( 2DMATRIX #i #j )

UNROT ( #j 2DMATRIX #i )

NTHCOMPDROP ( #j 1DMATRIX )

SWAP ( 1DMATRIX #j )

NTHCOMPDROP ( ob )

;

Los siguientes NULLNAME también funcionan para arreglos reales o complejos de dos dimensiones.

\* Retorna un elemento de una formación de 2 dimensiones ubicado

\* en la fila 'i' y en la columna 'j'. También pone TRUE.

\* Si el elemento no existe, sólo pone FALSE.

NULLNAME GetFrom2DMATRIX ( [[]]/2DMAT #i #j -> ob T // F )

:: ( [[]]/2DMAT #i #j )

TWO{}N ( [[]]/2DMAT {#i #j} )

OVER ( [[]]/2DMAT {#i #j} [[]]/2DMAT )

FLASHPTR FINDELN ( [[]]/2DMAT #ubic T // [[]]/2DMAT F )

NOTcase

DROPFALSE ( SALE CON: FALSE )

( [[]]/2DMAT #ubic )

FLASHPTR PULLEL[S] ( [[]]/2DMAT ob )

SWAPDROPTRUE ( ob T )

;

\* Retorna un elemento de una formación de 2 dimensiones ubicado

\* en la fila 'i' y en la columna 'j'.

NULLNAME GetFrom2DMATRIX ( [[]]/2DMAT #i #j -> ob )

:: ( [[]]/2DMAT #i #j )

TWO{}N ( [[]]/2DMAT {#i #j} )

OVER ( [[]]/2DMAT {#i #j} [[]]/2DMAT )

FLASHPTR FINDELN ( [[]]/2DMAT #ubic T )

DROP ( [[]]/2DMAT #ubic )

FLASHPTR PULLEL[S] ( [[]]/2DMAT ob )

SWAPDROP ( ob T )

;

46.1 Referencia

46.1.1 Creando una Matriz Identidad

**Direcc. Nombre Descripción**

12D007 FLASHPTR 007 12D ( # 🡺 2DMATRIX )

Crea una matriz identidad. Será una matriz simbólica con enteros. El bint no debe ser cero.

436001 FLASHPTR 001 436 ( % 🡺 [[%]]' )

Crea un arreglo identidad. El argumento es el número de filas.

Si es cero o negativo, genera el error: “Argumento: valor incorr”

Equivale al comando IDN de User RPL cuando en la pila hay un número real.

36E001 FLASHPTR 001 36E ( [[%]] 🡺 [[%]]' )

( [[C%]] 🡺 [[C%]]' )

Crea un arreglo identidad.

Verifica que el arreglo sea de 2 dimensiones y cuadrada.

Si el arreglo no cumple esto, genera el error “Dimensión inválida”.

367001 FLASHPTR 001 367 ( [[%]] 🡺 [[%]]' )

( [[C%]] 🡺 [[C%]]' )

Crea un arreglo identidad.

En la pila debe haber un arreglo de 2 dimensiones y cuadrada.

(comando tipo bang)

371006 ˆMATIDN ( 2DMATRIX 🡺 [[%]]/2DMATRIX )

( [[]] 🡺 [[%]]/2DMATRIX )

( % 🡺 [[%]]/2DMATRIX )

( Z 🡺 [[%]]/2DMATRIX )

( # 🡺 [[%]]/2DMATRIX )

Crea una formación identidad.

Si la calculadora está en modo aproximado (flag 105 activado), entonces crea un arreglo real. Si está en modo exacto (flag 105 desactivado), entonces crea una matriz simbólica con enteros.

La matriz o arreglo debe ser de dos dimensiones y el número de filas debe ser igual al número de columnas.

El número real debe ser mayor o igual a 1. Si es menor o igual a 1, se cambia de signo. El número entero debe estar entre 1 y 143 inclusive. También puede estar entre -1 y -143.

Equivale al comando IDN de User RPL cuando en la pila hay una formación o un entero.

438001 FLASHPTR 001 438 ( lam 🡺 )

El lam debe contener a una formación de dos dimensiones y cuadrada. Se guardará en el lam una formación identidad del mismo tamaño que la existente. Será arreglo o matriz, dependiendo del estado del flag 105.

El comando hace todas las verificaciones necesarias, generando errores si es necesario: “Nombre no definido” “Argumento incorrecto”, “Dimensión inválida”.

Equivale al comando IDN de User RPL cuando en la pila hay un nombre local.

**Direcc. Nombre Descripción**

437001 FLASHPTR 001 437 ( id 🡺 )

El id debe contener a una formación de dos dimensiones y cuadrada, o un real o un entero.

Se guardará en el id una formación identidad.

Si contiene un real, se guardará un arreglo real.

Si contiene a una formación o un entero, se guardará un arreglo o matriz, dependiendo del estado del flag 105.

El comando hace todas las verificaciones necesarias, generando errores si es necesario: “Nombre no definido” “Argumento incorrecto”, “Dimensión inválida”.

Equivale al comando IDN de User RPL cuando en la pila hay un nombre global.

46.1.2 Creando una Matriz Constante

**Direcc. Nombre Descripción**

374006 ˆOBJDIMS2MAT ( ob {#elem} 🡺 1DMATRIX )

( ob {#f,#c} 🡺 2DMATRIX )

Crea una matriz simbólica que tendrá todos sus elementos iguales a ob

03442 MAKEARRY\_ ( {#L1,#L2,…#Ln} ob 🡺 ARRY )

Crea un arreglo con todos sus elementos iguales a ob.

Las dimensiones del arreglo serán las que indica la lista.

373006 ^MAKEARRY ( {#L1,#L2,…#Ln} % 🡺 RealARRY )

( {#L1,#L2,…#Ln} C% 🡺 CmpARRY )

( {#L1,#L2,…#Ln} %% 🡺 ExtRealARRY )

( {#L1,#L2,…#Ln} C%% 🡺 ExtCmpARRY )

( {#elem}/{#f,#c} Z 🡺 MATRIX )

( {#elem}/{#f,#c} symb 🡺 MATRIX )

( {#elem}/{#f,#c} id 🡺 MATRIX )

( {#elem}/{#f,#c} lam 🡺 MATRIX )

( {#elem}/{#f,#c} {} 🡺 MATRIX )

Parecido al comando MAKEARRY\_, pero solo funciona para crear formaciones con objetos de los tipos mostrados, pudiendo crear también matrices simbólicas.

Si en la pila hay un lam, crea una matriz cuyos elementos son iguales al contenido del lam; si el lam no existe, manda un mensaje de error.

372006 ˆMATCON ( RealArry % 🡺 RealArry' )

( MATRIX % 🡺 RealArry' )

( CmpArry C% 🡺 CmpArry' )

( CmpArry % 🡺 CmpArry' )

( MATRIX C% 🡺 CmpArry' )

( RealArry/MATRIX ob 🡺 MATRIX )

( CmpArry/MATRIX ob 🡺 MATRIX )

Crea una formación constante de las mismas dimensiones que la formación original.

ob es un objeto de clase simbólica (id, lam, symb o entero)

Usado por el comando CON de User RPL.

46.1.3 Creando Matrices con Elementos Aleatorios

**Direcc. Nombre Descripción**

345006 ˆDIMRANM ( {#el} 🡺 1DMATRIX )

( {#f #c} 🡺 2DMATRIX )

Crea un matriz simbólica aleatoria cuyas dimensiones están indicadas en la lista. Sus elementos serán números enteros (flag -105 desactivado) o reales (flag -105 activado) comprendidos entre -9 y 9 inclusive.

Usado por el comando RANM de User RPL.

344006 ˆMATRANM ( RealArry 🡺 RealArry' )

( CmpArry 🡺 CmpArry' )

( MATRIX 🡺 RealArry' )

( MATRIX 🡺 CmpArry' )

( MATRIX 🡺 MATRIX' )

Crea una formación de las mismas dimensiones que la original.

Si una matriz simbólica contiene sólo números reales, devuelve un arreglo real.

Si una matriz simbólica contiene sólo números complejos, devuelve un arreglo complejo.

Si la matriz simbólica, no tiene todos sus elementos reales o todos complejos, devuelve una matriz con enteros o reales (según estado del flag -105).

Los elementos son números aleatorios del conjunto

{-9.,-8.,-7.,……,7.,8.,9.} o números complejos aleatorios cuyas partes real e imaginaria pertenecen a ese conjunto.

Equivale al comando RANM de User RPL cuando en la pila hay una formación.

053003 FLASHPTR 003 053 ( RealArry 🡺 RealArry' )

( CmpArry 🡺 CmpArry' )

Similar a ˆMATRANM, pero funciona sólo para arreglos.

Crea un arreglo de las mismas dimensiones que el original.

Los elementos son números reales aleatorios que pertenecen al conjunto {-9.,-8.,-7.,……,7.,8.,9.} o números complejos aleatorios cuyas partes real e imaginaria pertenecen a ese conjunto.

46.1.4 Creando Matrices con Elementos que estén en Función de su Fila y de su Columna

**Direcc. Nombre Descripción**

375006 ˆLCPROG2M ( #f #c prog 🡺 2DMATRIX )

( #f #c symb 🡺 2DMATRIX )

Crea una matriz simbólica de dos dimensiones. prog es un programa que toma 2 argumentos enteros (el número de fila y de columna del elemento) y retorna sólo un objeto.

symb es un simbólico cualquiera. En este simbólico las variables globales ID I e ID J serán reemplazadas por los enteros que representen al número de fila y al número de columna del elemento.

Si prog no retorna sólo un objeto, se genera el error “Muy pocos argumentos”

Equivale al comando LCXM de User RPL.

Llama al comando ˆMAKE2DMATRIX

**Direcc. Nombre Descripción**

376006 ˆMAKE2DMATRIX ( #f #c prog 🡺 2DMATRIX )

Crea una matriz simbólica de dos dimensiones.

prog es un programa que toma 2 argumentos enteros (el número de fila y de columna del elemento) y retorna sólo un objeto.

Si prog no retorna sólo un objeto, se genera el error “Muy pocos argumentos”

Llama al comando ˆmake2dmatrix

377006 ˆmake2dmatrix ( #f #c prog 🡺 mat1...matf #f )

Crea un meta de matrices (filas) de #c elementos cada una.

prog es un programa que toma 2 argumentos enteros (el número de fila y de columna del elemento) y retorna sólo un objeto.

Si prog no retorna sólo un objeto, se genera el error “Muy pocos argumentos”

El siguiente NULLNAME realiza algo similar, pero es más rápido.

\* Crea una matriz de 2 dimensiones cuyos elementos estén

\* en función de su número de fila y de columna

\* El programa es de la forma ( -> ob )

\* Dicho programa no toma argumentos, pero debe retornar un objeto.

\* Puedes llamar el número de la fila como un bint con el comando JINDEX@

\* Puedes llamar el número de la columna como un bint con el comando INDEX@

\* Puedes llamar el número de filas como un bint con: JSTOP@ #1-

\* Puedes llamar el número de columnas como un bint con: ISTOP@ #1-

NULLNAME MAKE\_2DMATRIX ( #f #c prog -> 2DMATRIX )

:: ( #f #c prog )

ROT ( #c prog #f )

FLASHPTR 3LAMBIND ( )

1GETLAM ( #f )

#1+\_ONE\_DO (DO)

3GETLAM ( ... #c )

#1+\_ONE\_DO (DO)

2GETEVAL ( ... ... obi )

LOOP

3GETLAM ( ... ob1 ob2...obc #c )

TYPEMATRIX\_ ( ... ob1 ob2...obc #c 9862 )

COMPN\_ ( ... MATRIX )

LOOP

( MAT1 MAT2 ... MATf )

1GETABND ( MAT1 MAT2 ... MATf #f )

TYPEMATRIX\_ ( MAT1 MAT2 ... MATf #f 9862 )

COMPN\_ ( 2DMATRIX )

;

\* Crea un arreglo real (o complejo) de 2 dimensiones cuyos elementos estén

\* en función de su número de fila y de columna

\* El programa es de la forma ( -> % ) o de la forma ( -> C% )

\* No toma argumentos, pero debe retornar un número real (o complejo)

\* Puedes llamar el número de la fila como un bint con el comando JINDEX@

\* Puedes llamar el número de la columna como un bint con el comando INDEX@

\* Puedes llamar el número de filas como un bint con: JSTOP@ #1-

\* Puedes llamar el número de columnas como un bint con: ISTOP@ #1-

NULLNAME CREA\_2DARRAY ( #f #c prog -> [[%]]/[[C%]] )

:: ( #f #c prog )

ROT ( #c prog #f )

FLASHPTR 3LAMBIND ( )

1GETLAM ( #f )

#1+\_ONE\_DO (DO)

3GETLAM ( ... #c )

#1+\_ONE\_DO (DO)

2GETEVAL ( ... ... obi )

LOOP

LOOP

( ob11...obfc )

3GETLAM ( ob11...obfc #c )

1GETABND ( ob11...obfc #c #f )

SWAP2DUP ( ob11...obfc #f #c #f #c )

TWO{}N ( ob11...obfc #f #c {#f #c} )

UNROT ( ob11...obfc {#f #c} #f #c )

#\* ( ob11...obfc {#f #c} #f\*c )

FLASHPTR XEQ>ARRAY1 ( [[%]]/[[C%]] )

;

46.1.5 Redimensionando Matrices

**Direcc. Nombre Descripción**

341006 ˆMATREDIM ( RealArry #el 🡺 [%] )

( RealArry {#f #c} 🡺 [[%]] )

( CmpArry #el 🡺 [C%] )

( CmpArry {#f #c} 🡺 [[C%]] )

( MATRIX #el 🡺 1DMATRIX )

( MATRIX {#f #c} 🡺 2DMATRIX )

Cambia el tamaño de una formación. Si es necesario quita elementos o agrega elementos: real cero, complejo cero o entero cero (en el caso de matrices simbólicas).

Usado por el comando RDM de User RPL.

342006 ˆVRRDM ( MATRIX #el 🡺 1DMATRIX )

( MATRIX {#f #c} 🡺 2DMATRIX )

Similar a ˆMATREDIM pero funciona sólo para matrices simbólicas. Agrega el entero 0 si es necesario.

25F68 !REDIMTEMP ( RealArry #n #el #el #1 🡺 [%] )

( RealArry #n #fxc #f #c #2 🡺 [[%]] )

( CmpArry #n #el #el #1 🡺 [C%] )

( CmpArry #n #fxc #f #c #2 🡺 [[C%]] )

Similar a ˆMATREDIM pero funciona sólo para arreglos reales o complejos de una o de dos dimensiones.

#n es el número de objetos del arreglo original.

Comando tipo bang.

**Direcc. Nombre Descripción**

343006 ˆVRRDMmeta ( meta #N 🡺 meta' )

Cambia el número de elementos del meta para que este tenga N objetos. Quita elementos del meta o agrega el entero cero varias veces si es necesario.

46.1.6 Conversión

**Direcc. Nombre Descripción**

16A006 ˆ{}TO[] ( {} 🡺 1DMATRIX )

( {MATRIX} 🡺 2DMATRIX )

Solo cambia el prólogo de lista a prólogo de matriz simbólica.

17A006 ˆLIST2MATRIX ( {} 🡺 1DMATRIX )

( {{}} 🡺 2DMATRIX )

( {MATRIX} 🡺 2DMATRIX )

( ob 🡺 ob )

Convierte una lista a matriz simbólica.

Cada objeto de la matriz puede ser real, complejo, clase simbólica (id, lam, symb o entero), unidad o arreglo.

Si en la pila hay una lista vacía o una lista que contiene una lista vacía, genera el error “Dimensión inválida”.

Ninguno de los objetos de la matriz resultante debe ser lista o matriz simbólica (llama al comando ˆDIMS).

ob puede ser real, complejo, clase simbólica (id, lam, symb o entero), unidad, arreglo o matriz simbólica.

16B006 ˆ[]TO{} ( 1DMATRIX 🡺 {} )

( 2DMATRIX 🡺 {MATRIX} )

Convierte una matriz simbólica de una dimensión a lista.

Convierte una matriz simbólica de dos dimensiones a lista de matrices, donde cada matriz es una fila de la matriz original.

Solo cambia el prólogo de matriz simbólica a prólogo de lista.

179006 ˆMATRIX2LIST ( RealArry 🡺 {}/{{}} )

( CmpArry 🡺 {}/{{}} )

( MATRIX 🡺 {}/{{}} )

( ob 🡺 ob )

Si en la pila hay una matriz simbólica o arreglo real o complejo de una o de dos dimensiones, es convertido a lista o lista de listas.

Antes de todo, quita todas las etiquetas existentes.

No funciona para arreglos que no sean como los indicados.

002007 ˆArryToMatrix ( [[]] 🡺 2DMATRIX )

( [] 🡺 1DMATRIX )

Si en la pila hay un arreglo de una o de dos dimensiones, es convertido a matriz simbólica.

Este comando es mucho más rápido que ˆARRAY2MATRIX

No funciona para arreglos con más de dos dimensiones.

17E006 ˆARRAY2MATRIX ( [[]] 🡺 2DMATRIX )

( [] 🡺 1DMATRIX )

( MATRIX 🡺 MATRIX )

Si en la pila hay un arreglo de una o de dos dimensiones, es convertido a matriz simbólica.

No funciona para arreglos con más de dos dimensiones.

**Direcc. Nombre Descripción**

175006 ˆSAMEMATRIX ( MATRIX1 MATRIX2 🡺 MATRIX1 MATRIX2 T )

( MATRIX RealArry/CmpArry 🡺 MATRIX MATRIX' T )

( RealArry/CmpArry MATRIX 🡺 MATRIX' MATRIX T )

( ARRY1 ARRY2 🡺 ARRY1 ARRY2 F )

Si uno de los dos objetos es un arreglo y el otro es una matriz simbólica, convierte ambos a matriz simbólica (solo se podran convertir los arreglos que sean reales o complejos y que tengan una o dos dimensiones).

Retorna TRUE para matrices simbólicas y FALSE para arreglos.

176006 ˆSAMEMATSCTYPE ( MATRIX obintmat 🡺 MATRIX obintmat T )

( ARRY obfloat 🡺 ARRY obfloat F )

( RealArry obnofloat 🡺 MATRIX obnofloat T )

( CmpArry obnofloat 🡺 MATRIX obnofloat T )

Si en la pila hay un arreglo y un objeto no float, entonces convierte el arreglo a matriz. (sólo se podrán convertir arreglos reales o complejos de una o de dos dimensiones).

Retorna TRUE si en la pila se devuelve una matriz.

Retorna FALSE si en la pila se devuelve una arreglo.

obintmat es real, complejo, real extendido, complejo extendido, clase simbólica (id, lam, symb, entero), unidad, lista, programa o pointer que contiene a un programa.

obfloat es real, complejo, real extendido o complejo extendido.

obnofloat es de clase simbólica (id, lam, symb, entero) o lista.

17D006 ˆMATEXPLODE ( MATRIX 🡺 ob1..obn MATRIX )

Pone en la pila a todos los objetos de la matriz simbólica.

La matriz simbólica puede ser de una o de dos dimensiones.

46.1.7 Tests

**Direcc. Nombre Descripción**

004007 ˆDIMS ( 1DMATRIX 🡺 1DMATRIX #elem #0 T )

( 1DMATRIX 🡺 F )

( 2DMATRIX 🡺 2DMATRIX #c #f T )

( 2DMATRIX 🡺 F )

( {} 🡺 {} #elem #0 T )

( {{}} 🡺 {{}} #c #f T )

( {{}} 🡺 F )

Una matriz (o lista) se considera de dos dimensiones, si su primer elemento es una matriz (o lista). De los contrario se considera como de una dimensión.

Esa matriz (o lista) de dos dimensiones, se considera válida (y retornará TRUE), si todos sus elementos son matrices (o listas) que tienen la misma cantidad de objetos. De lo contrario, retorna FALSE

Si alguno de los objetos de la matriz es una lista o matriz, retorna FALSE

16C006 ˆDUPNULL[]? ( ob 🡺 ob flag )

Agrega TRUE, si ob es una matriz vacía ( MATRIX ; ).

**Direcc. Nombre Descripción**

359006 ˆNULLVECTOR? ( comp 🡺 flag )

¿Es un vector nulo?

Si la calculadora está en modo exacto (flag 105 desactivado), retorna TRUE, sólo cuando todos los elementos del compuesto sean iguales al entero cero.

Si la calculadora está en modo aproximado (flag 105 activado), retorna TRUE, cuando todos los elementos del compuesto son enteros, reales extendidos o complejos extendidos (debe haber por lo menos un número extendido) cuya suma de los cuadrados de sus valores absolutos (o módulos) sea menor al valor de la variable reservada EPS.

16F006 ˆCKSAMESIZE ( ARRY ARRY' 🡺 ARRY ARRY' flag )

( ARRY MATRIX' 🡺 ARRY MATRIX' flag )

( MATRIX MATRIX' 🡺 MATRIX MATRIX' flag )

( MATRIX ARRY' 🡺 MATRIX ARRY' flag )

Devuelve TRUE si tienen las mismas dimensiones.

170006 ˆDTYPENDO? ( RealArry 🡺 MATRIX flag )

( CmpArry 🡺 MATRIX flag )

( MATRIX 🡺 MATRIX flag )

¿Matriz cuadrada?

Retorna TRUE si la formación tiene dos dimensiones y las dos dimensiones tienen la misma longitud.

Arreglos reales o complejos son convertidos a matriz simbólica.

173006 ˆ2DMATRIX? ( ob 🡺 ob flag )

Retorna TRUE si en la pila hay una matriz de dos dimensiones.

171006 ˆDTYPFMAT? ( 2DMATRIX 🡺 2DMATRIX flag )

( 1DMATRIX 🡺 1DMATRIX F )

( ob 🡺 ob F )

Retorna TRUE si en la pila hay una matriz simbólica válida de dos dimensiones.

Verifica que cada elemento de la matriz (es decir, cada fila) sea una matriz o matriz etiquetada y que todas las filas tengan el mismo número de elementos.

No verifica que los elementos de la matriz sean de tipo válido.

3B5001 FLASHPTR 001 3B5 ( [[]] 🡺 [[]] #n )

( 2DMATRIX 🡺 2DMATRIX #n )

Agrega el número de filas de una formación cuadrada.

Si la formación es de 1 dimensión o no es cuadrada, genera el error: “Dimensión inválida”.

3B6001 FLASHPTR 001 3B6 ( ob1 ob2 🡺 ob1 ob2 )

Verifica que los objetos tengan las mismas dimensiones.

Los objetos pueden ser arreglos, matrices o arreglos vinculados.

Por ejemplo, si ob1 y ob2 son formaciones, ambas con 3 filas y 4 columnas, entonces no hace nada.

Si no tienen las mismas dimensiones, genera el error: “Dimensión inválida”.

46.1.8 Calculos con matrices

Los comandos de esta sección que funcionan para arreglos, aceptan arreglos reales o complejos de una o de dos dimensiones. No aceptan otro tipo de arreglos. A veces esto no se indica en los diagramas de pila por razones de espacio.

**Direcc. Nombre Descripción**

320006 ˆMAT+ ( MATRIX MATRIX' 🡺 MATRIX'' )

( Arry Arry' 🡺 Arry'' )

( MATRIX Arry 🡺 MATRIX' )

( Arry MATRIX 🡺 MATRIX' )

Suma de dos formaciones de las mismas dimensiones.

El arreglo debe ser real o complejo. Si en la pila hay un arreglo real y otro complejo, retorna un arreglo complejo.

Si las formaciones no tienen el mismo tamaño, genera el error

”Dimensión inválida”.

37E001 FLASHPTR 001 37E ( Arry Arry' 🡺 Arry'' )

Suma de dos arreglos.

321006 ˆMADD ( 2DMATRIX 2DMATRIX' 🡺 2DMATRIX'' )

Suma de dos matrices simbólicas de dos dimensiones.

324006 ˆVADD ( 1DMATRIX 1DMATRIX' 🡺 1DMATRIX'' )

Suma de dos matrices simbólicas de una dimension.

322006 ˆMAT- ( MATRIX MATRIX' 🡺 MATRIX'' )

( Arry Arry' 🡺 Arry'' )

( MATRIX Arry 🡺 MATRIX' )

( Arry MATRIX 🡺 MATRIX' )

Resta de dos formaciones de las mismas dimensiones.

El arreglo debe ser real o complejo. Si en la pila hay un arreglo real y otro complejo, retorna un arreglo complejo.

Si las formaciones no tienen el mismo tamaño, genera el error

”Dimensión inválida”.

380001 FLASHPTR 001 380 ( Arry Arry' 🡺 Arry'' )

Resta de dos arreglos.

323006 ˆMSUB ( 2DMATRIX 2DMATRIX' 🡺 2DMATRIX'' )

Resta de dos matrices simbólicas de dos dimensiones.

325006 ˆVSUB ( 1DMATRIX 1DMATRIX' 🡺 1DMATRIX'' )

Resta de dos matrices simbólicas de una dimension.

326006 ˆMAT\* ( MATRIX MATRIX' 🡺 MATRIX'' )

( Arry Arry' 🡺 Arry'' )

( MATRIX Arry 🡺 MATRIX' )

( Arry MATRIX 🡺 MATRIX' )

Producto de dos formaciones.

La primera formación debe ser de dos dimensiones.

El arreglo debe ser real o complejo. Si en la pila hay un arreglo real y otro complejo, retorna un arreglo complejo.

Si las formaciones no tienen dimensiones válidas, genera el error ”Dimensión inválida”.

327006 ˆMMMULT ( 2DMATRIX 2DMATRIX' 🡺 2DMATRIX'' )

( [[]] [[]]' 🡺 [[]]'' )

( 2DMATRIX [[]] 🡺 2DMATRIX' )

( [[]] 2DMATRIX 🡺 2DMATRIX' )

Producto de dos formaciones de dos dimensiones.

El arreglo debe ser real o complejo. Si en la pila hay un arreglo real y otro complejo, retorna un arreglo complejo.

**Direcc. Nombre Descripción**

328006 ˆMVMULT ( 2DMATRIX 1DMATRIX 🡺 1DMATRIX' )

( [[]] [] 🡺 []' )

( 2DMATRIX [] 🡺 1DMATRIX )

( [[]] 1DMATRIX 🡺 1DMATRIX' )

Producto de una formacion de dos dimensiones por una formación de una dimensión.

El arreglo debe ser real o complejo. Si en la pila hay un arreglo real y otro complejo, retorna un arreglo complejo.

38A001 FLASHPTR 001 38A ( [[]] [[]]' 🡺 [[]]'' )

( [[]] [] 🡺 []' )

Producto de dos arreglos.

El arreglo debe ser real o complejo. Si en la pila hay un arreglo real y otro complejo, retorna un arreglo complejo.

329006 ˆSCL\*MAT ( %/C%% Arry 🡺 Arry' )

( symbclass Arry 🡺 MATRIX )

( ob MATRIX 🡺 MATRIX' )

Multiplicación de escalar por matriz.

symbclass es de clase simbólica (id, lam, symb, entero).

ob es real, complejo o de clase simbólica.

32A006 ˆMAT\*SCL ( Arry %/C%% 🡺 Arry' )

( Arry symbclass 🡺 MATRIX )

( MATRIX ob 🡺 MATRIX' )

Multiplicación de matriz por escalar.

symbclass es de clase simbólica (id, lam, symb, entero).

ob es real, complejo o de clase simbólica.

382001 FLASHPTR 001 382 ( RealArry % 🡺 RealArry' )

( RealArry C% 🡺 CmpArry' )

( CmpArry %/C% 🡺 CmpArry' )

Multiplicación de un arreglo por un escalar.

32B006 ˆVPMULT ( [] symbclass 🡺 1DMATRIX )

( 1DMATRIX ob 🡺 1DMATRIX' )

Multiplicación de vector por escalar.

symbclass es de clase simbólica (id, lam, symb, entero).

ob es real, complejo o de clase simbólica.

335006 ˆMATSQUARE ( 2DMATRIX 🡺 2DMATRIX' )

( [[]] 🡺 [[]]' )

Multiplica una formación por si misma.

El número de filas y de columnas debe ser el mismo.

Hace DUP luego FLASHPTR MAT\*

32C006 ˆMATˆ ( 2DMATRIX z/% 🡺 2DMATRIX' )

( [[]] z/% 🡺 [[]]' )

Potencia de una matriz cuadrada.

El número real no debe tener parte decimal.

El exponente puede ser positivo o negativo.

Si el exponente es cero, retorna una matriz identidad.

También funciona para arreglos.

32E006 ˆMATDOT ( 1DMATRIX 1DMATRIX' 🡺 ob )

( [] []' 🡺 %/C%% )

( 1DMATRIX [] 🡺 ob )

( [] 1DMATRIX 🡺 ob )

Producto escalar de vectores con el mismo nº de elementos.

Equivale al comando DOT de User RPL.

**Direcc. Nombre Descripción**

331006 ˆXYext ( 1DMATRIX 1DMATRIX' 🡺 ob )

Producto escalar de dos vectores simbólicos con el mismo número de elementos.

32D006 ˆMATCROSS ( 1DMATRIX 1DMATRIX' 🡺 1DMATRIX'' )

( [] []' 🡺 []'' )

( 1DMATRIX [] 🡺 1DMATRIX' )

( [] 1DMATRIX 🡺 1DMATRIX' )

Producto vectorial de vectores. Cada vector puede tener dos o tres elementos.

Usado por el comando CROSS de User RPL.

32F006 ˆRNDARRY ( MATRIX % 🡺 MATRIX' )

( Array % 🡺 Array' )

Redondea todos los elementos de la matriz de acuerdo al valor del real con número de decimales (real positivo o cero) o número de cifras significativas (real negativo).

También funciona para arreglos.

330006 ˆTRCARRY ( MATRIX % 🡺 MATRIX' )

( Array % 🡺 Array' )

Trunca todos los elementos de la matriz de acuerdo al valor del real con número de decimales (real positivo o cero) o número de cifras significativas (real negativo).

También funciona para arreglos.

332006 ˆMAT/SCL ( Arry %/C%% 🡺 Arry' )

( Arry symbclass 🡺 MATRIX )

( MATRIX ob 🡺 MATRIX' )

Divide matriz por escalar.

symbclass es de clase simbólica (id, lam, symb, entero).

ob es real, complejo o de clase simbólica.

386001 FLASHPTR 001 386 ( RealArry % 🡺 RealArry' )

( RealArry C% 🡺 CmpArry' )

( CmpArry %/C% 🡺 CmpArry' )

División de un arreglo entre un escalar.

333006 ˆMAT/ ( MATRIX 2DMATRIX 🡺 MATRIX' )

( Arry [[]] 🡺 Arry' )

( MATRIX [[]] 🡺 MATRIX' )

( Arry 2DMATRIX 🡺 MATRIX )

Divide las formaciones. Es decir: A/B=INV(B)xA

334006 ˆMATCHS ( MATRIX 🡺 Arry )

( MATRIX 🡺 MATRIX' )

( Arry 🡺 Arry' )

Cambia de signo a cada elemento.

Si una matriz tiene todos sus elementos reales o todos complejos, entonces es convertida a arreglo.

370001 FLASHPTR 001 370 ( Arry 🡺 Arry' )

Cambia de signo a cada elemento del arreglo.

**Direcc. Nombre Descripción**

34E006 ˆMATINV ( MATRIX 🡺 Arry )

( MATRIX 🡺 MATRIX' )

( Arry 🡺 Arry' )

Halla la inversa de una matriz cuadrada.

También funciona para arreglos.

Si una matriz tiene todos sus elementos reales o todos complejos, entonces es convertida a arreglo.

Verifica que el arreglo sea dos dimensiones y estas dimensiones sean iguales.

04A003 FLASHPTR 003 04A ( Arry 🡺 Arry' )

Halla la inversa de un arreglo. Verifica que el arreglo sea de dos dimensiones y estas dimensiones sean iguales.

336006 ˆMATCONJ ( MATRIX 🡺 Arry )

( MATRIX 🡺 MATRIX' )

( CmpArry 🡺 CmpArry' )

( RealArry 🡺 RealArry )

Halla la conjugada de cada elemento.

Si una matriz tiene todos sus elementos reales o todos complejos, entonces es convertida a arreglo.

337006 ˆMATRE ( MATRIX 🡺 RealArry )

( MATRIX 🡺 MATRIX' )

( CmpArry 🡺 RealArry )

( RealArry 🡺 RealArry )

Halla la parte real de cada elemento.

Si una matriz tiene todos sus elementos reales o todos complejos, entonces es convertida a arreglo.

338006 ˆMATIM ( MATRIX 🡺 RealArry )

( MATRIX 🡺 MATRIX' )

( CmpArry 🡺 RealArry )

( RealArry 🡺 [%0]/[[%0]] )

Halla la parte imaginaria de cada elemento.

Si una matriz tiene todos sus elementos reales o todos complejos, entonces es convertida a arreglo.

339006 ˆMATTRACE ( 2DMATRIX 🡺 ob )

( [[%]] 🡺 % )

( [[C%]] 🡺 C% )

Traza de una matriz cuadrada. También funciona para arreglos.

La traza es igual a la suma de los elementos de la diagonal principal y también es igual a la suma de los valores propios de la matriz.

Equivale al comando TRACE de User RPL.

043003 FLASHPTR 003 043 ( [[%]] 🡺 % )

( [[C%]] 🡺 C% )

Similar a ˆMATTRACE, pero funciona sólo para arreglos.

46.1.9 Transpuesta

**Direcc. Nombre Descripción**

263D2 !MATTRNnc ( [[%]] 🡺 [[%]]' )

( [[C%]] 🡺 [[C%]]' )

Halla la transpuesta de un arreglo real o complejo de dos dimensiones (intercambiando el número de filas y el número de columnas).

Para arreglos con tres o más dimensiones, intercambia solamente las longitudes de las dos primeras dimensiones.

(comando tipo bang)

33B006 ˆMATTRAN ( [[%]] 🡺 [[%]]' )

( [[C%]] 🡺 [[C%]]' )

( 2DMATRIX 🡺 2DMATRIX' )

( 2DMATRIX 🡺 [[%]]' )

( 2DMATRIX 🡺 [[C%]]' )

Similar al comando !MATTRNnc pero funciona también para matrices simbólicas.

Si la matriz contiene sólo números reales, entonces también la convierte a arreglo real.

Si la matriz contiene sólo números complejos, entonces también la convierte a arreglo complejo.

Es usado por el comando TRAN de User RPL.

33C006 ˆmattran ( Meta-MATRIX 🡺 Meta-MATRIX' )

Transpone una matriz de 2 dimensiones que está como un meta en la pila. Es decir, sus filas en la pila, seguidas del número de filas, como un bint.

Por ejemplo, si en la pila se encuentra:

MATRIX 4. 5. 9. ;

MATRIX 14. 15. 19. ;

BINT2

Retorna:

MATRIX 4. 14. ;

MATRIX 5. 15. ;

MATRIX 9. 19. ;

BINT3

33A006 ˆMATTRN ( [[%]] 🡺 [[%]]' )

( [[C%]] 🡺 [[C%]]' )

( 2DMATRIX 🡺 2DMATRIX' )

( 2DMATRIX 🡺 [[%]]' )

( 2DMATRIX 🡺 [[C%]]' )

Transposición de una formación de dos dimensiones y conjugación de los elementos complejos.

Si la matriz contiene sólo números reales, entonces también la convierte a arreglo real.

Si la matriz contiene sólo números complejos, entonces también la convierte a arreglo complejo.

Es usado por el comando TRN de User RPL.

33D006 ˆmattrn ( Meta-MATRIX 🡺 Meta-MATRIX' )

Transpone una matriz de 2 dimensiones que está como un meta en la pila. Es decir, sus filas en la pila, seguidas del número de filas, como un bint.

También conjuga los elementos complejos.

46.1.10 Determinante

**Direcc. Nombre Descripción**

347006 ˆMATRDET ( 2DMATRIX 🡺 ob )

Halla el determinante de una matriz simbólica.

046003 FLASHPTR 003 046 ( [[%]] 🡺 % )

( [[C%]] 🡺 C% )

Halla el determinante.

El arreglo debe ser de dos dimensiones y cuadrado. Si esto no se cumple, genera el error: “Dimensión inválida”

346006 ˆMATDET ( [[%]] 🡺 % )

( [[C%]] 🡺 C% )

( 2DMATRIX 🡺 ob )

Halla el determinante.

La formación debe ser de dos dimensiones y cuadrada. Si esto no se cumple, genera el error: “Dimensión inválida”.

Si en la pila hay una matriz simbólica que no tiene todos sus elementos reales o todos complejos y cuyo tamaño es mayor a 4x4, entonces llama a ˆMATRDET

Usado por el comando DET de User RPL.

39E001 FLASHPTR 001 39E ( [[%]] #2 🡺 % )

( [[C%]] #2 🡺 C% )

Halla el determinante de un arreglo de 2x2.

46.1.11 Norma de una Matriz

**Direcc. Nombre Descripción**

348006 ˆMATFNORM ( MATRIX 🡺 ob )

( RealArry 🡺 % )

( CmpArry 🡺 % )

Norma Frobenius para formaciones de dos dimensiones.

Longitud euclideana para formaciones de una dimensión.

Retorna la raíz cuadrada de la suma de los cuadrados de los valores absolutos (o módulos, para números complejos) de todos sus elementos.

Usado por el comando ABS de User RPL para formaciones.

39C001 FLASHPTR 001 39C ( RealArry 🡺 % )

( CmpArry 🡺 % )

Similar a ˆMATFNORM, pero funciona sólo para arreglos.

349006 ˆMATRNORM ( MATRIX 🡺 ob )

( RealArry 🡺 % )

( CmpArry 🡺 % )

Norma de fila para una formación de dos dimensiones. Para cada fila, se halla la suma de los valores absolutos (o módulos, para números complejos) de todos sus elementos y se retorna la mayor de estas sumas.

Para una formación de una dimension, retorna el máximo valor de los valores absolutos (o módulos) de sus elementos.

Usado por el comando RNRM de User RPL.

398001 FLASHPTR 001 398 ( RealArry 🡺 % )

( CmpArry 🡺 % )

Similar a ˆMATRNORM, pero funciona sólo para arreglos.

**Direcc. Nombre Descripción**

34A006 ˆMATCNORM ( M 🡺 ob )

Norma de columna para una formación de dos dimensiones. Para cada columna, se halla la suma de los valores absolutos (o módulos, para números complejos) de todos sus elementos y se retorna la mayor de estas sumas.

Para una formación de una dimension, retorna la suma de los valores absolutos (o módulos) de sus elementos.

Usado por el comando CNRM de User RPL.

399001 FLASHPTR 001 399 ( RealArry 🡺 % )

( CmpArry 🡺 % )

Similar a ˆMATCNORM, pero funciona sólo para arreglos.

019003 FLASHPTR 003 019 ( RealArry 🡺 % )

( CmpArry 🡺 % )

Halla el radio espectral de un arreglo de dos dimensiones y cuadrado.

174006 ˆMATRIXDIM ( 1DMATRIX 🡺 #1 )

( 2DMATRIX 🡺 #2 )

( ob 🡺 #0 )

Retorna la dimensión de la matriz simbólica.

Para otros objetos, retorna cero.

46.1.12 Algebra Lineal y Reducción Gaussiana

**Direcc. Nombre Descripción**

04F003 FLASHPTR 003 04F ( [[%]] 🡺 [[%]]' )

( [[C%]] 🡺 [[C%]]' )

Retorna la forma escalonada reducida por filas para arreglos.

34B006 ˆMATRREF ( [[%]] 🡺 [[%]]' )

( [[C%]] 🡺 [[C%]]' )

( 2DMATRIX 🡺 [[%]]'/[[C%]]'/2DMATRIX' )

Retorna la forma escalonada reducida por filas.

Si en la pila hay un arreglo, llama a FLASHPTR 3 4F

Si en la pila hay una matriz con todos sus elementos reales o complejos, la convierte a arreglo y llama a FLASHPTR 3 4F

Otras matrices simbólicas deben de tener sólo enteros, ids, lams u objetos simbólicos que no tengan números reales ni complejos. En este caso la calculadora debe estar en modo exacto (flag 105 desctivado).

Equivale al comando RREF de User RPL.

34C006 ˆMATREF ( [[%]] 🡺 [[%]]' )

( [[C%]] 🡺 [[C%]]' )

( 2DMATRIX 🡺 [[%]]'/[[C%]]'/2DMATRIX' )

Si en la pila hay un arreglo o una matriz con todos sus elementos reales o todos complejos, retorna una arreglo en la forma escalonada reducida por filas (con FLASHPTR 3 4F).

Otras matrices simbólicas deben de tener sólo enteros, ids, lams u objetos simbólicos que no tengan reales ni complejos y se retorna la matriz escalonada por filas. En este caso la calculadora debe estar en modo exacto (flag 105 desactivado).

Equivale al comando REF de User RPL.

**Direcc. Nombre Descripción**

34F006 ˆMATREFRREF ( 2DMATRIX #full\_ref 🡺 {pivotes} 2DMATRIX' )

Si #full\_ref es 1, retorna una matriz parecida a la forma escalonada reducida por filas (la diferencia es que los primeros elementos no nulos de cada fila no son siempre 1).

Si #full\_ref no es 1, retorna una matriz parecida a la forma escalonada por filas (la diferencia es que los primeros elementos no nulos de cada fila no son siempre 1).

2DMATRIX debe tener sólo enteros, ids, lams u objetos simbólicos que no tengan números reales ni complejos. La calculadora debe estar en modo exacto (flag 105 desctivado).

223006 ˆrref ( 2DMATRIX 🡺 {pivotes} 2DMATRIX' )

Hace:

:: FLASHPTR STOMAText BINT1 FLASHPTR MATREFRREF ;

Retorna una matriz parecida a la forma escalonada reducida por filas (la diferencia es que los primeros elementos no nulos de cada fila no son siempre 1). La calculadora debe estar en modo exacto (flag 105 desactivado).

Equivale al comando rref de User RPL.

367006 ˆMATRIXRCI ( ob #i 2DMAT constante 🡺 2DMAT' )

( ob #i 1DMAT constante 🡺 1DMAT' )

Multiplica la fila i de una matriz de 2 dimensiones por la constante.

Multiplica el elemento i de una matriz de 1 dimensión por la constante.

ob es un objeto cualquiera.

058003 FLASHPTR 003 058 ( 2DMAT constante %i/Zi 🡺 2DMAT' )

( [[]] constante %i/Zi 🡺 [[]]'/2DMAT' )

( 1DMAT constante %i/Zi 🡺 1DMAT' )

( [] constante %i/Zi 🡺 []'/1DMAT' )

Multiplica la fila i de una formación de 2 dimensiones por la constante.

Multiplica el elemento i de una formación de 1 dimensión por la constante.

Si en la pila hay un arreglo, este debe ser real o complejo.

Equivale al comando RCI de User RPL.

368006 ˆMATRIXRCIJ ( ob #i #j 2DMAT const 🡺 2DMAT' )

( ob #i #j 1DMAT const 🡺 1DMAT' )

Multiplica la fila i de una matriz de 2 dimensiones por la constante y la suma a la fila j.

Multiplica el elemento i de una matriz de 1 dimensión por la constante y la suma al elemento j.

ob es un objeto cualquiera.

059003 FLASHPTR 003 059 ( 2DMAT constante %i/Zi %j/Zj 🡺 2DMAT' )

( [[]] constante %i/Zi %j/Zj 🡺 [[]]'/2DMAT' )

( 1DMAT constante %i/Zi %j/Zj 🡺 1DMAT' )

( [] constante %i/Zi %j/Zj 🡺 []'/1DMAT' )

Multiplica la fila i de una formación de 2 dimensiones por la constante y la suma a la fila j.

Multiplica el elemento i de una formación de 1 dimensión por la constante y la suma al elemento j.

Si en la pila hay un arreglo, este debe ser real o complejo.

Equivale al comando RCIJ de User RPL.

**Direcc. Nombre Descripción**

03B003 FLASHPTR 3 3B ( [[%]] 🡺 %rango )

( [[C%]] 🡺 %rango )

Retorna el rango para un arreglo de dos dimensiones.

34D006 ˆMATRANK ( [[%]] 🡺 % )

( [[C%]] 🡺 % )

( 2DMATRIX 🡺 Z )

Retorna el rango.

Equivale al comando RANK de User RPL.

46.1.13 Resolviendo Sistemas Lineales

**Direcc. Nombre Descripción**

22A006 ˆSYSTEM ( Meq Minc 🡺 {Meq Minc} Lpiv Mresult )

( Seq Sinc 🡺 {Meq Minc} Lpiv Mresult )

Resuelve un sistema de ecuaciones lineales.

En el nivel 2 debe estar una matriz de una dimensión con las ecuaciones.

En el nivel 1 debe estar una matriz de una dimensión con las variables como ids.

Retorna los datos en una lista, una lista etiquetada con los pivotes y una matriz de una dimensión con los resultados de la forma ‘id=valor’

También funciona cuando los argumentos son simbólicos donde cada parte está separada por el comando AND.

Por ejemplo, en los niveles 2 y de la pila pueden estar:

:2: 'X+Y=8 AND X-Y=3'

:1: 'X AND Y'

Equivale al comando LINSOLVE de User RPL.

357006 ˆMAKESYSText ( Meq Minc 🡺 Meq 2DMAT lidnt flag )

Convierte un sistema de ecuaciones lineales a una matriz aumentada y retorna TRUE si las ecuaciones son lineales respecto a las variables especificadas en el nivel 1.

356006 ˆSTOSYSText ( Meq Minc 🡺 {Meq Minc} Meq 2DMAT lvar )

355006 ˆSYSText ( 2DMAT lvar 🡺 lvar Mvar Mresult Lpiv )

148007 FLASHPTR 007 148 ( Meq Minc 🡺 2DMAT )

Convierte un sistema de ecuaciones lineales a una matriz aumentada.

Verifica que las ecuaciones sean lineales.

Equivale al comando SYST2MAT de User RPL cuando en la pila hay 2 matrices simbólicas.

46.1.14 Otras operaciones con matrices

Puedes usar algunos comandos explicados en el capítulo 10, entre ellos:

ˆPULLEL[S] para obtener un elemento de un matriz simbólica.

ˆFINDELN para obtener la posición de un elemento dada su ubicación.

ˆXEQ>ARRY para crear una matriz simbólica.

ˆXEQARRY> para poner en la pila los elementos de una matriz simbólica.

**Direcc. Nombre Descripción**

35C006 ˆBANGARRY ( % #ubic RealArry 🡺 RealArry' )

( C% #ubic CmpArry 🡺 CmpArry' )

( symbclass #ubic RealArry 🡺 MATRIX )

( symbclass #ubic CmpArry 🡺 MATRIX )

( ob #ubic MATRIX 🡺 MATRIX' )

Pone el elemento en la ubicación indicada con #ubic

symbclass es de clase simbólica (id, lam, symb, entero).

ob es real, complejo o de clase simbólica.

35D006 ˆPUT[] ( ob #i 1DMATRIX 🡺 1DMATRIX' )

( 1DMATRIX #fila 2DMATRIX 🡺 2DMATRIX' )

Reemplaza el elemento de lugar #i de un vector por ob

También puede usarse para cambiar una fila de una matriz simbólica de dos dimensiones.

17B006 ˆLENMATRIX ( 1DMATRIX 🡺 #elem )

( 2DMATRIX 🡺 #filas )

( ob 🡺 #0 )

33E006 ˆMATSUB ( 2DMAT #fmin #nfil #cmin #ncol {ob ob’} 🡺 2DMAT' )

( 2DMAT ob ob’ #fmin #nfil {ob’’} 🡺 2DMAT' )

( 1DMAT ob ob’ #posmin #nelem {ob’’} 🡺 1DMAT' )

Extrae una submatriz de una matriz simbólica.

ob, ob’ y ob’’ son objetos cualesquiera.

Para el segundo y tercer caso mostrados, también puedes usar el comando SUBCOMP

0250E8 ROMPTR 0E8 025 ( 2DMATRIX #fmin #cmin #fmax #cmax 🡺 2DMATRIX' )

( [[]] #fmin #cmin #fmax #cmax 🡺 [[]]' )

( 1DMATRIX #1 #posmin #1 #posmax 🡺 1DMATRIX' )

( [] #1 #posmin #1 #posmax 🡺 []' )

Extrae una submatriz. También funciona para arreglos.

Arreglos deben ser reales o complejos.

Si hay una matriz simbólica en la pila, llama a ˆMATSUB

0240E8 ROMPTR 0E8 024 ( 2DMATRIX {%f %c} {%f' %c'} 🡺 2DMATRIX' )

( [[]] {%f %c} {%f' %c'} 🡺 [[]]' )

( 1DMATRIX {%1 %ubic} {%1 %ubic'} 🡺 1DMATRIX' )

( [] {%1 %ubic} {%1 %ubic'} 🡺 []' )

Extrae una submatriz. También funciona para arreglos.

Arreglos deben ser reales o complejos.

Usado por el comando SUB de User RPL.

Llama a ROMPTR 0E8 025

**Direcc. Nombre Descripción**

0260E8 ROMPTR 0E8 026 ( 2DMATRIX %ubic %ubic' 🡺 2DMATRIX' )

( [[]] %ubic %ubic' 🡺 [[]]' )

( 1DMATRIX %ubic %ubic' 🡺 1DMATRIX' )

( [] %ubic %ubic' 🡺 []' )

Extrae una submatriz. También funciona para arreglos.

Los arreglos deben ser reales o complejos.

Usado por el comando SUB de User RPL.

Llama a ROMPTR 0E8 025

340006 ˆMATREPL ( 2DMATRIX 2DMATRIX' 🡺 2DMATRIX'' )

( 1DMATRIX 2DMATRIX 🡺 2DMATRIX' )

( 1DMATRIX 1DMATRIX' 🡺 1DMATRIX'' )

Reemplaza la matriz del nivel 2 en la matriz del nivel 1.

Previamente deben crearse por lo menos 9 nombres locales.

Lam9 y lam8 indican la posición donde se reemplazará.

Lam7 es un flag. TRUE, si la matriz donde se reemplazará es 2D. FALSE, si es 1D.

Lam3 es un flag. TRUE, si la matriz que se va a reemplazar es 2D. FALSE, si es 1D.

Lam2 y Lam1 indican las dimensiones de la matriz que se va a reemplazar.

Dentro del comando ˆMATREPL se llama a ABND. Por lo tanto, ya no es necesario llamarlo después en tu programa.

Abajo indicamos los valores que deben tener. Los casilleros en blanco pueden albergar cualquier objeto.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 2DMAT 2DMAT’ | 1DMAT 2DMAT | 1DMAT 1DMAT’ |
| LAM 9 | #f posición | #f posición |  |
| LAM 8 | #c posición | #c posición | #posición |
| LAM 7 | TRUE | TRUE | FALSE |
| LAM 6 |  |  |  |
| LAM 5 |  |  |  |
| LAM 4 |  |  |  |
| LAM 3 | TRUE | FALSE | FALSE |
| LAM 2 | #filas | #1 |  |
| LAM 1 | #columnas | #elem | #elem |

0210E8 ROMPTR 0E8 021 ( MATRIX 2DMATRIX #fil #col 🡺 2DMATRIX' )

( RealArry [[%]] #fil #col 🡺 [[%]]' )

( CmpArry [[C%]] #fil #col 🡺 [[C%]]' )

( 1DMATRIX 1DMATRIX' #1 #pos 🡺 1DMATRIX'' )

( RealArry [%] #1 #pos 🡺 [%]' )

( CmpArry [C%] #1 #pos 🡺 [C%]' )

Reemplaza la formación del nivel 4 en la formación del nivel 3.

Si hay matrices simbólicas en la pila, llama a ˆMATREPL

0200E8 ROMPTR 0E8 020 ( 2DMATRIX {%fil %col} MATRIX 🡺 2DMATRIX' )

( [[%]] {%fil %col} RealArry 🡺 [[%]]' )

( [[C%]] {%fil %col} CmpArry 🡺 [[C%]]' )

( 1DMATRIX {%1 %pos} 1DMATRIX' 🡺 1DMATRIX'' )

( [%] {%1 %pos} RealArry 🡺 [%]' )

( [C%] {%1 %pos} CmpArry 🡺 [C%]' )

Reemplaza la formación del nivel 3 en la formación del nivel 1.

Usado por el comando REPL de User RPL.

Llama a ROMPTR 0E8 021

**Direcc. Nombre Descripción**

01F0E8 ROMPTR 0E8 01F ( 2DMATRIX %ubic MATRIX 🡺 2DMATRIX' )

( [[%]] %ubic RealArry 🡺 [[%]]' )

( [[C%]] %ubic CmpArry 🡺 [[C%]]' )

( 1DMATRIX %ubic 1DMATRIX' 🡺 1DMATRIX'' )

( [%] %ubic RealArry 🡺 [%]' )

( [C%] %ubic CmpArry 🡺 [C%]' )

Reemplaza la formación del nivel 3 en la formación del nivel 1.

Usado por el comando REPL de User RPL.

Llama a ROMPTR 0E8 021

35F006 ˆMATRIX>DIAG ( 2DMATRIX ob #term 🡺 1DMATRIX )

Pone en la pila un vector con los términos de la diagonal principal de la matriz simbólica de dos dimensiones.

ob es un objeto cualquiera.

#term es el número de términos que tendrá el vector.

#1 ≤ #term ≤ min(#filas,#col)

Si #term=min(#filas,#col), se obtienen todos los términos de la diagonal principal (como el comando 🡪DIAG de User RPL)

06A003 FLASHPTR 003 06A ( 2DMATRIX 🡺 1DMATRIX )

( [[%]] 🡺 [%] )

( [[C%]] 🡺 [C%] )

Obtiene un vector con los elementos de la diagonal principal.

Si en la pila hay una matriz simbólica de dos dimensiones, llama a ˆMATRIX>DIAG

Equivale al comando 🡪DIAG de User RPL.

360006 ˆMATRIXDIAG> ( ob 1DMATRIX ob' {#fil #col} 🡺 2DMATRIX )

Retorna una matriz de dos dimensiones (las del nivel 1), donde los elementos de su diagonal principal serán los elementos del vector del nivel 3.

El resto de sus elementos serán ceros (enteros).

ob y ob' son dos objetos cualesquiera.

06B003 FLASHPTR 003 06B ( 1DMATRIX {#fil #col} 🡺 2DMATRIX )

( [%] {#fil #col} 🡺 [[%]] )

( [C%] {#fil #col} 🡺 [[C%]] )

Retorna una formación de dos dimensiones (las del nivel 1), donde los elementos de su diagonal principal serán los elementos del vector del nivel 2. El resto de sus elementos serán ceros (enteros, reales o complejos, respectivamente).

Si en la pila hay un vector simbólico, llama a ˆMATRIXDIAG>

04D002 FLASHPTR 002 04D ( 1DMATRIX {%fil %col} 🡺 2DMATRIX )

( [%] {%fil %col} 🡺 [[%]] )

( [C%] {%fil %col} 🡺 [[C%]] )

Similar a FLASHPTR 3 6B pero las dimensiones se indican con números reales.

Llama al comando FLASHPTR 3 6B

Equivale al comando DIAG🡪 de User RPL cuando en la pila hay un vector y una lista.

**Direcc. Nombre Descripción**

04E002 FLASHPTR 002 04E ( 1DMATRIX %n 🡺 2DMATRIX )

( [%] %n 🡺 [[%]] )

( [C%] %n 🡺 [[C%]] )

Similar a FLASHPTR 2 4D pero la formación creada será cuadrada con número de filas y de columnas igual a %n

Llama al comando FLASHPTR 3 6B

Equivale al comando DIAG🡪 de User RPL cuando en la pila hay un vector y un número real.

370006 ˆSTOMAText ( MATRIX 🡺 MATRIX )

Guarda MATRIX en el directorio CASDIR, sólo cuando al aplicar LVARext a MATRIX resulte una lista no vacía.

379006 ˆVUNARYOP ( 1DMATRIX op 🡺 1DMATRIX' )

( comp op 🡺 1DMATRIX' )

Aplica la operación a cada elemento del vector simbólico.

También podría aplicarse a cada elemento de otro compuesto que al final se convertirá a vector simbólico.

op es un programa o comando que toma un argumento y devuelve un objeto.

37A006 ˆVBINARYOP ( 1DMATRIX 1DMATRIX' op 🡺 1DMATRIX'' )

( comp comp' op 🡺 1DMATRIX'' )

Aplica la operación binaria a los elementos de los vectores simbólicos de la pila (deben de tener el mismo número de elementos).

También podría aplicarse a cada elemento de otros compuestos. Al final se convertirá a vector simbólico.

op es un programa o comando que toma dos argumento y devuelve un objeto.

37B006 ˆPEVAL ( [%]/[C%]/1DMATRIX ob 🡺 ob' )

Evaluación del polinomio (cuyos coeficientes están en el vector del nivel 2) en el objeto ob.

ob es real, complejo o de clase simbólica (id, lam, symb o entero).

Equivale al comando PEVAL de USER RPL.

**Direcc. Nombre Descripción**

378006 ˆADDMATOBJext ( 2DMATRIX/[[]] ob 🡺 2DMATRIX/[[]] 2DMATRIX' )

( 2DMATRIX % 🡺 2DMATRIX 2DMATRIX'/[[%]] )

( [[]] % 🡺 [[]] 2DMATRIX'/[[%]] )

( 2DMATRIX C% 🡺 2DMATRIX 2DMATRIX'/[[C%]] )

( [[]] C% 🡺 [[]] 2DMATRIX'/[[C%]] )

( 2DMATRIX sc 🡺 2DMATRIX 2DMATRIX' )

( [[]] sc 🡺 2DMATRIX 2DMATRIX' )

( % [[]] 🡺 2DMATRIX'/[[%]] [[]] )

( C% [[]] 🡺 2DMATRIX'/[[%]] [[]] )

( sc [[]] 🡺 2DMATRIX' [[]] )

( 2DMATRIX/[[]] 2DMATRIX' 🡺 2DMATRIX 2DMATRIX' )

Crea una matriz escalar (los elementos de la diagonal principal iguales a la constante y el resto de elementos son cero) cuyas dimensiones son las mismas que la formación dada en la pila.

La formación de la pila debe ser de 2 dimensiones y cuadrada.

Si la constante es real o compleja crea un arreglo (modo aproximado, flag 105 activado) o una matriz simbólica (modo exacto, flag 105 desactivado).

Si la constante es de clase simbólica (id, lam, symb o entero), crea siempre una matriz con sus elementos fuera de la diagonal principal iguales al real cero (modo aproximado, flag 105 activado) o al entero cero (modo exacto, flag 105 desactivado)

Por último, si en la pila hay una formación cuadrada y una matriz de dos dimensiones, sólo convierte la formación del nivel 2 a matriz simbólica, si esta era un arreglo.

46.1.15 Insertar Filas en Matrices e Insertar objetos en Vectores

**Direcc. Nombre Descripción**

362006 ˆINSERTROW[] ( 1DMATRIX ob #i 🡺 1DMATRIX' )

( 2DMATRIX 1DMATRIX #i 🡺 2DMATRIX' )

Inserta ob en un vector simbólico en la posición #i

Inserta vector simbólico en la matrix simbólica en la fila #i

Si no cumple #1 ≤ #i ≤ #n+1, genera el error

“Argumento: valor incorr”.

No verifica que el número de elementos de 1DMATRIX y el número de columnas de 2DMATRIX sean iguales.

365006 ˆINSERT[]ROW[] ( 2DMATRIX 2DMATRIX' #i 🡺 2DMATRIX'' )

Inserta 2DMATRIX' en 2DMATRIX empezando en la fila #i

Verifica que #1 ≤ #i ≤ #n+1, pero no los tamaños de matrices.

064003 FLASHPTR 003 064 ( %i [%] %/%% 🡺 [%]' )

( %i [C%] C%/C%% 🡺 [C%]' )

Inserta el objeto del nivel 1 de la pila en un arreglo real o complejo de una dimensión en la posición %i

**Direcc. Nombre Descripción**

361006 ˆla+ELEMsym ( 1DMATRIX ob %i 🡺 1DMATRIX' )

( [] ob %i 🡺 1DMATRIX )

( [[]] 1DMATRIX %fila 🡺 2DMATRIX )

Inserta ob en el vector en la posición %i

Inserta vector 1DMATRIX en el arreglo [[]], en la fila %i

Llama a ˆINSERTROW[]

No verifica que el número de elementos de 1DMATRIX y el número de columnas de [[]] sean iguales.

El arreglo debe ser real o complejo.

Equivale al comando ROW+ de User RPL y también al comando COL+ de User RPL cuando en la pila hay 1 formación, un objeto de clase simbólica y un número real.

05D003 FLASHPTR 003 05D ( 2DMATRIX 2DMATRIX' %i 🡺 2DMATRIX'' )

( 2DMATRIX [[]] %i 🡺 2DMATRIX'' )

( 2DMATRIX 1DMATRIX %i 🡺 2DMATRIX'' )

( 2DMATRIX [] %i 🡺 2DMATRIX'' )

( [[]] 2DMATRIX %i 🡺 2DMATRIX'' )

( [[]] 1DMATRIX %i 🡺 2DMATRIX'' )

( [[]] [[]]' %i 🡺 [[]]'' )

( [[]] [] %i 🡺 [[]]'' )

Inserta la formación del nivel 2 en la del nivel 3 en la fila #i

Los arreglos deben ser reales o complejos.

Si hay un arreglo real y otro complejo, devuelve uno complejo.

Equivale al comando ROW+ de User RPL cuando en la pila hay 2 formaciones y un número real.

062003 FLASHPTR 003 062 ( 1DMATRIX ob %i 🡺 1DMATRIX' )

( [%] %/%% %i 🡺 [%]' )

( [C%] % %i 🡺 [C%]' )

Inserta el objeto del nivel 2 en el vector del nivel 3 en la posición %i

Equivale al comando ROW+ de User RPL y también al comando COL+ de User RPL cuando en la pila hay 1 formación y dos números reales.

063003 FLASHPTR 003 063 ( 1DMATRIX ob %i 🡺 1DMATRIX' )

( [C%] C%/C%% %i 🡺 [C%]' )

( [%] C% %i 🡺 [C%] )

Inserta el objeto del nivel 2 en el vector del nivel 3 en la posición %i

Equivale al comando ROW+ de User RPL y también al comando COL+ de User RPL cuando en la pila hay 1 formación, un número complejo y un número real.

**Direcc. Nombre Descripción**

146007 FLASHPTR 7 146 ( 1DMATRIX/[] 1DMATRIX/[] 🡺 2DMATRIX )

( 1DMATRIX %/C%/id/lam/symb/Z 🡺 1DMATRIX' )

( [] %/C%/id/lam/symb/Z 🡺 1DMATRIX' )

( 2DMATRIX/[[]] MATRIX/Arry 🡺 2DMATRIX' )

Si hay dos fomaciones de una dimensión en la pila (con el mismo número de elementos), las pone como filas en una matriz simbólica de dos dimensiones.

Si hay una fomación de una dimensión y otro objeto en la pila, convierte la formación a matriz simbólica y le agrega el objeto.

Si en el nivel 2 hay una formación de dos dimensiones, le agrega la formación del nivel 1 como filas en la parte inferior.

Los arreglos deben ser reales o complejos.

Equivale al comando AUGMENT de User RPL cuando en la pila hay una formación en el nivel 2.

Equivale al comando AUGMENT de User RPL cuando en la pila hay una formación en el nivel 2 y otro objeto en el nivel 1.

46.1.16 Insertar Columnas

**Direcc. Nombre Descripción**

364006 ˆINSERTCOL[] ( 2DMATRIX 1DMATRIX #i 🡺 2DMATRIX' )

Inserta vector simbólico en la matrix simbólica en columna #i

Si no cumple #1 ≤ #i ≤ #n+1, genera el error

“Argumento: valor incorr”.

No verifica que el número de elementos de 1DMATRIX y el número de filas de 2DMATRIX sean iguales.

366006 ˆINSERT[]COL[] ( 2DMATRIX 2DMATRIX' #i 🡺 2DMATRIX'' )

Inserta 2DMATRIX' en 2DMATRIX empezando en la fila #i

Verifica que #1 ≤ #i ≤ #n+1, pero no los tamaños de matrices.

060003 FLASHPTR 3 60 ( 2DMATRIX 2DMATRIX' %i 🡺 2DMATRIX'' )

( 2DMATRIX [[]] %i 🡺 2DMATRIX'' )

( 2DMATRIX 1DMATRIX %i 🡺 2DMATRIX'' )

( 2DMATRIX [] %i 🡺 2DMATRIX'' )

( [[]] 2DMATRIX %i 🡺 2DMATRIX'' )

( [[]] 1DMATRIX %i 🡺 2DMATRIX'' )

( [[]] [[]]' %i 🡺 [[]]'' )

( [[]] [] %i 🡺 [[]]'' )

Inserta la formación del nivel 2 en la del nivel 3 a partir de la columna %i

Los arreglos deben ser reales o complejos.

Si hay un arreglo real y otro complejo, devuelve uno complejo.

Equivale al comando COL+ de User RPL cuando en la pila hay 2 formaciones y un número real.

46.1.17 Intercambio o Extracción de Filas, Columnas y Elementos

**Direcc. Nombre Descripción**

0AC003 ˆSWAPROWS ( [[%]] #f1 #f2 🡺 [[%]]' #f1 #f2 )

( [[C%]] #f1 #f2 🡺 [[C%]]' #f1 #f2 )

( [%] #pos1 #pos2 🡺 [%]' #pos1 #pos2 )

( [C%] #pos1 #pos2 🡺 [C%]' #pos1 #pos2 )

En arreglos reales o complejos de dos dimensiones, intercambia dos filas.

En arreglos reales o complejos de una dimensión, intercambia las posiciones de dos elementos.

(comando tipo bang)

36A006 ˆMATRIXRSWAP ( 2DMATRIX #f1 #f2 🡺 2DMATRIX' )

( 1DMATRIX #pos1 #pos2 🡺 1DMATRIX' )

Similar al comando ˆSWAPROWS pero sólo funciona para matrices simbólicas.

056003 FLASHPTR 3 56 ( 2DMATRIX #f1 #f2 #filas 🡺 2DMATRIX' )

( [[%]] #f1 #f2 #filas 🡺 [[%]]' )

( [[C%]] #f1 #f2 #filas 🡺 [[C%]]' )

( 1DMATRIX #pos1 #pos2 #elem 🡺 1DMATRIX' )

( [%] #pos1 #pos2 #elem 🡺 [%]' )

( [C%] #pos1 #pos2 #elem 🡺 [C%]' )

En formaciones de dos dimensiones, intercambia dos filas.

En formaciones de una dimensión, intercambia las posiciones de dos elementos.

Si en la pila hay una matriz simbólica, llama a ˆMATRIXRSWAP

Si en la pila hay un arreglo real o complejo de dos dimensiones, llama a ˆSWAPROWS

055003 FLASHPTR 3 55 ( 2DMATRIX %f1 %f2 🡺 2DMATRIX' )

( [[%]] %f1 %f2 🡺 [[%]]' )

( [[C%]] %f1 %f2 🡺 [[C%]]' )

( 1DMATRIX %pos1 %pos2 🡺 1DMATRIX' )

( [%] %pos1 %pos2 🡺 [%]' )

( [C%] %pos1 %pos2 🡺 [C%]' )

En formaciones de dos dimensiones, intercambia dos filas.

En formaciones de una dimensión, intercambia las posiciones de dos elementos.

Verifica que los dos números reales sean válidos.

Llama a FLASHPTR 003 056

Equivale al comando RSWP de User RPL.

369006 ˆMATRIXCSWAP ( 2DMATRIX #c1 #c2 🡺 2DMATRIX' )

Intercambia dos columnas de una matriz simbólica de dos dimensiones.

36E006 ˆMETAMATCSWAP ( Meta-MATRIX #c1 #c2 🡺 Meta-MATRIX' )

Intercambia columnas en una matriz de 2 dimensiones que está como un meta en la pila. Es decir, sus filas en la pila, seguidas del número de filas, como un bint.

**Direcc. Nombre Descripción**

057003 FLASHPTR 3 57 ( 2DMATRIX %c1 %c2 🡺 2DMATRIX' )

( [[%]] %c1 %c2 🡺 [[%]]' )

( [[C%]] %c1 %c2 🡺 [[C%]]' )

( 1DMATRIX %pos1 %pos2 🡺 1DMATRIX' )

( [%] %pos1 %pos2 🡺 [%]' )

( [C%] %pos1 %pos2 🡺 [C%]' )

En formaciones de 2 dimensiones, intercambia dos columnas. En formaciones de 1 dimensión, intercambia las posiciones de dos elementos.

Si en la pila hay una matriz simbólica de dos dimensiones, llama a ˆMATRIXCSWAP

Equivale al comado CSWP de User RPL.

36B006 ˆMATRIX-ROW ( 2DMATRIX #fila 🡺 2DMATRIX' 1DMATRIX )

( 1DMATRIX #i 🡺 1DMATRIX' obi )

En matrices de 2 dimensiones, extrae la fila #fila

En matrices de 1 dimensión, extrae el objeto de la posición #i

Si no cumple #1 ≤ # ≤ #n, genera el error

“Argumento: valor incorr”.

05B003 FLASHPTR 3 5B ( [%] #i #elemfinal+1 🡺 [%]' %posi )

( [C%] #i #elemfinal+1 🡺 [C%]' C%posi )

Extrae el objeto del vector de la posición #i

El arreglo devuelto tendrá #elemfinal elementos.

05A003 FLASHPTR 3 5A ( 2DMATRIX %fila 🡺 2DMATRIX' 1DMATRIX )

( 1DMATRIX %i 🡺 1DMATRIX' obi )

( [[%]] %fila 🡺 [[%]]' [%] )

( [%] %i 🡺 [%]' %posi )

( [[C%]] %fila 🡺 [[C%]]' [C%] )

( [C%] %i 🡺 [C%]' C%posi )

Para una formación de dos dimensiones, extrae una fila.

Para una formación de una dimensión, extrae el elemento de la posición #i

Si hay una matriz simbólica en la pila, llama a ˆMATRIX-ROW

Si hay un arreglo de una dimensión en la pila, llama al comando FLASHPTR 3 5B

Equivale al comando ROW- de User RPL.

36D006 ˆMATRIX-COL ( 2DMATRIX #col 🡺 2DMATRIX' 1DMATRIX )

( 1DMATRIX #i 🡺 1DMATRIX' obi )

En matrices de 2 dimensiones, extrae la columna #col

En matrices de 1 dimensión, extrae el objeto de la posición #i

Si no cumple #1 ≤ # ≤ #n, genera el error

“Argumento: valor incorr”.

**Direcc. Nombre Descripción**

05C003 FLASHPTR 3 5C ( 2DMATRIX %col 🡺 2DMATRIX' 1DMATRIX )

( 1DMATRIX %i 🡺 1DMATRIX' obi )

( [[%]] %col 🡺 [[%]]' [%] )

( [%] %i 🡺 [%]' %posi )

( [[C%]] %col 🡺 [[C%]]' [C%] )

( [C%] %i 🡺 [C%]' C%posi )

Para una formación de dos dimensiones, extrae una columna.

Para una formación de una dimensión, extrae el elemento de la posición #i

Si hay una matriz simbólica en la pila, llama a ˆMATRIX-COL

Si hay un arreglo de una dimensión en la pila, llama al comando FLASHPTR 3 5B

Equivale al comando COL- de User RPL.

46.1.18 Eigenvalores, Eigenvectores, Reducción.

**Direcc. Nombre Descripción**

37C006 ˆMATEGVL ( 2DMATRIX 🡺 1DMATRIX )

( 2DMATRIX 🡺 {} )

( 2DMATRIX 🡺 [%]/[C%] )

( [[%]] 🡺 [%]/[C%] )

( [[C%]] 🡺 [%]/[C%] )

Halla los eigenvalores de una formación.

Verifica que sea de dos dimensiones y cuadrada.

Siempre devuelve todos los valores propios cuando está activado el modo aproximado (flag 105 activado).

Siempre devuelve todos los valores propios cuando los elementos son todos reales o todos complejos.

Equivale al comando EGVL de User RPL.

01B003 FLASHPTR 003 01B ( [[%]] 🡺 [%]/[C%] )

( [[C%]] 🡺 [%]/[C%] )

Halla los eigenvalores de un arreglo.

Debe ser cuadrado y de dos dimensiones.

37F006 ˆMATEGV ( 2DMATRIX 🡺 2DMATRIX 1DMATRIX )

( 2DMATRIX 🡺 {} {} )

( 2DMATRIX 🡺 [[%]]/[[C%]] [%]/[C%] )

( [[%]] 🡺 [[%]]/[[C%]] [%]/[C%] )

( [[C%]] 🡺 [[%]]/[[C%]] [%]/[C%] )

Halla los eigenvectores y eigenvalores de una formación.

Verifica que sea de dos dimensiones y cuadrada.

Siempre devuelve todos los valores y vectores propios cuando está activado el modo aproximado (flag 105 activado).

Siempre devuelve todos los valores y vectores propios cuando los elementos son todos reales o todos complejos.

Los eigenvectores son devueltas en cada columna de la formación del nivel 2.

Equivale al comando EGV de User RPL.

01D003 FLASHPTR 003 01D ( [[%]] 🡺 [[%]]/[[C%]] [%]/[C%] )

( [[C%]] 🡺 [[%]]/[[C%]] [%]/[C%] )

Halla los eigenvectores y eigenvalores de un arreglo.

Debe ser cuadrado y de dos dimensiones.

**Direcc. Nombre Descripción**

228006 ˆPCAR ( 2DMATRIX 🡺 symb )

Halla el polinomio característico de una matriz cuadrada en terminos de la variable independiente del CAS.

Verifica que la matriz sea de dos dimensiones y cuadrada y que no contenga a la variable independiente del CAS.

Equivale al comando PCAR de User RPL cuando en la pila hay una formación.

14B007 FLASHPTR 007 14B ( 2DMAT 🡺 2DMAT )

Halla el polinomio mínimo de una matriz.

Retorna una matriz. En esta matriz el polinomio mínimo está en la primera fila cuyos elementos desde el primero hasta el penúltimo sean ceros.

Cuando la calculadora está en modo paso a paso, este comando muestra los pasos de reducción de filas.

Equivale al comando PMINI de User RPL.

37E006 ˆMADJ ( 2DMAT/[[]] 🡺 c Mˆ-1 P[M] P[lambda] )

Retorna inversa, matriz polinomial y polinomio característico.

380006 ˆJORDAN ( 2DMAT/[[]] 🡺 minP carP {egvec} 1DMAT/[]/{} )

( pmadj pcar 🡺 minP carP {egvec} 1DMAT/[]/{} )

Halla polinomio mínimo, polinomio característico, eigenvectores (etiquetados y dentro de una lista) y eigenvalores.

22D006 ˆFLAGJORDAN ( 2DMAT/[[]] 🡺 minP carP {egvect} 1DMAT/[]/{} )

Halla el polinomio mínimo, el polinomio característico, eigenvectores y eigenvalores de una matriz.

Equivale al comando JORDAN de User RPL.

381006 ˆQXA ( symb 1DMAT 🡺 2DMATRIX 1DMAT )

Convierte una forma cuadrada a matriz simétrica.

En el nivel 1 debe estar un vector con las variables como nombres globales.

224006 ˆFLAGQXA ( symb 1DMAT 🡺 2DMATRIX 1DMAT )

Convierte una forma cuadrada a matriz simétrica.

Hace varias verificaciones.

Equivale al comando QXA de User RPL.

382006 ˆAXQ ( 2DMATRIX 1DMAT 🡺 symb 1DMAT )

Convierte una matriz cuadrada a su forma cuadrática.

En el nivel 1 debe estar un vector con las variables como nombres globales.

225006 ˆFLAGAXQ ( 2DMATRIX 1DMAT 🡺 symb 1DMAT )

Convierte una matriz cuadrada a su forma cuadrática.

Hace varias verificaciones.

Equivale al comando AXQ de User RPL.

384006 ˆSYLVESTER ( A\_2DMAT 🡺 D\_1DMAT P\_2DMAT )

Para una matriz simétrica A, retorna D y P, donde D es una matriz diagonal y A=PTDP.

La matriz D es retornada como vector.

227006 ˆFLAGSYLVESTER ( A\_2DMAT 🡺 D\_1DMAT P\_2DMAT )

Para una matriz simétrica A, retorna D y P, donde D es una matriz diagonal y A=PTDP.

La matriz D es retornada como vector.

Verifica que la matriz sea de dos dimensiones y simétrica.

Equivale al comando SYLVESTER de User RPL.

**Direcc. Nombre Descripción**

383006 ˆGAUSS ( symb 1DMAT 🡺 D\_1DMAT P\_2DMAT symb' 1DMAT )

Para una forma cuadrática (cuya matriz es A), retorna D y P, donde D es una matriz diagonal y A=PTDP.

En el nivel 1 debe estar un vector con las variables como nombres globales.

226006 ˆFLAGGAUSS ( symb 1DMAT 🡺 D\_1DMAT P\_2DMAT symb' 1DMAT )

Para una forma cuadrática (cuya matriz es A), retorna D y P, donde D es una matriz diagonal y A=PTDP.

En el nivel 1 debe estar un vector con las variables como nombres globales.

Equivale al comando GAUSS de User RPL.

46.2 Ejemplos

**Ejemplo 1 Matrices**

**Aplicando un programa o comando a cada uno de los elementos de una matriz de dos dimensiones**

¿Cómo aplicar un programa a cada elemento de una matriz de dos dimensiones?

En User RPL, puedes aplicar un programa a cada elemento de una matriz con el comando MAP.

En System RPL, puedes usar el siguiente NULLNAME, el cual también funciona para una lista de listas:

\* Evalúa un programa o comando a los elementos de un 2DMATRIX

\* Entrada:

\* Nivel 2: Matriz de dos dimensiones

\* Nivel 1: Programa o comando que tome 1 argumentos y devuelva un objeto

\* Salida: Un 2DMATRIX con el mismo número de elementos que el original

\* NOTA: También funciona para una lista de listas

\* Para llamar al número de fila usar JINDEX@ #1+

\* Para llamar al número de columna usar INDEX@ #1+

\* Para llamar a la cantidad de filas usar JSTOP@

\* Para llamar a la cantidad de columnas usar ISTOP@

NULLNAME Prog1a1MAT2D ( 2DMAT prog1a1 -> 2DMAT' )

:: ( 2DMAT prog1a1 )

OVER ( 2DMAT prog1a1 2DMAT )

TYPE ( 2DMAT prog1a1 #tipo )

FLASHPTR 2LAMBIND ( 2DMAT )

INNERDUP ( 1DMAT1...1DMATf #f #f )

ZERO\_DO (DO)

( ... #n )

ROLL ( ... 1DMATi )

INNERDUP ( ... obi1...obic #c #c )

ZERO\_DO (DO)

ROLL ( ... ... obij )

2GETEVAL ( ... ... obij' )

ISTOP@ ( ... ... obij' #c )

LOOP

1GETLAM ( ... obi1'...obic' #c #tipo )

COMPN\_ ( ... 1DMATi' )

ISTOP@ ( ... 1DMATi' #n )

LOOP

( 1DMAT1'...1DMATf' #n )

1GETABND ( 1DMAT1'...1DMATf' #n #tipo )

COMPN\_ ( 2DMAT' )

;

**Ejemplo 2 Matrices**

**Aplicando un programa o comando a los elementos de dos matrices de dos dimensiones**

Puedes usar el siguiente NULLNAME, el cual también funciona para listas de listas:

\* Evalúa un programa o comando a los elementos de dos 2DMATRIX

\* Entrada:

\* Nivel 3: Matriz de dos dimensiones

\* Nivel 2: Otra matriz de dos dimensiones

\* Nivel 1: Programa o comando que tome 2 argumentos y devuelva un objeto

\* Salida: Un 2DMATRIX con el mismo número de elementos que los originales

\* NOTA: También funciona para listas de listas

\* Para llamar al número de fila usar JINDEX@ #1+

\* Para llamar al número de columna usar INDEX@ #1+

\* Para llamar a la cantidad de filas usar JSTOP@

\* Para llamar a la cantidad de columnas usar ISTOP@

NULLNAME Prog2a1MAT2D ( 2DMATA 2DMATB prog2a1 -> 2DMAT' )

:: ( 2DMATA 2DMATB prog2a1 )

OVER ( 2DMATA 2DMATB prog2a1 2DMATB )

TYPE ( 2DMATA 2DMATB prog2a1 #tipo )

FLASHPTR 2LAMBIND ( 2DMATA 2DMATB )

>R ( 2DMATA )

INNERDUP ( 1DMATA1...1DMATAf #f #f )

ZERO\_DO (DO)

( ... #n )

ROLL ( ... 1DMATAi )

RSWAP

'R

RSWAP ( ... 1DMATAi 1DMATBi )

>R ( ... 1DMATAi )

INNERDUP ( ... obAi1...obAic #c #c )

ZERO\_DO (DO)

( ... ... )

ROLL ( ... ... obAij )

RSWAP

'R ( ... ... obAij obBij )

RSWAP

2GETEVAL ( ... ... obij' )

ISTOP@ ( ... ... obij' #c )

LOOP

( ... obi1'...obic' #c )

RDROP ( ... obi1'...obic' #c )

1GETLAM ( ... obi1'...obic' #c #tipo )

COMPN\_ ( ... 1DMATi' )

ISTOP@ ( ... 1DMATi' #n )

LOOP

( 1DMAT1'...1DMATf' #n )

1GETABND ( 1DMAT1'...1DMATf' #n #tipo )

COMPN\_ ( 2DMAT' )

;

**Ejemplo 3 Matrices**

**Aplicando un programa o comando a los elementos de varias matrices de dos dimensiones**

Puedes usar el siguiente NULLNAME, el cual también funciona para listas de listas:

\* Evalúa un programa o comando a los elementos de varias matrices de 2 dimensiones

\* Entrada:

\* Niveles N+2,...,5,4,3: Matrices de 2 dimensiones, todas del mismo tamaño

\* Nivel 2: Un bint (#N) Indica el número de matrices

\* Nivel 1: Programa o comando que tome #N argumentos y devuelva un objeto

\* Salida:

\* Una matriz de dos dimensiones con el mismo nº de elementos que los originales

\*

\* NOTA: También funciona para listas de listas

\* Para llamar al número de fila usar JINDEX@ #1+

\* Para llamar al número de columna usar INDEX@ #1+

\* Para llamar a la cantidad de filas usar JSTOP@

\* Para llamar a la cantidad de columnas usar ISTOP@

\* NOTA: Este NULLNAME usa el NULLNAME ProgNa1Comp del capítulo 11

NULLNAME ProgNa1MAT2D ( 2DMAT1...2DMATN #N progNa1 -> 2DMAT' )

:: ( 2DMAT1...2DMATN #N progNa1 )

3PICK ( 2DMAT1...2DMATN #N progNa1 2DMATN )

LENCOMP ( 2DMAT1...2DMATN #N progNa1 #filas )

SWAP4PICK ( 2DMAT1...2DMATN #N #filas progNa1 2DMATN )

TYPE ( 2DMAT1...2DMATN #N #filas progNa1 #tipo )

' NULLLAM BINT4 NDUPN DOBIND

( 2DMAT1...2DMATN )

\* 4LAM: #N 3LAM: #filas 2LAM: PROG 1LAM: #tipo

4GETLAM ( 2DMAT1...2DMATN #N )

ZERO\_DO (DO)

( ... )

ISTOP-INDEX ( ... #N-i )

ROLL ( ... 2DMATi )

>R ( ... )

RSWAP ( ... )

LOOP

( )

3GETLAM ( #filas )

ZERO\_DO (DO)

( ... )

4GETLAM ( ... #N )

ZERO\_DO (DO)

( ... )

4GETLAM ( ... #N )

#2+ ( ... #N+2 )

RROLL\_ ( ... )

'R ( ... 1DMATij )

RSWAP ( ... 1DMATij )

LOOP

( ... 1DMAT1j...1DMATNj )

4GETLAM ( ... 1DMAT1j...1DMATNj #N )

#1+ ( ... 1DMAT1j...1DMATNj #N+1 )

RROLL\_ ( ... 1DMAT1j...1DMATNj )

4GETLAM ( ... 1DMAT1j...1DMATNj #N )

2GETLAM ( ... 1DMAT1j...1DMATNj #N progNa1 )

ProgNa1Comp ( ... 1DMATj' )

LOOP

( 1DMAT1'...1DMATN' )

3GETLAM ( 1DMAT1'...1DMATN' #N )

1GETABND ( 1DMAT1'...1DMATN' #N #tipo )

COMPN\_ ( 2DMAT' )

;

**Ejemplo 4 Matrices**

**Retornar un objeto de un compuesto.**

Puedes usar el siguiente NULLNAME:

\* Retorna un elemento de una matriz de 2 dimensiones ubicado

\* en la fila 'i' y en la columna 'j'

\* NOTA: También funciona para una lista de listas

NULLNAME GetFrom2DMATRIX ( 2DMATRIX #i #j -> ob )

:: ( 2DMATRIX #i #j )

UNROT ( #j 2DMATRIX #i )

NTHCOMPDROP ( #j 1DMATRIX )

SWAP ( 1DMATRIX #j )

NTHCOMPDROP ( ob )

;

**Ejemplo 5 Matrices**

**Reemplazar un objeto dentro de una matriz de dos dimensiones.**

Puedes usar el siguiente NULLNAME:

\* Reemplaza un objeto en una matriz de 2 dimensiones

\* Entrada:

\* NIVEL 4: Matriz de dos dimensiones

\* NIVEL 3: Fila donde se colocará el objeto

\* NIVEL 2: Columna donde se colocará el objeto

\* NIVEL 1: Objeto que entrará en la matriz.

\* Salida:

\* NIVEL 1: Matriz de dos dimensiones modificada

\* NOTA: También funciona para una lista de listas

NULLNAME ReplaceIn2DMATRIX ( 2DMATRIX #f #c ob -> 2DMATRIX' )

:: ( comp #f #c ob )

4PICK ( comp #f #c ob comp )

TYPE ( comp #f #c ob #tipo )

' NULLLAM BINT4 NDUPN DOBIND

( comp )

INNERDUP ( 1DMAT1...1DMATm #m #m )

4GETLAM ( 1DMAT1...1DMATm #m #m #f )

#- ( 1DMAT1...1DMATm #m #m-f )

#2+ROLL ( 1DMAT1...1DMATm #m 1DMATf )

INNERDUP ( 1DMAT1...1DMATm #m ob1...obn #n #n )

3GETLAM ( 1DMAT1...1DMATm #m ob1...obn #n #n #c )

#- ( 1DMAT1...1DMATm #m ob1...obn #n #n-c )

#2+ROLL ( 1DMAT1...1DMATm #m ob1...obn #n obc )

DROP ( 1DMAT1...1DMATm #m ob1...obn #n )

2GETLAM ( 1DMAT1...1DMATm #m ob1...obn #n ob )

OVER ( 1DMAT1...1DMATm #m ob1...obn #n ob #n )

3GETLAM ( 1DMAT1...1DMATm #m ob1...obn #n ob #n #c )

#- ( 1DMAT1...1DMATm #m ob1...obn #n ob #n-c )

#2+UNROLL ( 1DMAT1...1DMATm #m ob1...obn #n ob )

1GETLAM ( 1DMAT1...1DMATm #m ob1...obn #n #tipo )

COMPN\_ ( 1DMAT1...1DMATm #m 1DMATf' )

OVER ( 1DMAT1...1DMATm #m 1DMATf' #m )

4GETLAM ( 1DMAT1...1DMATm #m 1DMATf' #m #f )

#- ( 1DMAT1...1DMATm #m 1DMATf' #m-f )

#2+UNROLL ( 1DMAT1...1DMATm #m )

1GETABND ( 1DMAT1...1DMATm #m #tipo )

COMPN\_ ( 2DMAT' )

;

**Ejemplo 6 Matrices**

**Hallando la transpuesta de una lista de listas o de una matriz de dos dimensiones.**

Puedes usar el siguiente NULLNAME:

\* TRANSPUESTA PARA UNA LISTA DE LISTAS

\* TAMBIEN FUNCIONA EN UNA MATRIZ DE 2 DIMENSIONES

NULLNAME Transpuesta ( {{}} -> {{}} )

:: ( {{}} )

INNERCOMP ( meta )

DUP ( meta #n )

3PICK

TYPE ( meta #n #tipo )

ZEROZEROTWO DOBIND

( meta )

OVER

LENCOMP ( meta #c>f )

#1+\_ONE\_DO (DO)

( meta )

DUP#0\_DO (DO)

2GETLAM

#1+PICK

JINDEX@

NTHCOMPDROP

LOOP

2GETLAM

1GETLAM

COMPN\_

OVER#2+UNROL

LOOP

#1-

NDROP ( fila1...filan {} )

LENCOMP ( fila1...filan #n )

1GETABND ( fila1...filan #n #tipo )

COMPN\_ ( {{}} )

;

Capítulo 47  
Manipulación de Expresiones

Los comandos de este capítulo son usados para manipular expresiones, cuando éstas están en su forma de objetos simbólicos (Vea el capítulo 48 para ver los comandos que tratan con expresiones que estén en la forma de un meta). Hay comendos relacionados a la compresión y expansión, transformaciones trigonométricas y exponenciales y sustitución de valores en las expresiones.

47.1 Referencia

47.1.1 Operaciones Básicas y Aplicación de Funciones

**Direcc. Nombre Descripción**

125006 ˆx+ext ( ob2 ob1 🡺 ob2+ob1 )

Adición simbólica, prueba por infinito.

126006 ˆx-ext ( ob2 ob1 🡺 ob2-ob1 )

Resta simbólica, prueba por infinito.

127006 ˆx\*ext ( ob2 ob1 🡺 ob2\*ob1 )

Multiplicación simbólica, prueba por infinito.

129006 ˆx/ext ( ob2 ob1 🡺 ob2/ob1 )

División simbólica, prueba por infinito.

12B006 ˆxˆext ( ob exponente 🡺 obˆexponente )

Potenciación.

12C006 ˆEXPANDˆ ( x y 🡺 xˆy=exp[y\*ln[x]] )

Power with simplifications. If y is a fraction of integers, use XROOTˆ instead.

4FB006 ˆQneg ( ob 🡺 -ob )

Symbolic negation.

4FC006 ˆRNEGext ( ob 🡺 -ob )

Symbolic negation.

4FA006 ˆSWAPRNEG ( ob2 ob1 🡺 ob1 -ob2 )

Does SWAP then symbolic negation.

4FE006 ˆRREext ( ob 🡺 Re(ob) )

Symboloc real part.

4FD006 ˆSWAPRRE ( ob2 ob1 🡺 ob1 Re(ob2) )

SWAP, then RREext.

500006 ˆRIMext ( ob 🡺 Im(ob) )

Symbolic imaginary part.

4FF006 ˆSWAPRIM ( ob1 ob2 🡺 ob2 Im(ob1) )

SWAP, then RIMext.

501006 ˆxREext ( symb 🡺 symb' )

Complex real part. Expands only + - \* / ˆ.

503006 ˆxIMext ( symb 🡺 symb' )

Complex imaginary part. Expands only + - \* / ˆ.

505006 ˆRCONJext ( ob 🡺 Conj(ob) )

Symbolic complex conjugate.

50D006 ˆxABSext ( ob 🡺 abs(ob) )

Symbolic ABS function.

**Direcc. Nombre Descripción**

50A006 ˆRABSext ( ob 🡺 abs(ob) )

Internal ABS. Internal representation.

52A006 ˆxINVext ( ob 🡺 1/ob )

Symbolic inversion.

557006 ˆxSYMINV ( symb 🡺 1/symb )

Symbolic inversion.

553006 ˆxSQext ( symb 🡺 sq(symb) )

Symbolic square.

555006 ˆxSYMSQ ( symb 🡺 symbˆ2 )

51B006 ˆSXSQRext ( ob 🡺 sqrt(ob) )

Does not take care of the sign.

51C006 ˆXSQRext ( ob 🡺 sqrt(ob) )

Tries to return a positive square root if nocareflag is cleared.

52B006 ˆxvext ( ob 🡺 sqrt(ob) )

Symbolic square root, tests for 0 and 1.

552006 ˆxSYMSQRT ( symb 🡺 sqrt(symb) )

521006 ˆCKLN ( ob 🡺 ln(ob) )

Symbolic LN with special handling for fractions.

Does not use the internal representation.

522006 ˆxLNext ( ob 🡺 ln(ob) )

Symbolic LN, without fraction handling.

525006 ˆEXPANDLN ( ob 🡺 ln(ob) )

Symbolic LN using internal representation. Before switching to internal representation, test for ABS, 0 and 1 and, in real mode, test if ob=exp(x).

528006 ˆREALLN ( ob 🡺 ln(ob) )

Internal natural logarithm for a real argument.

526006 ˆCMPLXLN ( ob 🡺 ln(ob) )

Internal complex natural logarithm.

527006 ˆLNATANext ( ob 🡺 ln(ob) )

Internal natural logarithm for complex.

529006 ˆxEXPext ( y d n 🡺 exp(y\*n/d\*i\*π) )

Symbolic EXP, tests for 0, infinity and i\*k\*π/12

where k is an integer. Tests for d=1,2,3,4,6.

52C006 ˆxCOSext ( ob 🡺 cos(ob) )

Symbolic COS, tests for 0 and multiples of π/12.

Also tests if ob=acos(x) or ob=asin(x).

536006 ˆxSYMCOS ( ob 🡺 cos(ob) )

533006 ˆxACOSext ( ob 🡺 acos(ob) )

Symbolic ACOS. Tests for 0, infinity and tables.

53F006 ˆxSYMACOS ( ob 🡺 acos(ob) )

52D006 ˆxSINext ( ob 🡺 sin(ob) )

Symbolic SIN, tests for 0 and multiplies of π/12.

Also tests if ob=acos(x) or ob=asin(x).

538006 ˆxSYMSIN ( ob 🡺 sin(ob) )

532006 ˆxASINext ( ob 🡺 asin(ob) )

Symbolic ASIN. Tests for 0, infinity and tables.

53D006 ˆxSYMASIN ( ob 🡺 asin(ob) )

**Direcc. Nombre Descripción**

52E006 ˆxTANext ( ob 🡺 tan(ob) )

Symbolic TAN. Tests for 0 and multiplies of π/12.

Also tests if ob=atan(x).

53A006 ˆxSYMTAN ( ob 🡺 tan(ob) )

534006 ˆxATANext ( ob 🡺 atan(ob) )

Symbolic ATAN. Tests for 0, infinity and tables.

541006 ˆxSYMATAN ( ob 🡺 atan(ob) )

52F006 ˆxCOSHext ( ob 🡺 cosh(ob) )

Symbolic COSH. Tests for 0, infinity and acosh(x).

545006 ˆxSYMCOSH ( ob 🡺 cosh(ob) )

54E006 ˆxACOSHext ( symb 🡺 acosh(symb) )

Symbolic ACOSH.

550006 ˆxSYMACOSH ( symb 🡺 acosh(symb) )

530006 ˆxSINHext ( ob 🡺 sinh(ob) )

Symbolic SINH. Tests for 0, infinity and asinh(x).

543006 ˆxSYMSINH ( ob 🡺 sinh(ob) )

54B006 ˆxASINHext ( symb 🡺 symb' )

Symbolic ASINH.

54D006 ˆxSYMASINH ( symb 🡺 asinh(symb) )

531006 ˆxTANHext ( ob 🡺 tanh(ob) )

Symbolic TANH. Tests for 0 and atanh(x).

547006 ˆxSYMTANH ( ob 🡺 tanh(ob) )

Symbolic TANH.

548006 ˆxATANHext ( symb 🡺 symb' )

Symbolic ATANH.

54A006 ˆxSYMATANH ( ob 🡺 atanh(ob) )

55F006 ˆxSYMFLOOR ( symb 🡺 symb' )

561006 ˆxSYMCEIL ( symb 🡺 symb' )

563006 ˆxSYMIP ( symb 🡺 symb' )

565006 ˆxSYMFP ( symb 🡺 symb' )

567006 ˆxSYMXPON ( symb 🡺 symb' )

569006 ˆxSYMMANT ( symb 🡺 symb' )

56B006 ˆxSYMLNP1 ( symb 🡺 symb' )

56D006 ˆxSYMLOG ( symb 🡺 symb' )

56F006 ˆxSYMALOG ( symb 🡺 symb' )

571006 ˆxSYMEXPM1 ( symb 🡺 symb' )

572006 ˆfactorial ( symb 🡺 symb🡺 )

Symbolic factorial.

573006 ˆfacts ( symb 🡺 symb🡺 )

Symbolic factorial.

575006 ˆxSYMFACT ( symb 🡺 symb🡺 )

578006 ˆxSYMNOT ( symb 🡺 symb' )

128006 ˆx=ext ( ob2 ob1 🡺 ob2=ob1 )

47.1.2 Operadores Trigonométricos y Exponenciales

**Direcc. Nombre Descripción**

408006 ˆCOS2TAN/2 ( symb 🡺 symb' )

x 🡺 (1-(tan(x/2))ˆ2)/(1+(tan(x/2))ˆ2)

40B006 ˆSIN2TAN/2 ( symb 🡺 symb' )

x 🡺 2 tan(x/2)/(1+(tan(x/2))ˆ2)

40E006 ˆTAN2TAN/2 ( symb 🡺 symb' )

x 🡺 2 tan(x/2)/(1-(tan(x/2))ˆ2)

412006 ˆCOS2TAN ( symb 🡺 symb2 )

x 🡺 1/sqrt(1+(tan(x))ˆ2)

414006 ˆSIN2TAN ( symb 🡺 symb' )

x 🡺 tan(x)/sqrt(1+(tan(x))ˆ2)

Direcc. Nombre Descripción

41A006 ˆLNP12LN ( symb 🡺 symb' )

x 🡺 ln(x+1)

41B006 ˆLOG2LN ( symb 🡺 symb' )

x 🡺 log(x)

41C006 ˆALOG2EXP ( symb 🡺 symb' )

x 🡺 alog(x)

41D006 ˆEXPM2EXP ( symb 🡺 symb' )

x 🡺 exp(x)-1

41E006 ˆSQRT2LNEXP ( symb 🡺 symb' )

x 🡺 exp(ln(x)/2)

41F006 ˆsqrt2lnexp ( meta 🡺 meta' )

x 🡺 exp(ln(x)/2)

420006 ˆTAN2EXP ( symb 🡺 symb' )

x 🡺 (exp(i2x)-1)/(i\*(exp(i2x)+1))

422006 ˆASIN2LN ( symb 🡺 symb' )

x 🡺 = i\*ln(x+sqrt(xˆ2-1))+pi/2.

424006 ˆACOS2LN ( symb 🡺 symb' )

x 🡺 ln(x+sqrt(xˆ2-1))/i

427006 ˆTAN2SC ( symb 🡺 symb' )

x 🡺 sin(x)/cos(x)

42A006 ˆSIN2TC ( symb 🡺 symb' )

x 🡺 cos(x)\*tan(x)

42C006 ˆCOS2ext ( symb 🡺 symb' )

x 🡺 sqrt(1-(sin(x))ˆ2).

42E006 ˆSIN2ext ( symb 🡺 symb' )

x 🡺 sqrt(1-(cos(x))ˆ2).

431006 ˆATAN2ASIN ( symb 🡺 symb' )

x 🡺 asin(x/sqrt(xˆ2+1))

434006 ˆASIN2ATAN ( symb 🡺 symb' )

x 🡺 atan(x/sqrt(1-xˆ2))

437006 ˆASIN2ACOS ( symb 🡺 symb' )

x 🡺 \_/2-acos(x)

43C006 ˆACOS2ASIN ( symb 🡺 symb' )

x 🡺 \_/2-asin(x)

43D006 ˆATAN2Lnext ( symb 🡺 symb' )

x 🡺 i/2\*ln((i+x)/(i-x))

440006 ˆTAN2SC2 ( symb 🡺 symb' )

x 🡺 (1-cos(2x))/sin(2x)

442006 ˆTAN2CS2 ( symb 🡺 symb' )

x 🡺 sin(2x)/(1+cos(2x))

444006 ˆSIN2EXPext ( symb 🡺 symb' )

x 🡺 (eˆ(i\*x)-1/eˆ(i\*x))/(2i)

446006 ˆCOS2EXPext ( symb 🡺 symb' )

x 🡺 (eˆ(i\*x)+1/eˆ(i\*x))/2

448006 ˆSINH2EXPext ( symb 🡺 symb' )

x 🡺 (eˆx-1/eˆx)/2

44A006 ˆCOSH2EXPext ( symb 🡺 symb' )

x 🡺 (eˆx+1/eˆx)/2

44C006 ˆTANH2EXPext ( symb 🡺 symb' )

x 🡺 (eˆ2x-1)/(eˆ2x+1)

44E006 ˆASINH2Lnext ( symb 🡺 symb' )

x 🡺 ln(x+sqrt(xˆ2+1))

450006 ˆACOSH2Lnext ( symb 🡺 symb' )

x 🡺 ln(x+sqrt(xˆ2-1))

Direcc. Nombre Descripción

452006 ˆATANH2Lnext ( symb 🡺 symb' )

x 🡺 ln((1+x)/(1-x))/2

454006 ˆXROOT2ext ( symb1 symb2 🡺 symb' )

x y 🡺 exp(ln(y)/x)

45A006 ˆLN2ATAN ( symb 🡺 symb' )

x 🡺 ln(x)

47.1.3 Simplificación, Evaluación y Sustitución

**Direcc. Nombre Descripción**

45B006 ˆVAR=LIST ( idnt {} 🡺 {}' )

Replaces all elements of the initial list by idnt=element.

464006 ˆSYMBEXEC ( ob symb 🡺 ob' )

If symb is an equation, executes the corresponding change of variables in ob, otherwise tries to find symb so that ob is zero. Note that change of variable works for change of user functions.

465006 ˆMEVALext ( ob {} {}' 🡺 ob' )

Replaces all occurrances of an element of list2 by the corresponding element of list1 in ob. Looks in ob from outer to inner expressions. list2 and list1 may contain secondaries. If vxxlflag is set SIGN var are leaved unchanged.

466006 ˆCASNUMEVAL ( symb list1 list2 🡺 symb' )

Evaluation of a symbolic. The lists' formats are

list1={idnt/lam1... idnt\_n/lam\_n}

list2={value1...value\_n}

The idnt's/lam's in list1 are not evaluated before replacing value1...value\_n.

467006 ˆCASCOMPEVAL ( symb 🡺 symb' )

Evaluation of a symbolic.

468006 ˆREPLACE2BY1 ( symb idnt a 🡺 symb' )

Evaluation of a symbolic replacing an idnt by a value; for example evaluation of F(X) for X=1/2)

469006 ˆNR\_REPLACE ( symb idnt a 🡺 symb' )

Like REPLACE2BY1 but prevents evaluation of INT.

46B006 ˆCASCRUNCH ( ob 🡺 % )

Like CRUNCH but in approximate mode.

46C006 ˆAPPROXCOMPEVAL ( symb 🡺 symb' )

Like CASCOMPEVAL but in approximate mode.

11A007 ˆALGCASCOMPEVAL ( expr 🡺 expr )

297006 ˆSLVARext ( Lvar 🡺 Lvar' )

Simplifies all elements of the list that are supposed to be variables.

298006 ˆSIMPLIFY ( symb 🡺 symb' )

Simplifies one object like EVAL.

**Direcc. Nombre Descripción**

299006 ˆSIMP1ext ( symb 🡺 symb' )

Simplifies one object like EXPAND. Object must be a symbolic, a real or a complex number.

29A006 ˆSYMEXPAN ( symb 🡺 symb' )

Simplifies one object like EXPAN. Object must be symb/real/cmplx.

29B006 ˆSIMPVAR ( ob 🡺 ob' )

Simplifies variable.

2A0006 ˆSIMPSYMBS ( inf sup fcn var 🡺 int(inf,sup,fcn,var) )

2A2006 ˆSIMPUSERFCN ( ob1..obn #n ob 🡺 id[] )

Simplification of user functions. Tests for derivative of user functions. Ob must be an id, a symbolic, a secondary or a romptr.

2A3006 ˆEVALUSERFCN ( V1..Vn #n fcn 🡺 f[] )

Evaluates a user function with stack checking.

2A4006 ˆSIMP| ( ob list 🡺 ob' )

Executes the WHERE operator.

2A9006 ˆSIMPext ( ob1 ob2 🡺 ob1' ob2' )

Simplifies two objects in internal representation.

Checks that o2 is not a complex or an irrquad because decomposition of the corresponding fraction with larg would generate a "Try to recover Memory".

2AD006 ˆSIMPGCDext ( o1 o2 gcd 🡺 o1/gcd o2/gcd )

Divides o1 and o2 by gcd.

2AE006 ˆSIMP3ext ( a b 🡺 g a'' b'' )

Calculates g = gcd(a,b) and a''=a/g and b''=b/g.

2B9006 ˆTSIMP2ext ( symb 🡺 symb )

Transcendental simplifications. Converts only sqrt ˆ and XROOT to EXP/LN. LN are returned as -1/INV[-LN[]] for use by SERIES.

2BA006 ˆTSIMPext ( symb 🡺 symb )

Transcendental simplifications. Convert transcendental functions to EXP and LN.

2BB006 ˆTSIMP3ext ( symb 🡺 symb )

47.1.4 Colección y Expansión

**Direcc. Nombre Descripción**

26E006 ˆCOLCext ( symb 🡺 symb' )

Factorization with respect to the current variable of symb and factorization of the integer content of symb.

2FE006 ˆTCOLLECT ( symb 🡺 symb' )

Performs trigonometric linearization and then collects sines and cosines of the same angle.

2FF006 ˆSIGMAEXPext ( symb 🡺 symb' )

Conversion to exp and ln with exponential linearization.

300006 ˆLINEXPext ( symb 🡺 Meta )

Meta = arg\_exp1 coef1 ... arg\_expn coefn #2n.

301006 ˆSIGMAEXP2ext ( Meta 🡺 symb )

Back conversion from arg\_exp/coef\_meta to symbolic.

303006 ˆSINEXPA ( symb 🡺 symb' )

Expands SIN.

**Direcc. Nombre Descripción**

316006 ˆLNEXPA ( symb 🡺 symb' )

Expands LN.

31C006 ˆMTRIG2SYMB ( Meta 🡺 symb )

Back conversion of trig-meta to symbolic.

309006 ˆCOSEXPA ( symb 🡺 symb' )

Expands COS.

30F006 ˆEXPEXPA ( symb 🡺 symb' )

Expands EXP.

31B006 ˆLINEXPA ( symb 🡺 Meta )

Alternates trig operator and coefficient.

31D006 ˆLNCOLCext ( symb 🡺 symb' )

Collects logarithms.

31F006 ˆTEXPAext ( symb 🡺 symb )

Main transcendental expansion program.

240006 ˆEXLR ( 'a=b' 🡺 a b )

( ob 🡺 X ob )

Internal equation splitter.

47.1.5 Transformaciones Trigonométricas

**Direcc. Nombre Descripción**

407006 ˆHALFTAN ( symb 🡺 symb' )

Converts trigonometric functions to TAN of the half angle.

411006 ˆTRIGTAN ( symb 🡺 symb' )

Convert sin and cos to tan of the same angle.

416006 ˆTRIGext ( symb 🡺 symb' )

Applies sinˆ2+cosˆ2=1 to simplify trigonometric expressions. If flag -116 is set, tries to keep only sin, else only cos.

417006 ˆHYP2EXPext ( symb 🡺 symb' )

Converts hyperbolic functions to exp and ln.

Converts XROOT and ˆ to exp and ln.

418006 ˆEXPLNext ( symb 🡺 symb' )

Converts all transcendental functions to exp and ln.

419006 ˆSERIESEXPLN ( symb 🡺 symb' )

Converts sqrt, ˆ and XROOT to EXP/LN.

426006 ˆTAN2Scext ( symb 🡺 symb' )

Converts tan to sin/cos.

429006 ˆSIN2Tcext ( symb 🡺 symb' )

Converts sin to cos\*tan.

430006 ˆATAN2Sext ( symb 🡺 symb' )

Converts ATAN to ASIN using asin(x)=atan(x/sqrt(1-xˆ2)).

433006 ˆASIN2Text ( symb 🡺 symb' )

Converts ASIN to ATAN using asin(x)=atan(x/sqrt(1-xˆ2)).

436006 ˆASIN2Cext ( symb 🡺 symb' )

Converts ASIN to ACOS using asin(x)=pi/2-acos(x).

43A006 ˆACOS2Sext ( symb 🡺 symb' )

Converts ACOS to ASIN using acos(x)=pi/2-asin(x).

43F006 ˆTAN2SC2ext ( symb 🡺 symb' )

Converts TAN to SIN/COS of the double angle.

If flag -116 is set calls TAN2SC2, else TAN2CS2.

456006 ˆLN2ext ( symb 🡺 symb' )

If symb contains x, returns -1/inv(-ln(x)), else ln(x).

Used by SERIES.

47.1.6 División, GCD y LCM

**Direcc. Nombre Descripción**

3E8006 ˆPSEUDODIV ( Q2 Q1 🡺 a Q2\*a/Q1 Q2\*a/Q1 )

3EA006 ˆBESTDIV2 ( o2 o1 🡺 quo mod )

3EC006 ˆQUOText ( o2 o1 🡺 o2 div o1 )

Euclidean quotient of 2 objets (works even if o2 mod o1=0).

3ED006 ˆNEWDIVext ( ob2 ob1 🡺 quo mod )

Euclidean division, ob2 and ob1 may be fractions of returns a fraction of Q.

3F3006 ˆQUOTOBJext ( a\_a-1...a0 bb\_1...b0 #b #a flag 🡺 r q )

SRPL Euclidean division: step 2 computes the remainder r only if flag is TRUE.

3F4006 ˆDIVISIBLE? ( a b 🡺 a/b T )

( a b 🡺 ob F )

Returns TRUE and quotient if b divides a, otherwise returns FALSE.

3F5006 ˆQDiv? ( a b 🡺 a/b T )

( a b 🡺 F )

Returns TRUE and quotient if b divides a, otherwise returns FALSE.

3F6006 ˆFastDiv? ( P Q 🡺 P/Q PmodQ T )

Euclidean division. Assumes P and Q have integer or Gaussian integer coefficient. Returns FALSE in complex mode or if sparse short division fails.

3F7006 ˆPOTENCEext ( z1 z2 🡺 q r )

Step by step Euclidean division for small integers.

2A5006 ˆDENOLCMext ( list 🡺 ob )

Calculates the LCM of the denominator of the elements of the list. If input is not a list, returns the denominator of the object.

2A6006 ˆMETADENOLCM ( Meta 🡺 ob )

Calculates LCM of the denominators of the elements of Meta.

2B1006 ˆLPGCDext ( {} 🡺 {} ob )

Calculates the GCD of all the elements in the list. The algorithm is far from optimal.

2B2006 ˆSLOWGCDext ( c 1 A B 🡺 c\* gcd(A,B) )

Euclidean algorithm for polynomial GCD. Used if A or B contains irrquads. c is the GCD of the contents of the original polynomials returned after failure of GCDHEUext.

2B3006 ˆQGcd ( ob2 ob1 🡺 gcd )

Generic internal GCD.

( LAM2: GCDext ob1, ob2 🡺 pgcd ).

Capítulo 48  
Manipulación de Metas Simbólicos

Este capítulo contiene comandos que modifican metas que representan a objetos simbólicos.

Estos comandos son usados para modificar las expresiones o para operar con éstas.

48.1 Referencia

48.1.1 Comandos Básicos para Manipular Expresiones

**Direcc. Nombre Descripción**

157006 ˆSYMBINCOMP ( symb 🡺 ob1 .. obN #n )

( ob 🡺 ob #1 )

( {} 🡺 {} #1 )

Descompone un objeto simbólico en un objeto meta. Otros objetos son convertidos en metas de 1 objeto poniendo #1 en la pila.

386006 ˆm-1&m+1 ( meta 🡺 meta&1&- meta&1&+ )

Crea dos copias del meta.

A uno le agrega el entero 1 y el comando x-.

Al otro le agrega el entero 1 y el comando x+.

388006 ˆ1&meta ( Meta 🡺 1&Meta )

Agrega el entero 1 al inicio del meta.

387006 ˆmeta1/meta ( meta 🡺 meta 1&meta&/ )

Duplica el meta e invierte la expression que el meta representa.

389006 ˆmeta/2 ( Meta 🡺 Meta&2&/ )

Divide la expresión entre el entero 2.

38A006 ˆaddt2 ( Meta 🡺 Meta&2 )

Agrega el entero 2 al final del meta.

38B006 ˆaddt/ ( Meta 🡺 Meta&/ )

Agrega símbolo de división al final del meta.

103001 FLASHPTR 001 103 ( Meta 🡺 Meta&\* )

Agrega símbolo de multiplicación al final del meta.

38C006 ˆmeta2\* ( Meta 🡺 2&Meta&\* )

Multiplica la expresión por el entero 2.

390006 ˆmeta-1 ( Meta 🡺 Meta&1&- )

Resta el entero 1 a la expresión.

398006 ˆaddtˆ ( Meta 🡺 Meta&ˆ )

Agrega símbolo de potenciación al final del meta.

39C006 ˆtop&addt\* ( meta #0 🡺 meta )

( meta1 meta2 🡺 meta1\*meta2 )

Multiplica dos expresiones.

Junta los metas y agrega el comando x\* al final.

39D006 ˆtop&addt/ ( meta #0 🡺 meta )

( meta1 meta2 🡺 meta1/meta2 )

Divide dos expresiones.

Junta los metas y agrega el comando x/ al final.

39E006 ˆaddti ( meta 🡺 meta&i )

Agrega i (la unidad imaginaria) al final del meta.

48.1.2 Operaciones Básicas y Aplicación de Funciones

**Direcc. Nombre Descripción**

459006 ˆmetai\* ( meta 🡺 i&meta&\* )

Multiplica la expresión por i.

38D006 ˆmeta1-sq ( Meta 🡺 Meta' )

Cambia x hacia 1-xˆ2, donde x es la expresión original.

38E006 ˆmetasq+1 ( Meta 🡺 Meta' )

Cambia x hacia xˆ2+1, donde x es la expresión original.

38F006 ˆmetasq-1 ( Meta 🡺 Meta&SQ&1&- )

Cambia x hacia xˆ2-1, donde x es la expresión original.

393006 ˆmetaadd ( Meta1 Meta2 🡺 Meta1+Meta2 )

Adds 2 meta objects with trivial simplifications.

metaadd checks for Meta1/2=Z0 ONE.

3AB006 ˆMetaAdd ( Meta2 Meta1 🡺 Meta2+Meta1 )

Adds 2 meta objects with trivial simplifications.

Checks for infinities then call metaadd.

1CE006 ˆckaddt+ ( Meta1 Meta2 🡺 Meta1+Meta2 )

Adds 2 meta objects with trivial simplifications.

394006 ˆmetasub ( Meta1 Meta2 🡺 Meta1+Meta2 )

Subtracts 2 meta objects with trivial simplifications.

metasub checks for Meta1/2=Z0 ONE.

3AD006 ˆMetaSub ( Meta2 Meta1 🡺 Meta2-Meta1 )

Subtracts 2 meta objects with trivial simplifications.

Checks for infinities then call metasub.

1CF006 ˆckaddt- ( Meta1 Meta2 🡺 Meta1+Meta2 )

Subtracts 2 meta objects with trivial simplifications.

395006 ˆmetamult ( Meta1 Meta2 🡺 Meta1\*Meta2 )

Multiplies 2 meta objects with trivial simplifications.

Checks for meta1, meta2= Z0 or Z1, checks for xNEG.

3AF006 ˆMetaMul ( Meta2 Meta1 🡺 Meta2\*Meta1 )

Multiplies 2 meta objects with trivial simplifications.

Checks for infinities/0 then call metamult.

1CD006 ˆckaddt\* ( Meta1 Meta2 🡺 Meta1\*Meta2 )

Multiplies 2 meta objects with trivial simplifications.

396006 ˆmetadiv ( Meta2 Meta1 🡺 Meta2/Meta1 )

Divides 2 meta objects with trivial simplifications.

Checks for infinities and 0, meta2 =1 or Z-1, checks for xNEG.

3B1006 ˆMetaDiv ( Meta2 Meta1 🡺 Meta2/Meta1 )

Divide 2 meta objects with trivial simplifications.

Checks for infinities and 0 then call metadiv.

3F1006 ˆDIVMETAOBJ ( o1...on #n ob 🡺 {o1/ob...on/ob} )

Division of all elements of a meta by ob. Tests if o=1.

397006 ˆmetaˆ ( Meta ob 🡺 Meta&ob&ˆ )

Elevates expression to a power. If ob=1, just returns the expression. Tests for present of xNEG in the end of meta for integral powers.

399006 ˆmetapow ( Meta2 Meta1 🡺 Meta2ˆMeta1 )

Elevates expression to a power (any other expression).

If length of Meta1 is ONE, calls metaˆ.

3B5006 ˆMetaPow ( Meta2 Meta1 🡺 Meta2ˆMeta1 )

Power. Checks for infinities then calls metapow.

39B006 ˆmetaxroot ( Meta2 Meta1 🡺 Meta2&XROOT&Meta1 )

Root of expression.

**Direcc. Nombre Descripción**

3B9006 ˆmetaneg ( meta 🡺 meta )

Checks only for meta finishing by xNEG.

3BA006 ˆmetackneg ( meta 🡺 meta )

Like metaneg but checks for meta=ob ONE.

3B7006 ˆMetaNeg ( Meta 🡺 Meta )

Negates meta. Only checks for metas finishing by xNEG.

502006 ˆxSYMRE ( meta 🡺 meta' )

Meta complex real part. Expands only + - \* / ˆ.

504006 ˆxSYMIM ( meta 🡺 meta' )

Meta complex imaginary part. Expands only + - \* / ˆ.

50E006 ˆaddtABS ( Meta 🡺 Meta' )

Meta ABS. Does a CRUNCH first to find sign.

510006 ˆaddtABSEXACT ( Meta 🡺 Meta' )

Meta ABS. No crunch, sign is only found using exact methods.

511006 ˆaddtSIGN ( Meta 🡺 Meta' )

Meta SIGN.

513006 ˆaddtARG ( Meta 🡺 Meta' )

Meta ARG.

12D006 ˆaddtXROOT ( Meta2 Meta1 🡺 Meta' )

Meta XROOT. XROOT(o2,o1) is o1ˆ[1/o2], compared to o2ˆo1.

12F006 ˆaddtMIN ( Meta2 Meta1 🡺 Meta' )

Meta MIN.

131006 ˆaddtMAX ( Meta2 Meta1 🡺 Meta' )

Meta MAX.

133006 ˆaddt< ( Meta2 Meta1 🡺 Meta' )

Meta <.

135006 ˆaddt<= ( Meta2 Meta1 🡺 Meta' )

Meta <=.

137006 ˆaddt> ( Meta2 Meta1 🡺 Meta' )

Meta >.

139006 ˆaddt>= ( Meta2 Meta1 🡺 Meta' )

Meta >=.

13B006 ˆaddt== ( Meta2 Meta1 🡺 Meta' )

Meta ==.

13D006 ˆaddt!= ( Meta2 Meta1 🡺 Meta' )

Meta ≠.

13F006 ˆaddt% ( Meta2 Meta1 🡺 Meta' )

Meta %.

141006 ˆaddt%CH ( Meta2 Meta1 🡺 Meta' )

Meta %CH. Meta2\*(1+Meta'/100)=Meta1.

143006 ˆaddt%T ( Meta2 Meta1 🡺 Meta' )

Meta %T.

145006 ˆaddtMOD ( Meta2 Meta1 🡺 Meta' )

Meta MOD.

147006 ˆaddtTRNC ( Meta2 Meta1 🡺 Meta' )

Meta TRNC.

149006 ˆaddtRND ( Meta2 Meta1 🡺 Meta' )

Meta RND.

14B006 ˆaddtCOMB ( Meta2 Meta1 🡺 Meta' )

Meta COMB.

14D006 ˆaddtPERM ( Meta2 Meta1 🡺 Meta' )

Meta PERM.

**Direcc. Nombre Descripción**

14F006 ˆaddtOR ( Meta2 Meta1 🡺 Meta' )

Meta OR.

151006 ˆaddtAND ( Meta2 Meta1 🡺 Meta' )

Meta AND.

153006 ˆaddtXOR ( Meta2 Meta1 🡺 Meta' )

Meta XOR.

506006 ˆaddtCONJ ( meta 🡺 meta' )

Meta complex conjugate.

523006 ˆaddtLN ( Meta 🡺 Meta' )

Meta LN.

535006 ˆaddtCOS ( Meta 🡺 Meta' )

Meta COS.

537006 ˆaddtSIN ( Meta 🡺 Meta' )

Meta SIN.

539006 ˆaddtTAN ( Meta 🡺 Meta' )

Meta TAN.

53B006 ˆaddtSINACOS ( meta 🡺 meta' )

If meta stands for x, meta' stands for sqrt[1-xˆ2].

53C006 ˆaddtASIN ( Meta 🡺 Meta' )

Meta ASIN.

53E006 ˆaddtACOS ( Meta 🡺 Meta' )

Meta ACOS.

540006 ˆaddtATAN ( Meta 🡺 Meta' )

Meta ATAN.

542006 ˆaddtSINH ( Meta 🡺 Meta' )

Meta SINH.

544006 ˆaddtCOSH ( Meta 🡺 Meta' )

Meta COSH.

546006 ˆaddtTANH ( Meta 🡺 Meta' )

Meta TANH.

549006 ˆaddtATANH ( Meta 🡺 Meta' )

Meta ATANH.

54C006 ˆaddtASINH ( Meta 🡺 Meta' )

Meta ASINH.

54F006 ˆaddtACOSH ( Meta 🡺 Meta' )

Meta ACOSH.

551006 ˆaddtSQRT ( Meta 🡺 Meta' )

Meta SQRT.

554006 ˆaddtSQ ( Meta 🡺 Meta' )

Meta SQ.

556006 ˆaddtINV ( Meta 🡺 Meta' )

Meta INV.

558006 ˆaddtEXP ( Meta 🡺 Meta' )

Meta EXP. Does not apply EXP[-..]=1/EXP[..].

559006 ˆxSYMEXP ( Meta 🡺 Meta' )

Meta EXP. Applies EXP[-..]=1/EXP[..].

55A006 ˆaddtD->R ( Meta 🡺 Meta' )

Meta D🡺R.

55C006 ˆaddtR->D ( Meta 🡺 Meta' )

Meta R🡺D.

55E006 ˆaddtFLOOR ( Meta 🡺 Meta' )

Meta FLOOR.

**Direcc. Nombre Descripción**

560006 ˆaddtCEIL ( Meta 🡺 Meta' )

Meta CEIL.

562006 ˆaddtIP ( Meta 🡺 Meta' )

Meta IP.

564006 ˆaddtFP ( Meta 🡺 Meta' )

Meta FP.

566006 ˆaddtXPON ( Meta 🡺 Meta' )

Meta XPON.

568006 ˆaddtMANT ( Meta 🡺 Meta' )

Meta MANT.

56A006 ˆaddtLNP1 ( meta 🡺 meta )

Meta LNP1.

56C006 ˆaddtLOG ( meta 🡺 meta )

Meta LOG.

56E006 ˆaddtALOG ( meta 🡺 meta )

Meta ALOG.

570006 ˆaddtEXPM ( meta 🡺 meta )

Meta EXPM.

574006 ˆaddtFACT ( Meta 🡺 Meta' )

Meta FACT.

577006 ˆaddtNOT ( Meta 🡺 Meta' )

Meta NOT.

48.1.3 Operadores Trigonométricos y Exponenciales

**Direcc. Nombre Descripción**

409006 ˆcos2tan/2 ( meta 🡺 meta' )

x 🡺 (1-(tan(x/2))ˆ2)/(1+(tan(x/2))ˆ2)

40A006 ˆ1-xˆ2/1+xˆ2 ( meta 🡺 meta' )

x 🡺 (1-xˆ2)/(1+xˆ2)

40C006 ˆsin2tan/2 ( meta 🡺 meta' )

x 🡺 2 tan(x/2)/(1+(tan(x/2))ˆ2)

40D006 ˆ2x/1+xˆ2 ( meta 🡺 meta' )

x 🡺 2x/(1+xˆ2)

40F006 ˆtan2tan/2 ( meta 🡺 meta' )

x 🡺 2 tan(x/2)/(1-(tan(x/2))ˆ2)

410006 ˆaddtTAN/2 ( meta 🡺 meta' )

x 🡺 tan(x/2)

413006 ˆcos2tan ( meta 🡺 meta' )

x 🡺 1/sqrt(1+(tan(x))ˆ2)

415006 ˆsin2tan ( meta 🡺 meta' )

x 🡺 tan(x)/sqrt(1+(tan(x))ˆ2)

421006 ˆtan2exp ( meta 🡺 meta' )

x 🡺 (exp(i2x)-1)/(i\*(exp(i2x)+1))

423006 ˆasin2ln ( meta 🡺 meta' )

x 🡺 = i\*ln(x+sqrt(xˆ2-1))+\_/2.

425006 ˆacos2ln ( meta 🡺 meta' )

x 🡺 ln(x+sqrt(xˆ2-1))/i

428006 ˆsin/cos ( meta 🡺 meta' )

x 🡺 sin(x)/cos(x)

42B006 ˆcos\*tan ( meta 🡺 meta' )

x 🡺 cos(x)\*tan(x)

**Direcc. Nombre Descripción**

42D006 ˆsqrt1-sinˆ2 ( meta 🡺 meta' )

x 🡺 sqrt(1-(sin(x))ˆ2).

42F006 ˆsqrt1-cosˆ2 ( meta 🡺 meta' )

x 🡺 sqrt(1-(cos(x))ˆ2).

432006 ˆatan2asin ( meta 🡺 meta' )

x 🡺 asin(x/sqrt(xˆ2+1))

435006 ˆasin2atan ( meta 🡺 meta' )

x 🡺 atan(x/sqrt(1-xˆ2))

438006 ˆpi/2-acos ( meta 🡺 meta' )

x 🡺 π /2-acos(x)

439006 ˆpi/2-meta ( meta 🡺 meta' )

x 🡺 π /2-x

43B006 ˆpi/2-asin ( meta 🡺 meta' )

x 🡺 π/2-asin(x)

43E006 ˆatan2ln ( meta 🡺 meta' )

x 🡺 i/2\*ln((i+x)/(i-x))

441006 ˆ2\*1-cos/sin ( meta 🡺 meta' )

x 🡺 (1-cos(2x))/sin(2x)

443006 ˆ2\*sin/1+cos ( meta 🡺 meta' )

x 🡺 sin(2x)/(1+cos(2x))

445006 ˆsin2exp ( meta 🡺 meta' )

x 🡺 (eˆ(i\*x)-1/eˆ(i\*x))/(2i)

447006 ˆcos2exp ( meta 🡺 meta' )

x 🡺 (eˆ(i\*x)+1/eˆ(i\*x))/2

449006 ˆsinh2exp ( meta 🡺 meta' )

x 🡺 (eˆx-1/eˆx)/2

44B006 ˆcosh2exp ( meta 🡺 meta' )

x 🡺 (eˆx+1/eˆx)/2

44D006 ˆtanh2exp ( meta 🡺 meta' )

x 🡺 (eˆ2x-1)/(eˆ2x+1)

44F006 ˆasinh2ln ( meta 🡺 meta' )

x 🡺 ln(x+sqrt(xˆ2+1))

451006 ˆacosh2ln ( meta 🡺 meta' )

x 🡺 ln(x+sqrt(xˆ2-1))

453006 ˆatanh2ln ( meta 🡺 meta' )

x 🡺 ln((1+x)/(1-x))/2

455006 ˆxroot2expln ( meta1 meta2 🡺 meta' )

x y 🡺 exp(ln(y)/x)

458006 ˆexp2sincos ( meta 🡺 meta' )

Returns EXP of meta as

EXP[RE]\*[COS+i\*SIN].

48.1.4 Infinitos e Indefinidos

**Direcc. Nombre Descripción**

3A1006 ˆ1metaundef# ( meta 🡺 meta # )

Test para la presencia de indefinido en el meta.

# es la posición de indefinido.

Retorna #0 si no está el indefinido en el meta.

3A0006 ˆ2metaundef# ( meta2 meta1 🡺 meta2 meta1 # )

Test para la presencia de indefinido en alguno de los metas.

# es la posición de indefinido.

Retorna #0 si no está el indefinido en ningún meta.

**Direcc. Nombre Descripción**

3A2006 ˆmetaundef ( 🡺 meta )

Retorna el meta indefinido.

3A4006 ˆ1metainf# ( meta 🡺 meta # )

Encuentra la posición de infinito en un meta.

Retorna #0 para metas que no tienen infinito.

Metas de longitud mayor a 2 son considerados como metas finitos.

Retorna #1 para +infinito, -infinito o infinito sin signo.

3A3006 ˆ2metainf# ( meta2 meta1 🡺 meta2 meta1 # )

Encuentra la posición de infinito en alguno de los metas.

Retorna #0 para metas que no tienen infinito.

Metas de longitud mayor a 2 son considerados como metas finitos.

3A5006 ˆmetainftype ( meta 🡺 # )

Retorna el tipo de infinito:

#1 para +infinito,

#2 para –infinito,

#0 para infinito sin signo.

3A6006 ˆunsignedinf ( 🡺 meta )

Retorna infinito sin signo.

3A7006 ˆplusinf ( 🡺 meta )

Retorna +infinito.

3A8006 ˆNDROPplusinf ( ob1..obn #n 🡺 meta )

Reemplaza meta por +infinito.

3A9006 ˆminusinf ( 🡺 meta )

Retorna -infinito.

3AA006 ˆNDROPminusinf ( ob1..obn 🡺 meta )

Reemplaza meta por -infinito.

48.1.5 Expansión y Simplificación

**Direcc. Nombre Descripción**

3BB006 ˆmetasimp ( Meta 🡺 Meta )

Simplifies a meta object. Non recursive rational simplification.

118007 ˆDISTRIB\* ( meta 🡺 meta' T )

( meta 🡺 meta F )

Distribute \*. Returns FALSE if no distribution done.

3C2006 ˆDISTRIB/ ( meta 🡺 meta' T )

( meta 🡺 meta F )

Distribute /. Returns FALSE if no distribution done.

304006 ˆMETASINEXPA ( Meta 🡺 Meta' )

Expands SIN.

305006 ˆSINEXPA+ ( Meta 🡺 Meta' )

Expands SIN(x+y).

306006 ˆSINEXPA- ( Meta 🡺 Meta' )

Expands SIN(x-y).

307006 ˆSINEXPA\* ( Meta 🡺 Meta' )

Expands SIN(x\*y). Expands if x or y is an integer.

308006 ˆSINEXPA\*1 ( Meta2 Meta1 🡺 Meta' )

Expands SIN(x\*y). Meta1 is assumed to be an integer.

30A006 ˆMETACOSEXPA ( Meta 🡺 Meta' )

Expands COS.

**Direcc. Nombre Descripción**

30B006 ˆCOSEXPA+ ( Meta 🡺 Meta' )

Expands COS(x+y).

30C006 ˆCOSEXPA- ( Meta 🡺 Meta' )

Expands COS(x-y).

30D006 ˆCOSEXPA\* ( Meta 🡺 Meta' )

Expands COS(x\*y).

30E006 ˆCOSEXPA\*1 ( meta2 meta1 🡺 Meta' )

Expands COS(x\*y). meta1 represents an integer.

310006 ˆMETAEXPEXPA ( Meta 🡺 Meta' )

Expands EXP.

311006 ˆEXPEXPA+ ( Meta 🡺 Meta' )

Expands EXP(x+y).

312006 ˆEXPEXPA- ( Meta 🡺 Meta' )

Expands EXP(x-y).

313006 ˆEXPEXPA\* ( Meta 🡺 Meta' )

Expands EXP(x\*y).

314006 ˆEXPEXPANEG ( Meta 🡺 Meta' )

Expands EXP(-x).

315006 ˆEXPEXPA\*1 ( Meta2 meta1 🡺 Meta' )

Expands EXP(x\*y). meta1 represents an integer.

317006 ˆMETALNEXPA ( Meta 🡺 Meta' )

Expands LN.

318006 ˆLNEXPA\* ( Meta 🡺 Meta' )

Expands LN(x\*y).

319006 ˆLNEXPA/ ( Meta 🡺 Meta' )

Expands LN(x/y).

31A006 ˆLNEXPAˆ ( Meta 🡺 Meta' )

Expands LN(xˆy).

31E006 ˆMETATANEXPA ( meta 🡺 tan[meta] )

Expands tan[meta].

48.1.6 Tests

**Direcc. Nombre Descripción**

39A006 ˆmetafraction? ( Meta 🡺 Meta flag )

Tests if meta is a fraction of integers.

3BC006 ˆmetapi? ( Meta 🡺 Meta# )

Tests presence of \_ in a meta. # is the last occurence of \_ or 0.

3BD006 ˆmetaCOMPARE ( Meta2 Meta1 🡺 Meta2 Meta1 # )

Comparison of 2 meta.

#=0 if undef

#=1 if >

#=2 if <

#=3 if =

Assumes generic situation, e.g. Xˆ2> 0 in real mode. Look below STRICTmetaCOMPARE for a more careful comparison.

**Direcc. Nombre Descripción**

3BE006 ˆSTRICTmetaCOMPARE ( Meta2 Meta1 🡺 Meta2 Meta1 # )

Comparison of 2 meta.

#=0 if undef

#=1 if >

#=2 if <

#=3 if =

Unlike metaCOMPARE it does not assume generic situation.

3C3006 ˆmetareal? ( meta 🡺 meta flag )

Tests if IM[meta]==0.

Capítulo 49  
Polinomios

Los comandos de este capítulo tratan sobre polinomios y operaciones con ellos.

49.1 Referencia

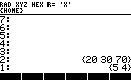
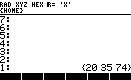
49.1.1 Operaciones con polinomios

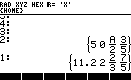
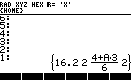
**Direcc. Nombre Descripción**

118006 ˆQAdd ( o1 🡺 o2+o1 )

Suma dos polinomios.

Por ejemplo:

 🡺 

 🡺 

119006 ˆRADDext ( o2 o1 🡺 o2+o1 )

Internal +. This is the same entry as ˆQAdd.

117006 ˆSWAPRADD ( o2 o1 🡺 o1+o2 )

SWAP, then QAdd.

115006 ˆQSub ( o2 o1 🡺 o2-o1 )

Subtracts two polynomials.

116006 ˆRSUBext ( o2 o1 🡺 o2-o1 )

Internal -. This is the same entry as ˆQSub.

114006 ˆSWAPRSUB ( o2 o1 🡺 o1-o2 )

SWAP, then QSub.

111006 ˆQMul ( Q1 Q2 🡺 Q )

Multiplication of polynomials with extensions.

112006 ˆRMULText ( Q1 Q2 🡺 Q )

Multiplication of polynomials with extensions.

This is the same entry as ˆQMul.

110006 ˆSWAPRMULT ( Q1 Q2 🡺 Q )

SWAP, then ˆQMul.

11C006 ˆQDiv ( o2 o1 🡺 o2/o1 )

Internal /.

11B006 ˆRDIVext ( o2 o1 🡺 o2/o1 )

Internal /. This is the same entry as ˆQDiv.

11A006 ˆSWAPRDIV ( o2 o1 🡺 o1/o2 )

SWAP, then QDiv.

0D9006 ˆQMod ( Q, Z 🡺 Q mod Z )

113006 ˆRASOP ( n1/d1 n2/d2 🡺 d1\*d2 n1\*d2 n2\*d1 )

Used by RADDext and RSUBext for rational input.

**Direcc. Nombre Descripción**

11F006 ˆRP# ( o2 # 🡺 o2ˆ# )

Internal power (not for matrices).

120006 ˆMPext ( ob #/Z/% prg\* 🡺 obˆ# )

General power with a specified multiplication program.

123006 ˆRPext ( o2 o1 🡺 o2ˆo1 )

Tries to convert o1 to an integer to call RP#, otherwise xˆext.

108006 ˆDISTDIVext ( P Q 🡺 quo mod T )

( P Q 🡺 P Q F )

Euclidean division. Assumes P and Q have integer coefficientes. Returns FALSE if sparse short division fails.

3E5006 ˆPTAYLext ( P, r 🡺 symb )

Taylor for polynomials.

15B006 ˆCARCOMPext ( Q1/Q2 🡺 Q1'/Q2' )

Extracts leading coefficients for the first variable from a rational polynomial.

3EE006 ˆQdivRem ( ob2 ob1 🡺 quo mod )

Polynomial Euclidean division of 2 objects.

Dispatchs to DIV2LISText for list polynomials.

3EF006 ˆDIV2LISText ( Z0 l1 l2 🡺 div mod )

Euclidean division, l1 and l2 are list polynomials.

Test first if l1=l2, then tries fast division, if it fails switch to SRPL division.

3F8006 ˆPDIV2ext ( A B 🡺 Q R )

Step by step Euclidean division for univar poly.

3F9006 ˆPsetSign ( P1 P2 🡺 sign[P2]\*P1 )

Sets sign of P1 according to leading coeff of P2.

3C4006 ˆModExpa ( Zn Fraction 🡺 Fraction modulo Zn )

3C5006 ˆModAdd ( Q1 Q2 Zn 🡺 Z )

Modular addition. Z = Q1+Q2 (mod Zn).

3C6006 ˆModSub ( Q1 Q2 Zn 🡺 Z )

Modular subtraction. Z = Q1-Q2 (mod Zn).

3C7006 ˆModMul ( Q1 Q2 Zn 🡺 Z )

Modular multiplication. Z = Q1\*Q2 (mod Zn).

3C8006 ˆModDiv ( Z1 Z2 Zn 🡺 Z )

Modular division. Z = Z1/Z2 (mod Zn).

3C9006 ˆModDiv2 ( Q1 Q2 Zn 🡺 quo mod mod' )

Modular division. mod' = Q1 mod Q2 mod Zn.

If Q1 and Q2 are integers, Q1 mod Q2 mod Zn is always 0.

3CA006 ˆModInv ( Z Zn 🡺 Z' )

Modular inversion. Z' = INV(Z) (mod Zn).

NONINTERR if GCD[Z,Zn] 6= 1 or if Z = 0 (otherwise the results would be unpredictable).

3CB006 ˆModGcd ( Q1 Q2 Zn 🡺 Q' )

Modular GCD.

49.1.2 Factorization

**Direcc. Nombre Descripción**

08E006 ˆBerlekampP ( P #prime 🡺 P F / P Lf #prime T )

Berlekamp's algorithm for finding modular factors of a univariate polynomial.

**Direcc. Nombre Descripción**

08F006 ˆBerlekamp ( P 🡺 P F / P Lf #prime T )

Berlekamp's algorithm for finding modular factors of a univariate polynomial with a leading frontend for finding linear factors faster.

The input polynomial must be square free, otherwise the polynomial is not fully factored.

Due to memory restrictions byte sized coefficients are used and the following restrictions were imposed: prime<128 and degree<256. If the conditions are not met FALSE is returned. BCD: prime≤97.

0A8006 ˆALG48FCTR? ( P 🡺 [ meta cst\_coeff TRUE | P FALSE ] )

Factorizes square-free polynomial in Erable format.

0A9006 ˆMfactTriv ( P 🡺 meta-factor P' )

Extracts all trivial power factors of P.

0AA006 ˆCheckPNoExt ( P 🡺 P flag )

Checks that P does not contain any DOCOL (i.e. extensions).

0AB006 ˆPPP ( P 🡺 PP PC )

Computes primitive polynomial and content of non-const P with respect to X1. The results are trimmed (provided P was).

0AC006 ˆPfactor ( P 🡺 Lfk Z )

Does a complete factorization of P. The result is trimmed.

0AD006 ˆPSqff ( P 🡺 Lfk )

Square-free and trivial factorization, including integer content, of P taken positive. Factors of same power are not necessarily merged or adjacent, but all Fi's are square-free.

0AE006 ˆPHFctr ( P 🡺 Lf )

Heuristic factorization of polynomial taken positive. LAM FullFact? must be bound. If LAM FullFact? is TRUE, a full factorization is done. If it is FALSE, only square-free and trivial

factorization is done.

0AF006 ˆPHFctr1 ( P 🡺 Lf )

Heuristic factorization of primitive polynomial.

LAM FullFact? must be bound. If TRUE, a full factorization is done. When FALSE, only a square-free and trivial factorization are done.

0B0006 ˆPHFctr0 ( P 🡺 Lf )

Heuristic factorization of primitive square-free non constant polynomial.

0D8007 ˆP2P# ( P 🡺 P' # )

Extracts trivial power of poly. P must be a valid poly (if list, begin with a non zero coeff).

0B1006 ˆDeCntMulti ( R 🡺 L )

Transforms list with count into simple list.

R = { {f1 #k1} ... {fn #kn} }

L = { f1 f1 .. fn fn }.

0B2006 ˆDoLS ( L S F 🡺 L' )

Applies program F(Li,S) to every elem of L.

0B3006 ˆPNFctr ( Z 🡺 Lf )

Factorization of positive integer as polynomial.

Lf = {} if Z is 1

Lf = { {Z1 #k1} ... {Zn #kn} }

o/w.

**Direcc. Nombre Descripción**

0B4006 ˆPSQFF ( P 🡺 Lsqff )

Computes the square-free factorization of primitive P. The result is trimmed (provided P was).

0B5006 ˆLiftZAdic ( p z F 🡺 L )

Lift n-1 z-adic factorization into n factorization.

0B6006 ˆLFCProd ( C L 🡺 C P )

Calculates combination product.

0B7006 ˆUfactor ( P 🡺 Lf )

Factorization of a square free primitive univariate polynomial.

0B8006 ˆUFactor1 ( P 🡺 Lf )

Factorization of a square free primitive univariate polynomial of degree > 2.

0B9006 ˆMonicLf ( Lfp p 🡺 Lfp' )

Converts true modular factorization to monic factorization by dividing by the leading coefficient of factor 1.

0BA006 ˆDemonicLf ( Lfp lc p 🡺 Lfp' )

Converts monic modular factorization to true modular factorization by multiplying factor1 by lcoeff.

0BB006 ˆLiftLinear ( #root1 .. #rootn #n 🡺 )

Lifts modular roots of a polynomial to find linear factors of a univariate polynomial.

Lflin = list of found true factors

Lfplin' = remaining linear factors

P' = remaining polynomial

Assumes UFactor lambda variables available and uses them for input and output.

0BC006 ˆLiftGeneral ( 🡺 )

Lifts factorization mod p to factorization mod pˆk where pˆk exceeds the factor bound for succesful true factor extraction. Assumes UFactor lambda variables.

0BD006 ˆUFactorDeg2 ( P 🡺 Lf )

Factorization of a degree 2 polynomial. Polynomial is univariate, square free and primitive.

0BE006 ˆCombineFac ( P Lfp p 🡺 Tf Tfp )

Combines modular factors to true factors. P is the polynomial to factor, Lfp is the list of modular factors, and p the modulo. The entry returns the a list of found true factors (Tf) and the list of modular factors for each true factor (Tfp)

0BF006 ˆCombProd ( lc Lfp p Cb 🡺 F )

Calculates modular combination.

0C0006 ˆCombInit ( #r 🡺 Cb )

Inits modular combination list to value { 1 0 0 0 .. }.

0C1006 ˆCombNext ( Cb 🡺 Cb' flag )

Gets next possible modular combination. Assumes Cb is valid and is in tempob area.

0C2006 ˆRmCombNext ( Lf Cb 🡺 Lfrm Lf' Cb' flag )

Removes next possible combination after a successful combination has been found, and remove the used factors from the factor list.

0C3006 ˆPfactTriv ( P 🡺 P' Lf )

Extracts all trivial power factors of P.

0C4006 ˆVarFactor ( P #var 🡺 P #n )

Calculates what power of the given variable is a factor in P.

**Direcc. Nombre Descripción**

0C5006 ˆPfactPowCnt ( P 🡺 P Lk flag )

Calculates trivial power factors in P. flag is TRUE if any of the powers is nonzero.

0C6006 ˆPdivLk ( P Lk 🡺 P' )

Divides polynomial by its trivial powers.

282006 ˆFEVIDENText ( P 🡺 meta-fact cst coeff )

Real mode: full factorization over the integer

Complex mode: find all 1st order factors of P.

49.1.3 General Polynomial Operations

**Direcc. Nombre Descripción**

09B006 ˆONE{}POLY ( ob 🡺 {ob} ob1 🡺 Q )

Replaces ONE{}N for polynomial building.

09C006 ˆTWO{}POLY ( ob1 ob2 🡺 Q )

Replaces TWO{}N for polynomial building.

09D006 ˆTHREE{}POLY ( ob1 ob2 ob3 🡺 Q )

Replaces THREE{}N for polynomial building.

09E006 ˆTWO::POLY ( ob1 ob2 🡺 :: )

Replaces 2Ob>Seco for polynomial building.

09F006 ˆ::POLY ( Meta 🡺 :: )

Replaces ::N for polynomial building. As opposed to the regular ::N code, we do pop the binary number. This is enforced by the entry to the common polyxml code.

0A0006 ˆ{}POLY ( Meta 🡺 Q )

Replaces {}N for polynomial building. As opposed to the regular {}N code, we do pop the binary number. This allows us to enter the code here with fixed sizes, as in ONE{}POLY and TWO{}POLY.

0A7006 ˆ>POLY ( Meta 🡺 Q )

Builds polynomial.

0A1006 ˆ>TPOLY ( P ob 🡺 P' )

Replaces >TCOMP for polynomial building.

0A2006 ˆ>HPOLY ( P ob 🡺 P' )

Replaces >HCOMP for polynomial building.

0A3006 ˆ>TPOLYN ( P ob1 .. obn #n 🡺 P' )

Improved >TCOMP for polynomial building.

0A4006 ˆ>HPOLYN ( P ob1 .. obn #n 🡺 P' )

Improved >HCOMP for polynomial building.

0A5006 ˆMKPOLY ( #n #k 🡺 P )

Crea polinomio de la n-ésima variable elevada al exponente k.

2AB006 ˆMAKEPROFOND ( ob # 🡺 {{{...{o}...}}} )

Embedds ob in the given number of lists.

4F4006 ˆTRIMext ( Q 🡺 Q' )

Removes unnecessary zeros from polynomial.

4F5006 ˆPTrim ( ob 🡺 ob' )

Trims polynomial.

0A6006 ˆONE>POLY ( Q 🡺 Q' )

Increases variable depth. Constants (Z,Irr,C) are not modified.

302006 ˆTCHEBext ( zint 🡺 P )

Tchebycheff polynomial. If zint>0 then 1st kind, if <0 then second kind.

**Direcc. Nombre Descripción**

3DE006 ˆLRDMext ( P # 🡺 [] )

Left ReDiMension. Adds 0 to the left of polynomial to get a symbolic vector of lenght #+1.

3DF006 ˆRRDMext ( {} # 🡺 {} )

Right ReDiMension: like LRDM but 0 at the right and {}.

3E0006 ˆDEGREext ( {} 🡺 degre )

Degree of a list-polynomial.

3E1006 ˆFHORNER ( P/d r 🡺 P[X]\_div\_[X-r]/d r P[r]/d )

Horner scheme.

3E2006 ˆHORNext ( P r 🡺 P[X]\_div\_[X-r] r P[r] )

Horner scheme.

3E4006 ˆMHORNext ( P r 🡺 P[X]\_div\_[X-r] r P[r] )

Horner scheme for matrices.

3E6006 ˆLAGRANGEext ( M 🡺 symb )

Lagrange interpolation. Format of the matrix is

[[ x1...xn ] [ f(x1)...f(xn) ]]

Returns a polynomial P such that P(xi)=f(xi)

10F007 ˆRESULTANT ( P1 P2 🡺 P )

Resultant of two polynomials. Depth of P is one less than depth of P1 and P2.

110007 ˆRESULTANTLP ( res g h P1 P2 🡺 +/-res g' h' P1' P2' )

Subresultant algorithm innerloop.

111007 ˆRESPSHIFTQ ( P Q 🡺 P' )

Resultant of P and Q shifted. gcd[Q(xr),P(x)]!=1 equivalent to r root of P' P' has same depth than P and Q.

112007 ˆADDONEVAR ( P 🡺 P' )

Adds one variable just below the main var. works for polynomial, not for fractions.

0CF007 ˆSHRINKEVEN ( P 🡺 P' )

Changes var Y=Xˆ2 in an even polynomial.

0D1007 ˆSHRINK2SYM ( N D 🡺 N' D' )

Shrinks 2 polynomials using symmetry properties.

0D2007 ˆSHRINKSYM ( N 🡺 N' )

Shrinks 1 polynomial using symmetry properties.

Degree of N must be even. If it is odd then N should be divided by X+1.

0D3007 ˆSHRINK2ASYM ( N D 🡺 N' D' )

Shrinks 2 polynomials using antisymmetry properties.

0D4007 ˆSHRINKASYM ( N 🡺 N' )

Shrinks 1 polynomial using antisymmetry properties. Degree of N must be even. If it is odd then N should be divided by X+1.

103006 ˆPNMax ( P 🡺 Z )

Gets the coefficient of P with max norm.

162006 ˆNDXFext ( Qnum Qden 🡺 Q )

Crea una expresión racional.

161006 ˆSWAPNDXF ( Qden Qnum 🡺 Q )

Crea una expresión racional.

164006 ˆFXNDext ( sym 🡺 num denom )

( ob 🡺 ob Z1 )

Retorna el numerador y el denominador de una expresión.

Equivale al comando FXND de User RPL cuando en la pila hay un objeto de clase simbólica.

**Direcc. Nombre Descripción**

163006 ˆSWAPFXND ( symb ob 🡺 ob num denom )

( ob’ ob 🡺 ob ob’ Z1 )

Hace SWAP, luego ˆFXNDext.

3D7006 ˆREGCDext ( a b 🡺 d u v au+bv=d )

3D8006 ˆEGCDext ( a b 🡺 d u v au+bv=d )

Bezout identity for polynomials.

0EA006 ˆPEvalFast? ( Z Pn 🡺 Z Pn F / Pn[Z] T )

Attempts to evaluate Pn at X1=Z using fast register arithmetic.

Fails if any of the following is true: Pn is not sunivariate; Z is polynomial after all; Z size is too big for register; Any overflow occurs during Horner evaluation.

10E007 ˆFLAGRESULTANT ( symb1 symb2 🡺 symb )

Resultant of two polynomials in symbolic form.

49.1.4 Tests

**Direcc. Nombre Descripción**

10B006 ˆUnivar? ( P 🡺 P flag )

Tests if polynomial is univariate.

10C006 ˆSUnivar? ( P 🡺 P flag )

Tests if polynomial is univariate and the coefficients are bounded by register size.

0CC007 ˆPOLYPARITY ( poly 🡺 Z )

Tests if a polynomial (internal rep) is even/odd/none. Z=1 if even, -1 if odd, 0 if neither even nor odd.

0D6007 ˆPOLYSYM ( P 🡺 Z )

Tests symmetry of coefficients of polynomial.

Z=1 for symmetric, -1 for anti, 0 otherwise.

0D7007 ˆPOLYASYM ( P 🡺 Z )

Tests "antisymmetry" of coef of polynomial. Z=1 for symmetric, -1 for anti, 0 otherwise.

Capítulo 50  
Hallando Raíces de Ecuaciones

En este capítulo encontrarás comandos relacionados a encontrar raíces de ecuaciones.

50.1 Referencia

50.1.1 Encontrando Raíces y Soluciones Numéricas

**Direcc. Nombre Descripción**

2F35B NUMSOLVE ( flag symb/prog id/lam aprox 🡺 raiz/extr #mens T )

( flag symb/prog id/lam aprox 🡺 #mens F )

Halla una solución a la ecuación para la variable id o lam.

Si la ecuación es un programa, éste debe ser uno que retorne un objeto algebraico.

Si la variable a hallar es un lam, ésta debe de existir.

Si la aproximación inicial es una mala estimación o hay operaciones imposibles (división entre cero, logaritmo de negativos, suma de unidades incompatibles, etc), retorna #A01 (Lackint#\_) y FALSE.

Si la ecuación es constante, retorna #A02 (Constant#\_) y FALSE.

Si al estar ocupada la máquina, el usuario presiona ON, retorna una lista con tres elementos (avance), #A03 (Attn#) y TRUE. Además si el flag es TRUE, retorna antes symb simplificado y el id/lam.

Si en la raíz encontrada, la función se hace cero, retorna la raíz, #A04 (Zero#\_) y TRUE.

Si en las inmediaciones de la raíz encontrada, la función cambia de signo, retorna la raíz, #A05 (RevSgn#\_) y TRUE.

Si no pudo hallar una raíz y en su lugar halló un extremo, retorna dicho extremo, #A06 (Extremum#\_) y TRUE.

Si una tecla diferente de ON es presionada mientras la maquina está ocupada, y:

- Si flag es TRUE, no se muestra el avance y al final se realiza la acción de la tecla presionada.

- Si flag es FALSE, se muestra el avance y la tecla presionada se quita de FLUSHKEYS.

Es llamado por los comandos ROOT y MROOT de User RPL y por el entorno SOLVE EQUATION.

272006 ˆMULMULText ( {} % 🡺 {}' )

Multiplica por % a todas las multiplicidades de la lista.

Es decir, el objeto del nivel 1 multiplica a todos los elementos de la lista de orden par.

La lista debe tener un número par de elementos.

Llama al comando ˆMETAMM2

274006 ˆMETAMM2 ( meta % 🡺 meta' )

Multiplica por % a todas las multiplicidades del meta.

Es decir, el objeto del nivel 1 multiplica a todos los elementos del meta de orden par.

**Direcc. Nombre Descripción**

273006 ˆMETAMULMULT ( % meta 🡺 meta' )

Multiplica por % a todas las multiplicidades del meta.

275006 ˆCOMPLISText ( {} 🡺 {}' )

Suprime múltiples ocurrencias de un factor sumando sus multiplicidades correspondientes.

Llama al comando ˆMETACOMPRIM

276006 ˆMETACOMPRIM ( Meta 🡺 Meta' )

Suprime múltiples ocurrencias de un factor sumando sus multiplicidades correspondientes.

278006 ˆMETACOMP1 ( f1..fk-1 mk-1 meta-res mk fk #

🡺 f1..fk-1 mk-1 meta-res )

279006 ˆADDLISText ( {} %n ob 🡺 {}' )

Adds ob with multiplicity %n to the list.

Checks if ob is in {}.

27A006 ˆDIVISext ( ob 🡺 {divisors} )

Returns list of divisors of ob.

27B006 ˆFACT1ext ( symb-poly 🡺 Lvar Q {} )

{} is the list of root/multiplicity of sym with respect to the current variable.

27C006 ˆFACTOext ( symb 🡺 Lvar Q {} )

{} is the list of factors/multiplicity of symb.

27D006 ˆZFACTO ( C 🡺 {} C Lfact )

27E006 ˆSOLVext ( symb 🡺 {} )

Numeric solver for univariate polynomials.

The list contains the roots without multiplicity.

27F006 ˆFRND ( ob 🡺 ob') )

Float rounding for %%, C%% or list of either type. Used by SOLVext to reconstruct factors.

280006 ˆBICARREE? ( P #5 🡺 meta cst\_coeff T )

( P #5 🡺 P #5 F )

( P # 🡺 P # F )

Searches if P is a bisquared 4-th order equation.

Returns a meta of factors and the multiplying coeff in that case.

281006 ˆREALBICAR ( f1 #1 coef 🡺 meta rest T )

113007 ˆIROOTS ( P 🡺 list )

Finds integer roots of a polynomial.

283006 ˆEVIDENText ( P 🡺 meta cst\_coeff )

Returns the roots of a polynomial P. Calls the numeric solver.

284006 ˆEVIDSOLV ( P 🡺 meta cst\_coeff )

Returns the roots of a 1st, 2nd order and some other poly. Calls the numeric solver if exact solving fails.

285006 ˆDEG2ext ( P 🡺 {} )

Returns the roots of a 2nd order polynomial.

286006 ˆMETADEG2 ( P 🡺 P meta )

Returns the roots of a 2nd order polynomial.

P must be of order 1 or 2.

287006 ˆMETADEG1 ( P 🡺 P meta )

Returns the roots of a 1st order polynomial.

P must be of order 1.

288006 ˆDEG1 ( f 🡺 r )

Root of a first order factor. f is one level depth deeper than r.

**Direcc. Nombre Descripción**

289006 ˆFDEG2ext ( P 🡺 meta-fact cst\_coef )

Returns factors of a 2nd order polynomial and the corresponding multiplying coefficient. tests for 1st order polynomial.

28B006 ˆRACTOFACext ( r 🡺 n d )

Converts root to factor. Factor is n/d, one level depth deeper than r.

28C006 ˆFACTORACext ( f 🡺 r cst\_coef )

Converts a factor to a root, solving 1st order factor. f and cst\_coef are one level depth deeper than r.

28D006 ˆRFACText ( ob # 🡺 {} intob meta )

{} is the list of variables. Meta is made of roots or factors of numerator (N) or denominator (D) or both (N/D), depending on #. ZERO for roots N/D; ONE for roots N; TWO for roots D with numeric solver call; THREE for roots D without num. solver call; FOUR for factors N/D; FIVE for factors N; SIX for factors D with numeric solver call; SEVEN for factors D without num.solver call.

28E006 ˆRFACT2ext ( ob {} # 🡺 {} intob meta )

Like RFACText, but the list of variables is given.

28F006 ˆRFACTSTEP3 ( ob 🡺 meta-fact )

Partial square-free factorization w.r.t. the main variable. Extract trivial factors Etape 3 ob 🡪 meta-fact.

290006 ˆRFACTSTEP5 ( %m on 🡺 add-to-meta-res )

Factorization of a square-free polynomial.

291006 ˆMETASOLV ( pn cst\_coeff 🡺 meta cst\_coeff )

Non-integer factorization (sqrt extensions and numeric). multiplicty is in LAM 5,.

293006 ˆMETASOLV2 ( cst\_coeff p 🡺 fr1 %m [fr2 %m] # cst\_coeff )

Returns roots/factors of 1st and 2nd order polynomials.

294006 ˆMETASOLV4 ( cst1 f1..fk #k cst2 🡺 fr1 %m..frn %m #2k cst\_coef )

Returns factors or convert to roots if needed. #k=1,2 or 4, fk are of order 1 or 2.

295006 ˆADDMULTIPL ( meta cst\_coeff 🡺 meta' cst\_coeff )

Adds multiplicities to a meta. Multiplicity is in LAM 5.

296006 ˆFACTOOBJext ( { fact mult } flag prg\* prgˆ 🡺 ob )

Rebuilds an object from its list of factors (flag=TRUE) or roots (flag=FALSE) using prg\* to multiply and prgˆ to take multiplicity power.

093006 ˆALG48MSOLV ( Lp 🡺 Lidnt Lsol )

Calculates Groebner basis multivar solution.

LAM3 must be bound to Lvar and LAM4 to Lidnt.

094006 ˆGMSOLV ( Lp 🡺 meta-sol )

Calculates Groebner basis multivar solutions.

LAM1 must be bound to the number of vars A solution is a

list { o1 ... on } where #n=LAM1 ok embedded in k-1 lists is the value of the k-th var ok may be undef.

095006 ˆGBASIS ( Lp 🡺 G )

Calculate Groebner basis.

G = { 1 } if no solutions

G = { 0 } if identically true.

**Direcc. Nombre Descripción**

096006 ˆGSOLVE ( Lp 🡺 Lg )

Calculate factorized Groebner basis.

Lg = { Lg1 Lg2 .. Lgn }

Lgi = independent solution (probably)

Lg = {} if no solutions

Lg = { { 0 } } if identically true.

097006 ˆGFACTOR ( Lp fctr? ! Lg )

Calculate Groebner basis or factorized Groebner basis. Redundant bases are not removed.

099006 ˆREDUCE ( p G 🡺 q )

Reduces polynomial with respect to given basis.

09A006 ˆFASTREDUCE ( r P 🡺 q T / r P F )

Assembly version of REDUCE for polynomials with short coefficients. Returns FALSE if an overflow occurs during the reduction. Assumes r is a genuine polynomial (not constant). Assumes G is not empty. Assumes G does not contain zeros (is trimmed).

37D006 ˆROOTM2ROOT ( {}/V 🡺 V' )

Transforms list of root/multiplicites to vector of roots.

0F2007 ˆPASCAL\_NEXTLINE ( {} 🡺 {}' )

Finds next line in the Pascal triangle.

0F3007 ˆDELTAPSOLVE ( Q 🡺 P )

Solves P(x+1)-P(x)=Q(x). Internal polynomial function.

Capítulo 51  
Operaciones de Cálculo

Los comandos en este capítulo están relacionados con varios aspectos del Cálculo, tales como límites, derivadas, expansión de fracciones parciales y transformadas de Laplace.

51.1 Referencia

51.1.1 Limites y Expansión de Series

**Direcc. Nombre Descripción**

46F006 ˆSYMTAYLOR ( symb id %/z 🡺 symb )

Taylor series expansion around point 0 (McLaurin's series) with regard to given variable, and of the given order.

471006 ˆTRUNCDL ( DL-l reste-l 🡺 truncated\_DL )

Series expansion truncation.

472006 ˆLIMSERIES! ( expression X=a|X %|zint 🡺 )

a lim DL-l rest-l num-l/deno-l equiv-l lvar # Series expansion. #=1 for X=a-h or X=-1/h.

477006 ˆLIMIT! ( symb 🡺 DL-l reste-l num-l/deno-l equiv.-l lim. lvar flag )

lim. = { symf direction }

478006 ˆLIMSTEP1! ( symb 🡺 { DL-l reste-l num-l/deno-l equiv.-l } flag )

47C006 ˆLIMLIM! ( # lvar equiv-l 🡺 lvar lim )

47F006 ˆLIMCMPL! ( reste-1-l reste-2-l 🡺 reste-l )

480006 ˆLIMEQUFR! ( n/d # 🡺 n/d-l equiv % )

481006 ˆLIMEQU! ( {} # 🡺 {} / {}-equiv-l {}-equiv-l { # # # } )

483006 ˆLIM+-! ( DL1...DLn #n op 🡺 DL flag )

DL = { DL-l reste-l num-l/deno-l equiv-l }.

48C006 ˆLIMDIVPC! ( #ordre num-l deno-l 🡺 num-l deno-l )

48E006 ˆLIMPROFEND! ( num deno #prof 🡺 num deno )

490006 ˆLIM%#! ( num-l deno-l {%...%} 🡺 num-l' deno-l' #prof {%...%} )

49E006 ˆLIM#VARX! ( lvar lvar 🡺 #varx )

4A1006 ˆHORNEXP! ( lim lvar X-l reste-l 🡺 lvar DL reste-l )

4B6006 ˆVARCOMP! ( var1 var2 🡺 flag )

4BA006 ˆVARCOMP32! ( var 🡺 0: )

4BD006 ˆLIMVALOBJ! ( ob lvar 🡺 symb )

4BE006 ˆLIMVAL! ( ob 🡺 coeff val )

4BF006 ˆEQUIV! ( {} lequiv 🡺 equiv ordre )

4C0006 ˆLVARXNX2! ( ob 🡺 ob lvarx lvarnx )

4C2006 ˆFindCurVar ( symb 🡺 symb )

Sets a new current var if needed.

4C3006 ˆLIMVAR! ( symb 🡺 symb lvar )

15C006 ˆRISCH13 ( {}/{}' 🡺 {}'' )

Assuming {}' has length 1, divides all elements of {} by this element. Used by RISCHext and by SERIES to have a nicer output of series.

51.1.2 Derivadas

**Direcc. Nombre Descripción**

3DC006 ˆPDer ( {} 🡺 der )

1A1006 ˆDERIVext ( ob id 🡺 ob' )

( ob sym 🡺 ob' )

( ob V 🡺 V' )

Calculates the derivative of the object. For a list argument calculates the gradient with respect to the variables in the list. If the variable is a symbolic, the first variable in it is used. Note that the gradient is a vector quantity, thus the result is returned as a list.

1A3006 ˆDERIVIDNT ( ob id 🡺 ob' )

Main entry point for derivative with respect to a identifier.

1A4006 ˆDERIVIDNT1 ( ob 🡺 ob' )

Main entry point for derivative with respect to the identifier stored in LAM1.

1A5006 ˆDERIV ( symb 🡺 symb' )

Derivative of symb with respect to the variable stored in LAM1.

1A6006 ˆMETADERIV ( Meta 🡺 Meta' )

Derivative of Meta object.

1BD006 ˆMETADER&NEG ( Meta 🡺 Meta' )

Meta derivative and negate.

1A9006 ˆMETADER+ ( Meta&+ 🡺 Meta' )

Meta derivative of addition.

1AA006 ˆMETADER- ( Meta&- 🡺 Meta' )

Meta derivative of subtraction.

1AB006 ˆMETADER\* ( Meta&\* 🡺 Meta' )

Meta derivative of multiplication.

1AC006 ˆMETADER/ ( Meta&/ 🡺 Meta' )

Meta derivative of division.

1AD006 ˆMETADERˆ ( Meta&ˆ 🡺 Meta' )

Meta derivative of power.

1AE006 ˆMETADERFCN ( Meta 🡺 Meta' )

Meta derivative of a function.

1AF006 ˆMETADERDER ( symb\_id\_; sym\_fcn\_; xDER #3 🡺 Meta' )

Meta derivative of a derivative of a function.

1B0006 ˆMETADERI4 ( Meta 🡺 Meta' )

Meta derivative of a defined integral.

1B1006 ˆMETADERI3 ( Meta 🡺 Meta' )

Meta derivative of an undefined integral.

1B2006 ˆMETADERIFTE ( Meta 🡺 Meta' )

Meta derivative of IFTE.

1B4006 ˆMETADEREXP ( Meta 🡺 Meta' )

Meta derivative of EXP.

1B5006 ˆMETADERLN ( Meta 🡺 Meta' )

Meta derivative of LN.

1B6006 ˆMETADERLNP1 ( Meta 🡺 Meta' )

Meta derivative of LNP1.

1B7006 ˆMETADERLOG ( Meta 🡺 Meta' )

Meta derivative of LOG.

1B8006 ˆMETADERALOG ( Meta 🡺 Meta' )

Meta derivative of ALOG.

**Direcc. Nombre Descripción**

1B9006 ˆMETADERABS ( Meta 🡺 Meta' )

Meta derivative of ABS.

1BA006 ˆMETADERINV ( Meta 🡺 Meta' )

Meta derivative of INV.

1BB006 ˆMETADERNEG ( Meta 🡺 Meta' )

Meta derivative of NEG.

1BC006 ˆMETADERSQRT ( Meta 🡺 Meta' )

Meta derivative of SQRT.

1BE006 ˆMETADERSQ ( Meta 🡺 Meta' )

Meta derivative of SQ.

1BF006 ˆMETADERSIN ( Meta 🡺 Meta' )

Meta derivative of SIN.

1C0006 ˆMETADERCOS ( Meta 🡺 Meta' )

Meta derivative of COS.

1C1006 ˆMETADERTAN ( Meta 🡺 Meta' )

Meta derivative of TAN.

1C2006 ˆMETADERSINH ( Meta 🡺 Meta' )

Meta derivative of SINH.

1C3006 ˆMETADERCOSH ( Meta 🡺 Meta' )

Meta derivative of COSH.

1C4006 ˆMETADERTANH ( Meta 🡺 Meta' )

Meta derivative of TANH.

1C5006 ˆMETADERASIN ( Meta 🡺 Meta' )

Meta derivative of ASIN.

1C6006 ˆMETADERACOS ( Meta 🡺 Meta' )

Meta derivative of ACOS.

1C7006 ˆMETADERATAN ( Meta 🡺 Meta' )

Meta derivative of ATAN.

1C8006 ˆMETADERASH ( Meta 🡺 Meta' )

Meta derivative of ASINH.

1C9006 ˆMETADERACH ( Meta 🡺 Meta' )

Meta derivative of ACOSH.

1CA006 ˆMETADERATH ( Meta 🡺 Meta' )

Meta derivative of ATANH.

1B3006 ˆDERARG ( meta-symb 🡺 arg1 .. argk der1 .. derk #k op )

Finds derivative of arguments.

1CB006 ˆpshder\* ( Meta1 Meta2 🡺 Meta2&Meta1'&\* )

Meta derivative utility.

1CC006 ˆSQRTINVpshd\* ( Meta1 Meta2 🡺 Meta2&SQRT&INV&Meta1'&\* )

Meta derivative utility.

51.1.3 Integración

**Direcc. Nombre Descripción**

07F007 ˆODE\_INT ( symb idnt 🡺 symb )

Integration with addition of a constant.

2C5006 ˆIBP ( u'\*v u 🡺 u\*v -u\*v' )

Internal integration by parts. If u is a constant return INTVX(u'\*v)+u. If stack 2 is a list it must e of the form { olduv u'\*v } then olduv will e added to u\*v at stack level 2. This permits multiple IBP in algebraic mode, e.g.

IBP(ASIN(X)ˆ2,X)

IBP(ANS(1),sqrt(1-Xˆ2))

IBP(ANS(1),C) the last step with an integral

containing a cst C.

2D0006 ˆPREVALext ( symb inf sup x 🡺 symb|x=sup - symb|x=inf )

Evaluates an antiderivative between 2 bounds

Does not check for discontinuities of symb in this interval.

2D1006 ˆWARNSING ( symb inf sup vx 🡺 symb inf sup vx )

Warns user for singularity.

2D2006 ˆINText ( symb x 🡺 int[$,x, symb, xt] )

Return unevaluated integral.

2D3006 ˆINT3 ( f(x) x y 🡺 F(y) where F'=f )

Undefined integration. No limit for underdetermined form.

3DD006 ˆINTEGRext ( {} 🡺 prim )

51.1.4 Fracciones Parciales

**Direcc. Nombre Descripción**

3D2006 ˆPARTFRAC ( o 🡺 symb )

Partial fraction expansion of o with respect to the current variable.

3D3006 ˆINPARTFRAC ( o list 🡺 symb )

Partial fraction expansion of o. lvar must be bound to LAM2, list is =lvar if o is in external format. list is NULL{} if o is still in internal format.

51.1.5 Ecuaciones Diferenciales

**Direcc. Nombre Descripción**

07E007 ˆDESOLVE ( symb symb1 🡺 list\_sols )

Resuelve una ecuación diferencial ordinaria.

1EC006 ˆFLAGDESOLVE ( symb symb1 🡺 list\_sols )

Resuelve una ecuación diferencial ordinaria.

Llama al comando ˆDESOLVE

Equivale al comando DESOLVE de User RPL.

081007 ˆLDECSOLV ( symb symb1 🡺 solución )

Resuelve una ecuación diferencial lineal de coeficientes.

Los argumentos son los mismos que ˆFLAGLDECSOLV

**Direcc. Nombre Descripción**

1EE006 ˆFLAGLDECSOLV ( symb symb1 🡺 solución )

Resuelve una ecuación diferencial lineal de coeficientes constantes de la forma: anyn+…a2y''+a1y'+a0=g(X)

En el nivel 2 debe estar la función g(X).

En el nivel 1 debe estar el polinomio característico.

Llama al comando ˆLDECSOLV

Equivale al comando LDEC de User RPL.

082007 ˆLDEGENE ( eq. carac 🡺 sol generale )

083007 ˆLDEPART ( 2nd membre, eq carac 🡺 eq. carac, sol part )

084007 ˆLDSSOLVext ( V M 🡺 V' )

M is the matrix of the system. V is the vector of the 2nd members.

085007 ˆODETYPESTO ( type 🡺 )

Store ode type in variable ODETYPE.

086007 ˆODE\_SEPAR ( symb 🡺 symb symb-y symb-x T )

( symb 🡺 symb F )

Tries to separate symb as a product of a function of y and a function of x.

51.1.6 Transformadas de Laplace

**Direcc. Nombre Descripción**

087007 ˆLAPext ( sym 🡺 sym' )

Transformadas de Laplace para polinomios\*exp/sin/cos.

Muestra LAP() para transformadas desconocidas.

1EA006 ˆFLAGLAP ( sym/MATRIX/%/C% 🡺 ob )

Transformadas de Laplace para polinomios\*exp/sin/cos.

Equivale al comando LAP de User RPL.

088007 ˆILAPext ( sym 🡺 sym' )

Transformada inversa de Laplace.

Inverse Laplace transform for rational fractions.

Delta functions for the integral part.

1EB006 ˆFLAGILAP ( sym/MATRIX/%/C% 🡺 ob )

Transformadas inversa de Laplace.

Equivale al comando ILAP de User RPL.

Capítulo 52  
Sumatorias

En este capítulo se encuentran los principales comandos relacionados con sumatorias, y también algunas subrutinas usadas por esos comandos.

52.1 Referencia

**Direcc. Nombre Descripción**

0F9007 ˆSUM ( sym idnt 🡺 sym )

Calcula la antiderivada discreta de una función.

Equivale al comando SIGMA de User RPL cuando en la pila hay un objeto de clase simbólica y un id.

0FB007 ˆSUMVX ( sym 🡺 sym )

Internal SUMVX.Works always with respect to the current variable.

0FD007 ˆRATSUM ( sym 🡺 sym )

Discrete rational sum.

0FE007 ˆFTAYL ( f shift 🡺 f' )

Taylor shift for rational fractions.

0FF007 ˆCSTFRACTION? ( ob 🡺 ob flag )

Taylor shift for rational fractions. Returns TRUE if ob is a cst fraction.

104007 ˆHYPERGEO ( symb 🡺 symb )

Tests and does hypergeometric summation.

100007 ˆNONRATSUM ( z/symb 🡺 symb )

Discrete summation (hypergeometric case).

103007 ˆmeta\_cst? ( meta 🡺 meta flag )

Tests for meta to be cst with respect to current var.

108007 ˆZEILBERGER ( f(n,k) n k d 🡺 C T )

( f(n,k) n k d 🡺 F )

Zeilberger algorithm \* NOT IMPLEMENTED YET\*.

109007 ˆSYMPSI ( sym 🡺 Psi(x) )

Digamma function.

10B007 ˆSYMPSIN ( sym int 🡺 Psi(x,n) )

Digamma function.

11C007 ˆ%%PSI ( %%x 🡺 %% )

Digamma function.

10D007 ˆIBERNOULLI ( #/zint 🡺 Q )

Bernoulli numbers.

0D9007 ˆNDEvalN/D ( num deno n d 🡺 num' deno' )

Evals list poly over a list fraction.

0DA007 ˆPEvalN/D ( P n d 🡺 num d # )

Evals list poly over a list fraction.

3C1006 ˆvgerxssSYMSUM ( Meta2 Meta1 🡺 meta )

Symbolic sum with tests for two zints. lam'sumvar bound to 'id/lam' and lam'sumexpr to 'expr'.

Capítulo 53  
Operaciones Modulares

Los comandos en este capítulo están relacionados con aritmética modular y otras operaciones modulares.

53.1 Referencia

53.1.1 Operaciones Modulares

**Direcc. Nombre Descripción**

252006 ˆFLAGFACTORMOD ( symb 🡺 symb )

FACTOR modulo.

253006 ˆMFACTORMOD ( M 🡺 M' )

FACTOR modulo for amtrices.

256006 ˆLIFCext ( {contfrac} 🡺 fraction )

Converts continued fraction to rational.

0E1006 ˆPEvalMod ( Q Z Zn 🡺 Q' )

Computes value of polynomial mod Zn.

0E2006 ˆQAddMod ( Q1 Q2 Zn 🡺 Q' )

Polynomial addition modulo Zn.

0E3006 ˆQSubMod ( Q1 Q2 Zn 🡺 Q' )

Polynomial subtraction modulo Zn.

0E4006 ˆQMulMod ( Q1 Q2 Zn 🡺 Q' )

Polynomial multiplication modulo Zn.

0E5006 ˆQDivMod ( Q1 Q2 Zn 🡺 Qquo Qrem )

Polynomial division modulo Zn. In regular division the coefficients in the remainder can increase very quickly to tens of digits, thus it is important to normalize the coefficients whenever possible.

0E6006 ˆQInvMod ( Q Zn 🡺 Q' )

Polynomial inversion modulo Zn.

0E7006 ˆQGcdMod ( Q1 Q2 Zn 🡺 Q' )

Polynomial GCD modulo Zn for univariate polynomials. The result is made monic.

4C5006 ˆISOL1 ( symb id 🡺 id symb' )

4C6006 ˆISOLALL ( symb id 🡺 id {} )

Internal SOLVE.

4C7006 ˆISOL2ext ( symb id 🡺 symb' )

( symb id 🡺 {} )

Like ISOL1 if isolflag is set. Otherwise returns the list of all found solutions.

4C8006 ˆBEZOUTMSOLV ( Lpoly Lidnt 🡺 Lidnt sols )

If no extension in Lpoly, calls ALG48 GSOLVE

Otherwise, solves by Bezout "Gaussian" elimination.

In the latter case, if system seems underdetermined, Lidnt is truncated. Then the system must be exactly determined and polynomials must be prime together.

4C9006 ˆROOT{}N ( meta of roots 🡺 list of roots )

Drops tagged roots.

**Direcc. Nombre Descripción**

4CA006 ˆMHORNER ( poly-l {r1...rk} # 🡺 P[r1...rk] )

Top-level call. Poly-l might be a matrix.

4CB006 ˆMHORNER1 ( P { r } 🡺 P[..r..] )

4CC006 ˆSQFFext ( Q 🡺 { F1 mult1 .. Fn multn } )

4CD006 ˆMSQFF ( Q 🡺 F1 mult1 .. Fn multn #2n )

Full square-free factorization of object. The result is given as a Meta object.

4CE006 ˆ%1TWO ( ob 🡺 ob %1 #2 )

Square free factorization of unknown (?) object.

See MSQFF.

4CF006 ˆMZSQFF ( Z 🡺 Z1 mult1 .. Zn multn #2n )

Full factorization of an integer.

4D0006 ˆMZSQFF1 ( Meta curfac %n newfac T 🡺 Meta curfac %n+1 )

( Meta curfac %n newfac F 🡺 Meta' newfac %1 )

Adds integer factor to factor list. If the factor is the same as the last time, only the multiplicity is increased.

4D2006 ˆMLISTSQFF ( P 🡺 Meta )

Full square-free factorization of a polynomial with a recursive call on the GCD of all coefficients.

4D3006 ˆMETASQFFext ( P-list 🡺 S1 %1 ..Se-1 %e-1 %e ee Te Re )

Square-free factorization.

4DF006 ˆLVARXNXext ( symb 🡺 symb x lvarnx lvarx )

Finds variable of symb depending on current variable and other variable. Using LVAR is impossible here because of sqrt.

4E0006 ˆISPOLYNOMIAL? ( ob 🡺 flag )

Retorna TRUE si symb es un polinomio con respecto a la variable actual.

4E1006 ˆ2POLYNOMIAL? ( symb1 symb2 🡺 symb1 symb2 flag )

Returns TRUE if symb1 and symb2 are polynomial with respect to current variable.

4E2006 ˆVXINDEP? ( symb 🡺 symb flag )

Returns TRUE if symb is independent of current variable.

4E4006 ˆRLVARext ( ob 🡺 {} )

Recursive search of all variables.

53.1.2 Comando LNAME y Similares

**Direcc. Nombre Descripción**

4DE006 ˆLIDNText ( ob 🡺 {id/lam} )

Retorna una lista de variables.

23F006 ˆUSERLIDNT ( ob 🡺 ob []/{} )

Retorna un vector simbólico con las variables (ids o lams) o una lista vacía.

Equivale al comando LNAME de User RPL cuando en la pila hay un objeto de clase simbólica o una formación.

Entre los comandos de esta sección (53.1.2) es el único que ordena las variables.

4EC006 ˆLIDNTLVAR ( ob ob' 🡺 ob {id/lam} {}lvar )

Retorna en el nivel 2 una lista con los ids y lams de ob' (usando LIDNText).

Retorna en el nivel 1 la lista anterior además de las variables de ob (usando LVARext) al final de esa lista.

53.1.3 Comando LVAR y Similares

**Direcc. Nombre Descripción**

4EE006 ˆLISTOPext ( 🡺 {+ \* - / ^ = NEG SQ i INV NEGNEG} )

( 🡺 {+ \* - / ^ = NEG SQ i INV NEGNEG RE IM CONJ} )

Lista de operaciones “racionales” básicas sin raíz cuadrada.

Si está activado el modo complejo (flag 103 activado), retorna 3 elementos más.

4EF006 ˆLISTOPSQRT ( 🡺 {+ \* - / ^ = NEG SQ i INV NEGNEG SQRT} )

( 🡺 {+ \* - / ^ = NEG SQ i INV NEGNEG RE IM CONJ SQRT} )

Lista de operaciones “racionales” básicas con raíz cuadrada.

Si está activado el modo complejo (flag 103 activado), retorna 3 elementos más.

4ED006 ˆLISTOPRAC ( 🡺 {} )

Lista de operaciones “racionales” básicas.

Si el flag 115 está activado (SQRT !simplified), llama a ˆLISTOPext

Si el flag 115 está desactivado (SQRT simplified), llama a ˆ ^LISTOPSQRT

4F0006 ˆLVARDext ( ob {}listop 🡺 {}lidnt )

( Meta {}listop 🡺 {}lidnt )

Retorna una lista de variables en ob (o meta) usando la lista dada de operaciones "racionales" básicas.

4E7006 ˆLVARext ( ob 🡺 ob {} )

Retorna una lista de variables.

Hace:

:: DUP FLASHPTR LISTOPRAC FLASHPTR LVARDext ;

Raíces cuadradas podrían estar incluidas en la lista de operaciones “racionales” básicas según el estado del flag 115.

23E006 ˆUSERLVAR ( ob 🡺 ob []/{} )

Retorna un vector simbólico con las variables o una lista vacía

Equivale al comando LVAR de User RPL cuando en la pila hay un objeto de clase simbólica o una formación.

4E9006 ˆVX> ( {} 🡺 {}' )

Si VX ya estaba en la lista, es movida al inicio de la lista.

Si VX no estaba en la lista, no hace nada.

4EA006 ˆVX! ( {} 🡺 {}' )

Si VX ya estaba en la lista, es movida al inicio de la lista.

Si VX no estaba en la lista, agrega la variable VX al inicio de la lista.

4E6006 ˆVXLVARext ( ob 🡺 ob {} )

Retorna una lista de variables. Si VX está en esa lista, es movida al inicio de la lista.

Raíces cuadradas no están incluidas en la lista de operaciones “racionales” básicas. Hace:

:: DUP FLASHPTR LISTOPext FLASHPTR LVARDext FLASHPTR VX> ;

4E8006 ˆVX>LVARext ( ob 🡺 ob {} )

Similar al comando ^LVARext, pero si VX está en la lista de variables, es movida al inicio de la lista.

Hace:

:: FLASHPTR LVARext FLASHPTR VX> ;

4F2006 ˆDEPTHext ( ob 🡺 #depth )

Retorna el número de niveles de listas incrustadas en ob.

**Direcc. Nombre Descripción**

4F3006 ˆDEPTHOBJext ( ob # 🡺 #depth )

Retorna el número de niveles de listas incrustadas en ob.

4E5006 ˆLLVARDext ( ob 🡺 #depth ob lvar )

Retorna lista de variables con ^LVARext y el número de niveles de listas incrustadas con ˆDEPTHOBJext

4F6006 ˆTRIMOBJext ( ob 🡺 ob ' )

Trims object.

4F7006 ˆNEWTRIMext ( Q 🡺 Q )

Recursively tests if Q is a list of one constant element. This is much faster than TRIMOBJext and sufficient for the output of programs which are trimmed on the fly.

4F8006 ˆ>POLYTRIM ( meta 🡺 {} )

Equivalent to {}POLY TRIMOBJext

4F9006 ˆELMGext ( % 🡺 %' )

( C% 🡺 C%'/% )

( %% 🡺 %%' )

( C%% 🡺 C%%'/%% )

( {} 🡺 {}' )

( symb 🡺 symb' )

( MATRIX 🡺 MATRIX' )

Trims small numbers (less than epsilon).

0E9006 ˆIsV>V? ( v1 v2 🡺 flag )

Returns TRUE if v1 is lexicographically after v2.

0EB006 ˆPZadic ( Q Z 🡺 Q' )

104006 ˆLISTMAXext ( P 🡺 P Z T depth )

( P 🡺 P ? F #0 )

Step 1 for gcdheu: Returns FALSE if gcdheu can not be applied (e.g. if P contains irrquads).

Returns TRUE otherwise, Z is the max of all integers of P or 2\*max if there are complex in P.

0EC006 ˆGCDHEUext ( A B 🡺 a b c pr[pgcd] A'/G' B'/G' flag )

Heuristic GCD.

Capítulo 54  
Tablas de Signos

A sign table is a list which describes thes sign of a expression in different intervals of a parameter. The list has an odd number of elements and looks like this:

{ value1 sign1.2 value2 sign2.3 ...signN-1.N valueN }

The values are key values of the parameter, usually −1, +1, and the locations of singularities or zeros in the expression. The values must be ordered and can be numbers or symbolic expressions. The signs show the sign of the expression in the interval between the adjacent values. Signs are ’-’, ’+’, and ’?’ (if the sign is unknown). To compute the sign table of an expression with respect to the current variable, use the entry SIGNE1ext. For example, the sign table of the expression ‘X2 − 1’ is

{ −1 ’+’ -1 ’-’ 1 ’+’ +1 }

Below is a list of the entries related to sign tables.

54.1 Referencia

**Direcc. Nombre Descripción**

237006 ˆSIGNE ( symb 🡺 sign )

Compute the sign table of the expression ith respect to the current variable. Internal version of the UserRPL command SIGNTAB.

0DC007 ˆSIGNE1ext ( expr 🡺 sign )

Sign table of a polynomial or rational expression.

0DE007 ˆSIGNUNDEF ( 🡺 sign )

Returns undefined sign table.

0DF007 ˆSIGNPLUS ( 🡺 sign )

Returns always positive sign table.

0E0007 ˆSIGNMOINS ( 🡺 sign )

Returns always negative sign table.

0E1007 ˆSIGNELN ( sign 🡺 sign )

Returns ln of a sign table.

0E2007 ˆSIGNEEXP ( sign 🡺 sign' )

Returns exp of a sign table.

0E3007 ˆSIGNESIN ( sign 🡺 sign' )

Returns sin of a sign table.

0E4007 ˆSIGNECOS ( sign 🡺 sign' )

Returns cos of a sign table.

0E5007 ˆSIGNETAN ( sign 🡺 sign' )

Returns tan of a sign table.

0E6007 ˆSIGNEATAN ( sign 🡺 sign' )

Returns atan of a sign table.

0E7007 ˆSIGNESQRT ( sign 🡺 sign' )

Returns sqrt of a sign table.

0E8007 ˆSUBSIGNE ( sign min max 🡺 sign' )

Truncates a sign table.

**Direcc. Nombre Descripción**

0E9007 ˆSIGNERIGHT ( sign ob 🡺 sign' )

Places ob at the end of a sign table.

0EA007 ˆSIGNELEFT ( sign ob 🡺 sign' )

Places ob at the beginning of a sign table.

0EB007 ˆ>SIGNE ( sign 🡺 sign' )

Prepends { -infinity ? } to a sign table.

0EC007 ˆSIGNE> ( sign 🡺 sign' )

Appends { ? +infinity } to a sign table.

0ED007 ˆSIGNMULText ( sign1 sign2 🡺 sign' )

Multiplies two sign tables.

0DB007 ˆPOSITIFext ( ob 🡺 ob flag )

Tries to determine if ob is positive. In internal representation, this depends on increaseflag so that x-1 is positive if increase flag is cleared, negative otherwise, because x is assumed to tend to +infinity or zero.

0EE007 ˆZSIGNECK ( ob 🡺 ob flag )

Returns sign of an expression. Error if unable to find sign.

0F0007 ˆZSIGNE ( ob 🡺 zint )

Returns sign of an expression. zint=1 for +, -1 for -, 0 for undef.

Expression does not need to be polynomial/rational.

0F1007 ˆzsigne ( meta 🡺 zint )

Returns sign of a meta symbolic. zint=1 for +, -1 for -, 0 for undef. Expression does not need to be polynomial/rational.

07D007 ˆCHECKSING ( symb inf sup vx 🡺 symb inf sup vx flag )

Checks for singularities in expr.

Capítulo 55  
Errores

Todos los mensajes de error del CAS tienen números de la forma DEXX (comienzan todos con DE). Puedes ver cada uno de estos mensajes de error en el apéndice E.

Los comandos ˆERABLEERROR y ˆGETERABLEMSG agregan DE00 al bint que se encuentra en la pila, para generar un error o conseguir la cadena correspondiente, de manera que sólo tendrás que especificar los dos últimos dígitos del número del mensaje de error.

Naturalmente, también puedes usar los comandos de error descritos en el capítulo 23 con los errores del CAS, usando los números de error completos.

55.1 Referencia

**Direcc. Nombre Descripción**

57E006 ˆERABLEERROR ( # 🡺 )

Llama a un error del CAS.

Equivale a usar

:: # DE00 #+ ERROROUT ;

Por ejemplo:

:: # 1C FLASHPTR ERABLEERROR ;

Genera el error # DE1Ch "Unable to factor"

57D006 ˆGETERABLEMSG ( # 🡺 $ )

Consigue una cadena de la tabla de mensajes de error.

Equivale a usar

:: # DE00 #+ JstGETTHEMSG ;

Por ejemplo:

:: # 1C FLASHPTR GETERABLEMSG ;

Consigue la cadena "Unable to factor", correspondiente al mensaje de error # DE1Ch

090006 ˆErrInfRes Error 305h

Genera el error "Infinite Result"

091006 ˆErrUndefRes Error 304h

Genera el error "Undefined Result"

092006 ˆErrBadDim Error 501h

Genera el error "Invalid Dimension"

57F006 ˆCANTFACTOR Error DE1Ch

Genera el error "Unable to factor"

580006 ˆTRANSCERROR Error DE20h

Genera el error "Not reducible to a rational expression"

581006 ˆNONUNARYERR Error DE21h

Genera el error "Non unary operator"

582006 ˆINTERNALERR Error DE26h

Genera el error "CAS internal error"

583006 ˆINVALIDOP Error DE28h

Genera el error "Operator not implemented (SERIES)"

584006 ˆISOLERR Error DE2Ah

Genera el error "No solution found"

585006 ˆNONINTERR Error DE2Ch

Genera el error "No solution in ring"

**Direcc. Nombre Descripción**

586006 ˆINTVARERR Error DE32h

Genera el error "No name in expression"

587006 ˆZ>#ERR Error DE35h

Genera el error "Integer too large"

0EF007 ˆSIGNEERROR Error DE36h

Genera el error "Unable to find sign"

588006 ˆZ<0ERR Error DE46h

Genera el error "Negative integer"

589006 ˆVXINDEPERR Error DE47h

Genera el error "Parameter is cur. var. dependent"

58A006 ˆNONPOLYSYST Error DE49h

Genera el error "Non polynomial system"

58B006 ˆCOMPLEXERR Error DE4Dh

Genera el error "Complex number not allowed"

58C006 ˆVALMUSTBE0 Error DE4Eh

Genera el error "Polyn. valuation must be 0"

58D006 ˆSWITCHNOTALLOWED Error DE4Fh

Genera el error "Mode switch not allowed here"

119007 ˆNONALGERR Error DE50h

Genera el error "Non algebraic in expression"

58E006 ˆERR$EVALext ( flag # 🡺 )

( comp 🡺 ? )

( comp %flag 🡺 ? )

Si en la pila hay TRUE y un bint, los borra de la pila.

Si en la pila hay FALSE y un bint, quita el flag y ejecuta el comando ˆERABLEERROR sobre el bint.

Si en la pila hay un compuesto, entonces cambia los bints del compuesto a cadenas (con ˆGETERABLEMSG) y luego ejecuta uno por uno los objetos del compuesto (como COMPEVAL).

Si en la pila hay un compuesto y un real (positivo), entonces efectúa lo descrito en el párrafo anterior, sólo si el número real representa a un flag de sistema que se encuentra activado.

Ejemplo:

::

{ "ERRORES del CAS:" "\0A1 " &$ BINT1 &$ "\0A2 " &$ BINT2 &$ "\0A3 " &$ BINT3 &$ "\0A..." &$ }

FLASHPTR ERR$EVALext

;

Retorna la cadena:

“ERRORES del CAS:

1 denominator(s)

2 root(s)

3 last

...”

Obs: la lista debe ir en una sóla línea en el editor de Debug 4x

58F006 ˆSys1IT ( comp 🡺 )

( comp 🡺 ? )

Si el flag 99 está desactivado (modo CAS conciso), borra el compuesto y no hace nada.

Si el flag 99 está activado (modo CAS verboso), llama al comado ˆERR$EVALext sobre el compuesto.

Capítulo 56  
CAS Configuración

Los comandos de este capítulo proveen maneras de configurar las operaciones del CAS.

Las configuraciones que pueden ser hechas aquí son las mismas que pueden ser hechas por el usuario cambiando los flags o con el formulario de entrada CAS MODES.

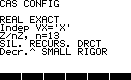
56.1 Referencia

**Direcc. Nombre Descripción**

08F007 ˆCFGDISPLAY ( 🡺 )

Muestra la configuración actual del CAS en la pantalla.

Llama a ˆEXACT?, ˆRCLVX y ˆRCLMODULO



Flag 103. Modo complejo o real. “COMPLEX” “REAL”

Flag 105. Modo aproximado o exacto. “APPROX” “EXACT”

Variable independiente del CAS.

Módulo actual usado por operaciones modulares del CAS.

Flag 99. Modo verboso o conciso. “VERB” “SIL.”

Flag 111. Simp. recurs. expr. rac. . “NON REC.” “RECURS.”

Flag 100. Modo paso a paso o directo. “STEP” “DRCT”

Flag 114. Orden polin. creciente o decreciente. “Incr.^” “Decr.^”

Flag 110. Matrices grandes o pequeñas. “BIG” “SMALL”

Flag 119. Modo descuidado o riguroso. “SLOPPY” “RIGOR”

090007 ˆNEWVX ( 🡺 )

Permite escribir el valor de VX en la línea de comandos.



Llama a ^RCLVX y a InputLine,

Si uno cancela, llama a ˆCFGDISPLAY

Si uno presiona ENTER llama a ^STOVX y ˆCFGDISPLAY

Si uno ingresa una entrada incorrecta genera el error

“Argumento: valor incorr”

Solo se acepta un id o una lista cuyo primer elemento sea un id

Si uno ingresa otro objeto, se guarda ‘X’ en VX.

091007 ˆNEWMODULO ( 🡺 )

Input new modulo from the user.

**Direcc. Nombre Descripción**

092007 ˆSWITCHON ( #flag 🡺 )

Pregunta al usuario si un cierto modo de operación debe ser encendido activando el flag de sistema #flag

1) Si #flag está activado, no hace nada.

2) Si #flag está desactivado, entonces:

a) Si el flag 123 está activado (modo switch prohibido), genera el error "Mode switch not allowed here"

b) Si el flag 123 está desactivado, (modo switch permitido), entonces:

• Si el flag 120 está activado (modo silencioso encendido), activa #flag sin preguntar.

• Si el flag 120 está desactivado (modo silencioso apagado), pregunta al usuario si desea activar el modo de operación correspondiente a #flag.

Si el usuario no quiere encenderlo, entonces genera el error “Mode switch cancelled”

093007 ˆSWITCHOFF ( #flag 🡺 )

Pregunta al usuario si un cierto modo de operación debe ser apagado desactivando el flag de sistema #flag

1) Si #flag está desactivado, no hace nada.

2) Si #flag está activado, entonces:

a) Si el flag 120 está activado (modo silencioso encendido), desactiva #flag sin preguntar.

b) Si el flag 120 está desactivado (modo silencioso apagado), pregunta al usuario si desea desactivar el modo de operación correspondiente a #flag.

Si el usuario no quiere apagarlo, entonces genera el error “Mode switch cancelled”

094007 ˆFLAGNAME ( # 🡺 # $ )

Encuentra el nombre de un flag.

BINT2 "Const -> value"

BINT3 "Function -> num"

BINT17 "Radian"

BINT103 "Complex"

BINT105 "Approx."

BINT114 "Incr. power"

BINT100 "Step by step"

BINT119 "ABS(X)->X"

Para otros, sólo retorna la palabra “flag” seguida del número.

BINT4 "flag 4."

No funciona correctamente.

1DC007 (ˆPUSHFLAGS) ( 🡺 )

Guarda los flags de usuario, los flags de sistema y la ruta actual en la variable ENVSTACK del directorio CASDIR.

Puede usarse este comando varias veces y se guardan todas las configuraciones en ENVSTACK.

Equivale al comando PUSH de User RPL.

**Direcc. Nombre Descripción**

1DD007 (ˆPOPFLAGS) ( 🡺 )

Restaura los flags de sistema, flags de usuario y el directorio guardado por el comando ˆPUSHFLAGS. Al hacer esto, borra la lista respectiva de la variable ENVSTACK.

Puede usarse este comando varias veces y se restauran sucesivamente las configuraciones guardadas por el comando ˆPUSHFLAGS. Si se acaban, la variable ENVSTACK contendrá una lista vacía.

Si la variable ENVSTACK no existe o no contiene una lista, manda el mensaje de error ”No valid environment stored”

Equivale al comando POP de User RPL.

095007 ˆCOMPLEXON ( 🡺 )

Turns complex mode on.

Depending on system flag 120, the user is asked first.

096007 ˆCOMPLEXOFF ( 🡺 )

Turns complex mode off.

Depending on system flag 120, the user is asked first.

097007 ˆEXACTON ( 🡺 )

Turns exact mode on.

Depending on system flag 120, the user is asked first.

098007 ˆEXACTOFF ( 🡺 )

Turns exact mode off.

Depending on system flag 120, the user is asked first.

09A007 ˆSETCOMPLEX ( 🡺 )

Pone la calculadora en modo complejo.

Para esto activa el flag 103.

099007 ˆCOMPLEXMODE ( 🡺 )

Hace ˆCOMPLEXMODE, luego ˆCFGDISPLAY

09B007 ˆCOMPLEX? ( 🡺 flag )

¿Calculadora en modo complejo?

Retorna TRUE si está en modo complejo (flag 103 activado).

Retorna FALSE si está en modo real (flag 103 desactivado).

09D007 ˆCLRCOMPLEX ( 🡺 )

Pone la calculadora en modo real.

Para esto desactiva el flag 103.

09C007 ˆREALMODE ( 🡺 )

Hace ˆCLRCOMPLEX, luego ˆCFGDISPLAY

09F007 ˆSETEXACT ( 🡺 )

Pone la calculadora en modo exacto y en modo GCD.

Para esto desactiva los flags 105 y 102.

09E007 ˆEXACTMODE ( 🡺 )

Hace ˆSETEXACT, luego ˆCFGDISPLAY

0A1007 ˆCLREXACT ( 🡺 )

Pone la calculadora en modo aproximado y en modo “no GCD”.

Para esto activa los flags 105 y 102.

0A0007 ˆNUMMODE ( 🡺 )

Hace ˆCLREXACT, luego ˆCFGDISPLAY

0A2007 ˆEXACT? ( 🡺 flag )

¿Calculadora en modo exacto?

a) Si el flag 105 está desactivado (exacto), retorna TRUE y desactiva el flag 102 (si GCD).

b) Si el flag 105 está activado (aproximado), retorna FALSE y activa el flag 102 (no GCD).

**Direcc. Nombre Descripción**

0A3007 ˆSTEPBYSTEP ( 🡺 )

Pone la calculadora en modo paso a paso. Para esto, activa el flag 100.

Luego muestra la configuración actual del CAS en la pantalla.

0A4007 ˆNOSTEPBYSTEP ( 🡺 )

Quita de la calculadora el modo paso a paso. Para esto, desactiva el flag 100.

Luego muestra la configuración actual del CAS en la pantalla.

0A5007 ˆVERBOSEMODE ( 🡺 )

Set verbose mode, refresh configuration display.

0A6007 ˆSILENTMODE ( 🡺 )

Set silent mode, refresh configuration display.

0A7007 ˆRECURMODE ( 🡺 )

Set recursive mode, refresh configuration display.

0A8007 ˆNONRECMODE ( 🡺 )

Set nonrecursive mode, refresh configuration display.

0A9007 ˆPLUSAT0 ( 🡺 )

Set positive mode, refresh configuration display.

0AA007 ˆSETPLUSAT0 ( 🡺 )

Set positive mode.

0AB007 ˆPLUSATINFTY ( 🡺 )

Set positive infinity mode, refresh configuration display.

0AC007 ˆCLRPLUSAT0 ( 🡺 )

Set positive infinity mode.

0AD007 ˆSPARSEDATA ( 🡺 )

Set full data mode, refresh configuration display.

0AE007 ˆFULLDATA ( 🡺 )

Set sparse mode, refresh configuration display.

0AF007 ˆRIGORMODE ( 🡺 )

Activa el modo riguroso, desactivando el flag 119 y luego muestra la configuración actual del CAS en la pantalla con el comando ^CFGDISPLAY

0B0007 ˆSLOPPYMODE ( 🡺 )

Activa el modo descuidado (no riguroso), activando el flag 119 y luego muestra la configuración actual del CAS en la pantalla con el comando ^CFGDISPLAY

0B1007 ˆSLOPPY? ( 🡺 flag )

¿Calculadora en modo descuidado?

Retorna TRUE si está en modo descuidado (flag 119 activado).

Retorna FALSE si está en modo riguroso (flag 119 desactivado).

1D2006 ˆSAVECASFLAGS ( 🡺 )

Saves CAS flags and current var.

1D4006 ˆRESTORECASFLAGS ( 🡺 )

Restore CAS flags and current var.

1D5006 ˆCASFLAGEVAL ( 🡺 )

Execute next runstream object with flag protection.

0C2007 ˆRCLMODULO ( 🡺 Z )

Trae MODULO desde el directorio CASDIR.

Si MODULO no es un entero, entonces guarda el entero 13 en la variable MODULO.

**Direcc. Nombre Descripción**

0C3007 ˆRCLPERIOD ( 🡺 sym )

( 🡺 % )

( 🡺 C% )

Trae PERIOD desde el directorio CASDIR.

Retorna el contenido de PERIOD, si este es real, complejo o de clase simbólica (id, lam, symb o entero). Si el contenido es un objeto de otro tipo, retorna el nombre global ID PERIOD

0C4007 ˆRCLVX ( 🡺 id )

Trae VX desde el directorio CASDIR.

Si VX contiene una lista retorna el primer elemento de esa lista, que debe ser un id.

Si VX no es un id o es una lista cuyo primer elemento no es un id, entonces guarda ‘X’ en la variable VX, primero.

0C5007 ˆSTOVX ( ob 🡺 )

Guarda ob en la variable VX de CASDIR.

0C6007 ˆSTOMODULO ( Z 🡺 )

Guarda Z en la variable MODULO de CASDIR.

Primero, si Z es menor a 2, entonces es convertido al entero 2

0C7007 ˆRCLEPS ( 🡺 % )

Trae EPS desde el directorio CASDIR.

Si EPS contiene un objeto no real o al real cero, entonces guarda 1.E-10 en la variable EPS.

0C8007 ˆISIDREAL? ( id/lam 🡺 id/lam ob T )

( id/lam 🡺 id/lam F )

¿La variable se asume como real por el CAS?

Si la variable REALASSUME no existe o no es una lista, la crea como una lista vacía y luego:

a) hace DUP TRUE (flag 128 activado, variables son reales).

b) agrega FALSE (flag 128 desact, vars complej permitidas).

Si la variable REALASSUME existe y es una lista:

a) Si la variable está en REALASSUME, agrega en la pila el objeto de REALASSUME donde está la variable y TRUE.

b) Si la variable no está en REALASSUME y el flag 128 está activado (variables son reales), hace DUP TRUE.

c) Si la variable no está en REALASSUME y el flag 128 está desactivado (vars complejas permitidas), agrega FALSE.

0C9007 ˆADDTOREAL ( id 🡺 )

( lam 🡺 )

( symb 🡺 )

Add idnt to the list of real var.

Equivale al comando ADDTOREAL de User RPL.

1A1007 FLASHPTR 007 1A1 ( 🡺 )

Establece a CASDIR como el directorio actual.

Si CASDIR no existe en HOME, entonces crea el directorio.

Si en HOME existe un objeto que no es directorio con el nombre CASDIR, entonces lo borra y luego crea el directorio.

Finalmente establece a CASDIR como el directorio actual.

**Direcc. Nombre Descripción**

1A3007 FLASHPTR 007 1A3 ( 🡺 )

Restaura las cinco variables fundamentales de CASDIR a sus valores predeterminados.

EPS 1.E-10 real

VX X id

PERIOD 2•Π symb

REALASSUME { X Y t S1 S2 } lista de ids

MODULO 13 entero

Si CASDIR no existe en HOME, entonces crea el directorio.

Si en HOME existe un objeto que no es directorio con el nombre CASDIR, entonces lo borra y luego crea el directorio.

Finalmente crea esas cinco variables.

0CA007 ˆRESETCASCFG ( 🡺 )

Restaura la configuración predeterminada del CAS. Para esto hace las siguientes acciones:

Llama a FLASHPTR 007 1A3

Activa los flags de sistema 27 y 128.

Desactiva los flags 2,3, 99-116 y 118-127.

Finalmente, pone la calculadora en modo radianes.

Equivale al comando CASCFG de User RPL.

1D0006 ˆVERNUMext ( 🡺 %version )

Retorna el número de version del CAS y la fecha en que fue liberado en la forma V.AAAAMMDD

Usado por el comando VER de User RPL.

Capítulo 57  
CAS Menus

Los comandos de este capítulo retornan los menús ya incorporados que muestran comandos del CAS, o hacen otras acciones relacionadas a los menús.

Para información general de los menús, lee el capítulo 37.

57.1 Referencia

**Direcc. Nombre Descripción**

1D1006 ˆMENUXYext ( #2 #1 🡺 {} )

Make list of Erable commands between the given numbers.

08D007 ˆMENUext ( $6...$1 🡺 )

If the CAS quiet flag is not set, displays the six strings as menu keys. Otherwise does nothing.

0B2007 ˆMENUCHOOSE? ( 🡺 prg flag )

Return best CHOOSE command.

0B3007 ˆMENUCHOOSE ( {} 🡺 )

Offers a selection to the user. If Flag -117 is set, only installs a menu. If not, offer a CHOOSE box.

0B4007 ˆMENUGENE1 ( 🡺 {} )

Menu for CAS.

0B5007 ˆMENUBASE1 ( 🡺 {} )

Base algebra menu.

0B6007 ˆMENUCMPLX1 ( 🡺 {} )

Complex operations menu.

0B7007 ˆMENUTRIG1 ( 🡺 {} )

Trigonometric operations menu.

0B8007 ˆMENUMAT1 ( 🡺 {} )

Matrix operations menu.

0B9007 ˆMENUARIT1 ( 🡺 {} )

Arithmetic operations menu.

0BA007 ˆMENUSOLVE1 ( 🡺 {} )

Solver menu.

0BB007 ˆMENUEXPLN1 ( 🡺 {} )

Exponential and logarithmic operations menu.

0BC007 ˆMENUDIFF1 ( 🡺 )

Differential calculus menu.

Capítulo 58  
Versiones Internas de los Comandos User RPL

Los comandos de este capítulo son cercanos a los correspondientes comandos de User RPL.

58.1 Referencia

**Direcc. Nombre Descripción**

218006 ˆISPRIME ( z/% 🡺 %0/%1 )

Internal ISPRIME.

1D6006 ˆFLAGEXPAND ( symb 🡺 symb' )

Internal xEXPAND. Expands symbolic expression.

1D8006 ˆFLAGFACTOR ( symb 🡺 symb' )

( z 🡺 symb )

Internal xFACTOR. Factors symbolic or number.

1D9006 ˆFLAGLISTEXEC ( symb {} 🡺 symb' )

Internal xSUBST for the case that level 1 is an array or a matrix.

1DA006 ˆFLAGSYMBEXEC ( symb symb' 🡺 symb'' )

Internal xSUBST for the case that level 1 is a ymbolic.

1DB006 ˆFLAGIDNTEXEC ( symb id 🡺 symb' )

Internal xSUBST for the case that level 1 is an id or a lam.

1DC006 ˆFLAGINTVX ( symb 🡺 symb' )

Internal xINTVX.

1DD006 ˆDERVX ( symb 🡺 symb' )

Internal xDERVX.

1DE006 ˆSOLVEXFLOAT ( % 🡺 {} )

Internal xSOLVEVX for a float.

1DF006 ˆSYMLIMIT ( symb symb' 🡺 symb'' )

Internal xLIMIT for scalars.

1E0006 ˆFLAGMATRIXLIMIT ( [] symb 🡺 []' )

Internal xLIMIT for matrices.

1E1006 ˆTAYLOR0 ( symb 🡺 symb' )

Internal xTAYLOR0.

1E2006 ˆFLAGSERIES ( symb id z 🡺 {} symb' )

Internal xSERIES.

1E4006 ˆPLOTADD ( symb 🡺 )

Internal xPLOTADD.

1E5006 ˆFLAGIBP ( symb1 symb2 🡺 symb3 symb4 )

Internal xIBP.

1E6006 ˆFLAGPREVAL ( symb1 symb2 symb3 🡺 symb4 )

Internal xPREVAL. Evaluates symb1 at the points symb2 and symb3 and takes the difference.

1E7006 ˆMATRIXRISCH ( [] id 🡺 symb' )

Internal xRISCH for matrix arguments.

1E8006 ˆFLAGRISCH ( symb id 🡺 symb' )

Internal xRISCH for non-matrix argumetns.

**Direcc. Nombre Descripción**

1E9006 ˆFLAGDERIV ( symb id 🡺 symb' )

Internal xDERIV.

1EA006 ˆFLAGLAP ( symb 🡺 symb' )

Internal xLAP.

1EB006 ˆFLAGILAP ( symb 🡺 symb' )

Internal xILAP.

1EC006 ˆFLAGDESOLVE ( symb symb' 🡺 symb'' )

Internal xDESOLVE.

1ED006 ˆFLAGLDSSOLV ( symb1 symb2 🡺 symb3 )

Internal xLDEC.

1EF006 ˆFLAGTEXPAND ( symb 🡺 symb' )

Internal xTEXPAND.

1F0006 ˆFLAGLIN ( symb 🡺 symb' )

Internal xLIN.

1F1006 ˆFLAGTSIMP ( symb 🡺 symb' )

Internal xTSIMP.

1F2006 ˆFLAGLNCOLLECT ( symb 🡺 symb' )

Internal xLNCOLLECT.

1F3006 ˆFLAGEXPLN ( symb 🡺 symb' )

Internal xEXPLN.

1F4006 ˆFLAGSINCOS ( symb 🡺 symb' )

Internal xSINCOS.

1F5006 ˆFLAGTLIN ( symb 🡺 symb' )

Internal xTLIN.

1F6006 ˆFLAGTCOLLECT ( symb 🡺 symb' )

Internal TCOLLECT.

1F7006 ˆFLAGTRIG ( symb 🡺 symb' )

Internal xTRIG.

1F8006 ˆFLAGTRIGCOS ( symb 🡺 symb' )

Internal xTRIGCOS.

1F9006 ˆFLAGTRIGSIN ( symb 🡺 symb' )

Internal xTRIGSIN.

1FA006 ˆFLAGTRIGTAN ( symb 🡺 symb' )

Internal xTRIGTAN.

1FB006 ˆFLAGTAN2SC ( symb 🡺 symb' )

Internal xTAN2SC.

1FC006 ˆFLAGHALFTAN ( symb 🡺 symb' )

Internal xHALFTAN.

1FD006 ˆFLAGTAN2SC2 ( symb 🡺 symb' )

Internal xTAN2SC2.

1FE006 ˆFLAGATAN2S ( symb 🡺 symb' )

Internal xATAN2S.

1FF006 ˆFLAGASIN2T ( symb 🡺 symb' )

Internal xASIN2T.

200006 ˆFLAGASIN2C ( symb 🡺 symb' )

Internal xASIN2C.

201006 ˆFLAGACOS2S ( symb 🡺 symb' )

Internal xACOS2S.

206006 ˆSTEPIDIV2 ( z1 z2 🡺 z3 z4 )

Internal xIDIV2.

207006 ˆFLAGDIV2 ( symb1 symb2 🡺 symb3 symb4 )

Internal xDIV2.

**Direcc. Nombre Descripción**

208006 ˆFLAGGCD ( symb1 symb2 🡺 symb3 )

Internal xGCD for the case with two symbolic arguments.

209006 ˆPEGCD ( symb1 symb2 🡺 symb3 symb4 symb5 )

Internal xEGCD for polynomials.

20B006 ˆABCUV ( symb1 symb2 symb3 🡺 symb4 symb5 )

Internal polynomial xABCUV.

20C006 ˆIABCUV ( z1 z2 z3 🡺 z4 z5 )

Internal integer xIABCUV.

20D006 ˆFLAGLGCD ( {} 🡺 {} symb )

Internal xLGCD.

20E006 ˆFLAGLCM ( symb1 symb2 🡺 symb3 )

Internal xLCM.

20F006 ˆFLAGSIMP2 ( symb1 symb2 🡺 symb3 symb4 )

Internal xSIMP2.

210006 ˆFLAGPARTFRAC ( symb 🡺 symb' )

Internal xPARTFRAC.

211006 ˆFLAGPROPFRAC ( symb 🡺 symb' )

Internal xPROPFRAC.

212006 ˆFLAGPTAYL ( P(X) r 🡺 P(X+r) )

Internal xPTAYL.

213006 ˆFLAGHORNER ( symb1 symb2 🡺 symb3 symb4 symb5 )

Internal xHORNER.

214006 ˆEULER ( z 🡺 z' )

Internal xEULER.

216006 ˆFLAGCHINREM ( A1 A2 🡺 A3 )

Internal xCHINREM.

217006 ˆICHINREM ( A1 A2 🡺 A3 )

Internal xICHINREM.

219006 ˆSOLVE1EQ ( symb id 🡺 {} )

Internal xSOLVE for single equations.

21A006 ˆSOLVEMANYEQ ( [] []' 🡺 {}'' )

Internal xSOLVE for arrays of equations.

21B006 ˆZEROS1EQ ( symb id 🡺 {} )

Internal xZEROS for single equations.

21C006 ˆZEROSMANYEQ ( [] []' 🡺 {} )

Internal xZEROS for arrays of equations.

21D006 ˆFCOEF ( [] 🡺 symb )

Internal xFCOEF.

21E006 ˆFROOTS ( symb 🡺 [] )

Internal xFROOTS.

21F006 ˆFACTORS ( symb 🡺 {} )

Internal xFACTORS.

220006 ˆDIVIS ( symb 🡺 {} )

Internal xDIVIS.

223006 ˆrref ( M 🡺 A M' )

Internal xrref.

229006 ˆMADNOCK ( M 🡺 symb1 []' []'' symb3 )

Internal xMAD.

22A006 ˆSYSTEM ( [] []' 🡺 []'' {} []''' )

Internal xLINSOLVE.

22B006 ˆVANDERMONDE ( {} 🡺 M )

Internal xVANDERMONDE.

**Direcc. Nombre Descripción**

22C006 ˆHILBERTNOCK ( z 🡺 M )

Internal xHILBERT.

22E006 ˆCURL ( [exprs] [vars] 🡺 [] )

Internal xCURL.

22F006 ˆDIVERGENCE ( [exprs] [vars] 🡺 symb )

Internal xDIV.

230006 ˆLAPLACIAN ( [expr] [vars] 🡺 symb )

Internal xLAPL.

231006 ˆHESSIAN ( symb A 🡺 M A' A'' )

Internal xHESS.

232006 ˆHERMITE ( z 🡺 symb )

Internal xHERMITE.

233006 ˆTCHEBNOCK ( %degree 🡺 symb )

Internal xTCHEBYCHEFF.

234006 ˆLEGENDRE ( z 🡺 symb )

Internal xLEGENDRE.

235006 ˆLAGRANGE ( A 🡺 symb )

Internal xLAGRANGE.

236006 ˆFOURIER ( symb z 🡺 C% )

Internal xFOURIER.

238006 ˆTABVAR ( symb 🡺 symb {{}} grob )

Internal xTABVAR.

239006 ˆFLAGDIVPC ( symb1 symb2 z 🡺 symb3 )

Internal xDIVPC.

23A006 ˆFLAGTRUNC ( symb1 symb2 🡺 symb3 )

Internal xTRUNC.

23B006 ˆFLAGSEVAL ( symb 🡺 symb' )

Internal xSEVAL.

23C006 ˆXNUM ( symb 🡺 symb' )

Internal xXNUM.

23D006 ˆREORDER ( symb id 🡺 symb' )

Internal xREORDER.

23E006 ˆUSERLVAR ( symb 🡺 symb [] )

Internal xLVAR.

23F006 ˆUSERLIDNT ( symb 🡺 [] )

Internal xLNAME.

241006 ˆADDTMOD ( symb1 symb2 🡺 symb3 )

Internal xADDTMOD for scalars.

242006 ˆMADDTMOD ( M M' 🡺 M'' )

Internal xADDTMOD for matrices.

243006 ˆSUBTMOD ( symb1 symb2 🡺 symb3 )

Internal xSUBTMOD for scalars.

244006 ˆMSUBTMOD ( M M' 🡺 M'' )

Internal xSUBTMOD for matrices.

245006 ˆMULTMOD ( symb1 symb2 🡺 symb3 )

Internal xMULTMOD.

Capítulo 59  
Miscelánea

En este capítulo se muestran los comandos que no encajaban en ninguno de los capítulos anteriores.

59.1 Referencia

59.1.1 Rutinas Para Mostrar Modo Verboso

**Direcc. Nombre Descripción**

579006 ˆVerbose1 ( $ 🡺 )

Display message on line 1 if verbose mode on.

57A006 ˆVerbose2 ( $ 🡺 )

Display message on line 2 if verbose mode on.

57B006 ˆVerbose3 ( $ 🡺 )

Display message on line 3 if verbose mode on.

57C006 ˆVerboseN ( $ # 🡺 )

Display message on given line if verbose mode on.

59.1.2 Evaluación

**Direcc. Nombre Descripción**

257006 ˆEvalNoCKx\* ( ob ob' 🡺 ob'' )

258006 ˆEvalNoCKx+ ( ob ob' 🡺 ob'' )

259006 ˆEvalNoCKx- ( ob ob' 🡺 ob'' )

25A006 ˆEvalNoCKx/ ( ob ob' 🡺 ob'' )

25B006 ˆEvalNoCKxˆ ( ob ob' 🡺 ob'' )

25C006 ˆEvalNoCKxCHS ( ob 🡺 ob' )

25D006 ˆEvalNoCKxINV ( ob 🡺 ob' )

25E006 ˆEvalNoCKxMOD ( ob ob' 🡺 ob'' )

25F006 ˆEvalNoCKxPERM ( ob ob' 🡺 ob'' )

260006 ˆEvalNoCKxCOMB ( ob ob' 🡺 ob'' )

261006 ˆEvalNoCKxOR ( ob ob' 🡺 ob'' )

262006 ˆEvalNoCKxAND ( ob ob' 🡺 ob'' )

263006 ˆEvalNoCKxXOR ( ob ob' 🡺 ob'' )

264006 ˆEvalNoCKxXROOT ( ob ob' 🡺 ob'' )

265006 ˆTABVALext ( fnct x {} 🡺 {}' )

Table of values.

59.1.3 Conversión

**Direcc. Nombre Descripción**

266006 ˆTOLISText ( o1..on #n 🡺 Lvar Q1..Qn )

Convert meta of symbolic objects to internal form.

267006 ˆFROMLISText ( Lvar Meta L 🡺 L' )

Conversion of elements of Meta objec to user format. Meta does not contain the #n number of element. L is the list of depth of the elements of Meta. For example to convert a polynomial, a vector and a matrix:

Lvar = { X }

Meta = { Z1 Z3 }

{ Z0 Z1 }

{ { Z1 { Z1 Z0 } } }

L = { #0 #1 #2 }

L' = { 'X+2' { 0 1 } { { 1 X } } }.

59.1.4 Qpi

**Direcc. Nombre Descripción**

074007 ˆQPI ( ob 🡺 ob' )

Internal xXQ.

073007 ˆQpiZ ( ob 🡺 symb )

Calls ˆQpi% and converts the resulting (real) integers into zints.

075007 ˆQpiSym ( symb 🡺 symb' )

Internal xXQ for symbolics.

076007 ˆQpiArry ( [] 🡺 []' )

Internal xXQ for arrays. Converts each element of the array.

077007 ˆQpiList ( {} 🡺 {}' )

Internal xXQ for lists. Converts each element of the list.

078007 ˆQpi ( %/C% 🡺 symb )

Internal xXQ for real and complex numbers.

079007 ˆQpi% ( % 🡺 symb )

xXQ for reals, but does not convert numbers to zints.

07A007 ˆGetRoot ( %' 🡺 %' %'' )

Tries to find a square number which is a factor of the argument.

The algorithm only tries numbers smaller than 1024ˆ2-1 and assumes that % is an integer. The returned results are such that %=(%')ˆ2\*%''. For numbers which do not contain a square factor, %'=1 and %''=%.

07B007 ˆApprox ( % 🡺 %' %'' )

Approximates a real number with a fraction. Returns numerator %' and denominator %''. The accuracy of the approximation is determinated by the current display format.

59.1.5 Infinito

**Direcc. Nombre Descripción**

2E2006 ˆINFINIext ( 🡺 '∞' )

2E3006 ˆMINUSINFext ( 🡺 '-∞' )

2E4006 ˆPLUSINFext ( 🡺 '+∞' )

2E5006 ˆ?ext '?'

Pushed the undefined symbolic.

2E6006 ˆPOSINFext ( symb 🡺 symb # )

Returns #1 if the symbolic contains '∞'.

2E1006 ˆTESTINFINI ( ob 🡺 ob flag )

Test if object contains infinity.

2E7006 ˆPOSUNDEFext ( symb 🡺 symb #1/#0 )

( ob 🡺 ob #0 )

Retorna #1 si el objeto simbólico contiene al simbólico indefinido '?'.

59.1.6 Constantes Ya Incorporadas

**Direcc. Nombre Descripción**

2EA006 ˆpi ( 🡺 'π' )

2EB006 ˆmetapi ( 🡺 π #1 )

2F1006 ˆmeta-pi ( 🡺 π xNEG #2 )

2E8006 ˆpisur2 ( 🡺 ' π/2' )

2F2006 ˆmetapi/2 ( 🡺 π 2 x/ #3 )

2E9006 ˆpisur-2 ( 🡺 '-π/2' )

2F4006 ˆmeta-pi/2 ( 🡺 π 2 x/ xNEG #4 )

2F3006 ˆmetapi/4 ( 🡺 π 4 x/ #3 )

2F5006 ˆmeta-pi/4 ( 🡺 π 4 x/ xNEG #4 )

2F6006 ˆpifois2 ( 🡺 '2\*π' )

2EC006 ˆ'xPI ( 🡺 xPI )

2F9006 ˆbase\_ln ( 🡺 'e' )

2FA006 ˆmeta\_e ( 🡺 e #1 )

2EE006 ˆ'xi ( 🡺 xi )

2ED006 ˆmetai ( 🡺 i #1 )

2EF006 ˆipi ( 🡺 'i\*π' )

2F0006 ˆmetaipi ( 🡺 i π x\* #3 )

2F8006 ˆmetapi\*2 ( 🡺 π 2 x\* #3 )

2F7006 ˆdeuxipi ( 🡺 '2\*i\*π' )

59.1.7 Aplicaciones de Listas

**Direcc. Nombre Descripción**

3F0006 ˆDIVOBJext ( {o1...on} ob 🡺 {o1/ob...on/ob} )

Division of all elements of a list by ob. Tests if ob=1.

3F2006 ˆLOPDext ( {o1...on} ob 🡺 {o1/ob...on/ob} )

LOPDext calls QUOText for the division, unlike DIVOBJ which calls RDIVext.

269006 ˆLOP1ext ( {} ob binop 🡺 {}' )

Applies non-recursively << ob binop >> to the elements of the list.

**Direcc. Nombre Descripción**

26A006 ˆLOPAext ( {} ob binop 🡺 {}' )

Applies recursively << op binop >> to the elements of the list (not the list elements themselves).

10F006 ˆLOPMext ( ob {} 🡺 {}' )

Multiplies each element of the list by the given object.

45F006 ˆLISTEXEC ( ob {} 🡺 ob' )

( ob {} 🡺 {}' )

The list should be of the form { 'X=1' 'Y=2' ... } in the first case or { 'X=1' 'X=2' } in the second case. In the first case, all occurences of X in ob are replace by 1, or Y by 2, etc. In the second case ob is evaluated with X=1, X=2 successively.

460006 ˆLISTEXEC1 ( {} objet 🡺 {}' )

461006 ˆSECOEXEC ( {} prog 🡺 {} )

Executes prog on each element of ob.

268006 ˆPFEXECext ( symb prg 🡺 symb )

26B006 ˆLISTSECOext ( composite 🡺 composite )

Applies 1LAM non-recursively to all elements of the list.

59.1.8 Irrquads

**Direcc. Nombre Descripción**

167006 ˆTYPEIRRQ? ( ob 🡺 flag )

Is ob an irrquad?

168006 ˆDTYPEIRRQ? ( ob 🡺 ob flag )

DUP, then ˆTYPEIRRQ?.

165006 ˆQXNDext ( irrq 🡺 a b c )

b=0 and c=1 if stack level 1 is not an irrq.

166006 ˆNDXQext ( a b c 🡺 irrq )

2D8006 ˆIRRQ#ULTIMATE ( ob 🡺 # c )

Finds « depth and returns ultimate c of an irrq.

508006 ˆQCONJext ( irrq 🡺 irrq' )

irrq-conjugate of an irrq. This is not the complex conjugate.

509006 ˆQABSext ( irrq 🡺 irrq sign )

Finds the sign of an irrq. Work always if irrq is made of Z.

51A006 ˆQNORMext ( Zirr 🡺 aˆ2-b\*cˆ2 )

Irrq-norm of an irrquad. This is not the complex modulus.

4D4006 ˆSECOSQFFext ( :: x<< a b c x>> 🡺 { fact1 mult1 ... factn multn } )

Factorization of irrquads and Gauss integers.

124006 ˆPREPARext ( o1 o2 🡺 a1 b1 c1 a2 b2 c2 )

Returns irrquad decomposition of o1 and o2.

with either c1=c2 or c1 and c2 have no factors in comon. c1<c2, ordering handled by LESSCOMPLEX? is made by type, then by CRC.

2DA006 ˆLISTIRRQ ( ob {} 🡺 {}' )

Add the C-part of all irrquads of object to the list.

59.1.9 Miscelánea

**Direcc. Nombre Descripción**

3E7006 ˆPSEUDOPREP ( o2 o1 🡺 o2\*a1.nˆ o1 a1.nˆ )

3FB006 ˆHSECO2RCext ( ob 🡺 ob' )

Conversion of constants from internal to user form.

3FC006 ˆSECO2CMPext ( seco 🡺 symb )

Back conversion of complex. polarflag should be disabled if not at the top level of rational expressions.

3FF006 ˆVALOBJext ( # {..{Q}..} {var1..varn} 🡺 {..{ob}..} )

Back conversion of objects embedded at depth # in lists. Simplifies var1..varn.

401006 ˆVAL2ext ( # {..{Q}..} {var1..varn} 🡺 {..{ob}..} )

Back conversion of objects embedded at depth # in lists. Does not simplify var1..varn. Conversion is done in asc. power if positivfflag is set, which is useful for SERIES and LIMIT commands.

402006 ˆINVAL2 ( P # 🡺 symbpoly )

LAM2 must contain Lvar, # is the depth.

403006 ˆMETAVAL2 ( # Meta\_list 🡺 Meta\_symb )

LMA2 must contain Lvar, LAM1 is modified.

404006 ˆVAL1 ( ob 🡺 ob )

LAM2 must contain Lvar, LAM1 is modified.

405006 ˆVAL1M ( ob 🡺 Meta\_symb )

LAM2 must contain Lvar, LAM1 is modified.

45C006 ˆIDNTEXEC ( symb idnt 🡺 symb' )

Tries to find idnt such that symb=0. Return a solution as an equality 'idnt=..' in symb'.

121006 ˆMP0 ( ob 🡺 ob 1 )

Returns number 1 of the selected type. The symbolic/ROMPTR one looks very strange it is used to avoid infinityˆ0/undefˆ0 to return 1.

26C006 ˆrpnQOBJext ( ob 🡺 ob' )

prg is fetched from the stack. Looks for all d1, d2, ... at the beginning of the name of idnt to determine if idnt represents a derivative of a user function. Stops if at a time the stripped idnt is in the current directory. Example 'd2d1Y' returns { #2 } << >> if 'd2d1Y' is not defined and 'd1Y' is defined as << >> or { #2 #1 } 'Y' if d2d1Y d1Y and Y are not defined.

29D006 ˆSIMPIDNT ( idnt 🡺 ob )

Evaluates idnt (looks recursively for its content if defined). Does not error for circular definition, but displays a warning.

29F006 ˆRCL1IDNT ( idnt/lam 🡺 ob )

Recursive content of an idnt. LAM1 to LAM3 must be bound.

2A7006 ˆSWPSIMPNDXF ( ob2 ob1 🡺 ob1/ob2 )

Simplified fraction (internal).

2A8006 ˆSIMPNDXFext ( ob2 ob1 🡺 ob2/ob1 )

Simplified fraction (internal).

2B6006 ˆCMODext ( C2 C1 🡺 C1 C2\_mod\_C1 )

2BD006 ˆSQFF2ext ( l1...ln #n-1 🡺 l1'...ln' #n-1 )

2BE006 ˆPPZ ( p 🡺 p/pgcd pgcd )

ob is the gcd of all constant coefficients of P (integer, Gauss integers, irrquads with the mplementation of the "gcd" for irrquads).

**Direcc. Nombre Descripción**

117007 ˆPPZZ ( ob 🡺 ob zint )

PPZ with further check to ensure returning a zint.

2BF006 ˆPZHSTR ( a z 🡺 a mod z )

2C0006 ˆHORNER1ext ( P r 🡺 P[r] )

2C1006 ˆPeval ( P r 🡺 P[r] )

P must be a list polynomial.

2C6006 ˆSQRT\_IN? ( {} 🡺 {} flag )

Returns TRUE if one element of {} is a symb containing a sqrt.

2C7006 ˆIS\_SQRT? ( symb 🡺 flag )

2C9006 ˆIS\_XROOT? ( symb 🡺 flag )

2CA006 ˆSTOPRIMIT ( symb 🡺 )

Stores antiderivative in PRIMIT variable.

2CB006 ˆCONTAINS\_LN? ( symb 🡺 symb flag )

2D4006 ˆFOURIERext ( symb n 🡺 cn )

Computes n-th Fourier coefficient of a 2 \_ periodic function.

2D9006 ˆLESSCOMPLEX? ( ob1 ob2 🡺 ob1 ob2 flag )

Compares objects by type and then by CRC.

flag is true if ob1 is less complex than ob2 (ob1>ob2).

If ob1 or ob2 is an irrq, find first ultimate type of ob1 and ob2.

If these ultimate types are equal sort is done by comparing the << depth.

2DD006 ˆTABLECOSext ( 🡺 {} )

Table of special COS values (k\*pi/12).

2DE006 ˆTABLETANext ( 🡺 {} )

Table of special TAN values (k\*pi/12).

101007 ˆLINEARAPPLY ( symb nonrat\_prg rat\_prg 🡺 symb )

Applies linearity. nonrat\_prg is applied for a non rational partç symb 🡺 symb. rat\_prg is applied for a rational part symb🡺symb. Linearity is applied on symb.

106007 ˆA/B2PQR ( A B 🡺 P Q R )

Writes a fraction A/B as E[P]/P\*Q/E[R]. Q and positive shifts of R are prime together.

107007 ˆGOSPER? ( P Q R 🡺 P R Y T )

( P Q R 🡺 F )

Solves P = Q E[Y] - R Y for Y.

0CB007 ˆFRACPARITY ( fr 🡺 Z )

Tests if a fraction (internal rep) is even/odd/none. Z=1 if even, 1 if odd, 0 if neither even nor odd.

0D5007 ˆFR2ND% ( fraction-l 🡺 N D % )

Extract trivial power of fraction.

4D1006 ˆMSECOSQFF ( ob 🡺 Meta )

Factorization of an extension.

Parte V

Apéndices

Apéndice A  
Herramientas para Desarrollar

Desarrollar software para la calculadora HP se puede hacer de dos maneras. Los programas pueden ser escritos y probados en un ordenador usando Debug 4x (que trae incorporado el emulador), o también puedes escribir software directamente en la calculadora.

En este apéndice describiremos algunas bibliotecas que permiten crear programas desde la calculadora, ver la representación en System RPL de objetos en la calculadora y convertir una biblioteca en un directorio.

Las herramientas ya incorporadas que necesitarás, por defecto, no son accesibles al usuario. Estas están en dos bibliotecas, las cuales no están vinculadas por defecto. La biblioteca 256 contiene varios comandos para “hacking” con la calculadora, y también el desensamblador. La biblioteca 257 contiene MASD, el compilador. Deberías tener vinculadas a estas dos bibliotecas. Si tienes instalada la biblioteca EXTABLE, entonces la biblioteca 256 será vinculada de manera automática. La biblioteca 257 (MASD) realmente no es necesario que esté vinculada, pues es posible llamar a MASD desde la biblioteca 256. Sin embargo, aun es buena idea vincularla.

La variable STARTUP es útil para configurar la calculadora. Esta variable (la cual debería estar en el directorio HOME) contiene un objeto que será ejecutado luego de cada warmstart. Esto puede ser usado para fijar todos los parámetros perdidos en un warmstart o para hacer cualquier otra cosa que quieras. El siguiente programa activa el teclado de usuario (el cual se pierde en cada warmstart) y también vincula la biblioteca 257.

« -62 SF 257 ATTACH »

A.1 La Biblioteca de Entradas (EXTABLE)

La biblioteca extable contiene las tablas de entradas soportadas y sus direcciones. Gracias a esta biblioteca, puedes escribir DUP y conseguir la direccion correcta para este comando; sin esta biblioteca, hubieras necesitado escribir PTR 3188 cada vez.

Transfiere extable a tu calculadora e instalala como cualquier otra biblioteca. Esto es todo lo que necesitas para usar nombres de comandos en lugar de sus direcciones. Extable aparece en el menú de bibliotecas y ésta contiene 5 comandos accesibles al usuario.

El primer comando nop no hace nada.

El comando **GETADR** retorna la dirección de un comando. Sólo debes colocar el nombre del comando como cadena en el nivel 1 de la pila y obtendrás la dirección como un hxs. La operación inversa es hecha por **GETNAME**: dada una dirección en la pila, retorna el nombre del comando.

Si no recuerdas bien el nombre de un comando puedes recurrir a **GETNAMES** y **GETNEAR**. Si colocas en la pila una cadena y ejecutas el comando **GETNAMES** podrás obtener una lista con todos los comandos cuyos nombres comienzan con la cadena ingresada. El comando **GETNEAR** es todavía más poderoso: dada una cadena, retorna una lista con todos los comandos cuyos nombres contienen a la cadena ingresada (incluso si está en medio del nombre).

**Direcc. Nombre User Descripción**

000102 nop ( 🡺 )

No hace nada.

001102 GETADR ( $ 🡺 hxs )

Consigue la dirección de un comando dado su nombre como una cadena.

002102 GETNAME ( hxs 🡺 $ )

Consigue el nombre correspondiente a una dirección.

003102 GETNAMES ( $inicio 🡺 {$} )

Retorna todos los nombres de los comandos que comienzan con la cadena dada.

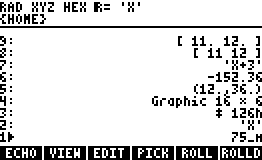
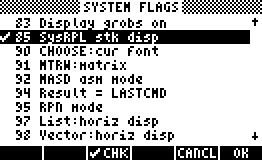
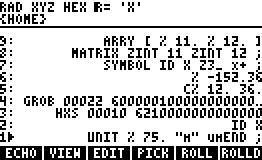
003102 GETNEAR ( $sub 🡺 {$} )

Retorna todos los nombres de los comandos que contienen a la cadena dada.

A.2 Hacking Tools

Las herramientas descritas aquí le hacen la vida más fácil al programador. Estas nos dan acceso a algunas funciones que no están normalmente disponibles para los usuarios de User RPL puro de la calculadora. Primero, las herramientas ya incorporadas en la calculadora serán descritas. Luego se describirá a una biblioteca externa.

Antes de describir las herramientas ya incorporadas encontradas en la biblioteca 256, mencionaremos un flag que es muy útil para los programadores en System RPL: el flag -85. Cuando este flag está activado, la “pila System RPL” está activa, esto significa que los objetos serán descompilados usando el descompilador System RPL antes de que sean mostrados. Las siguientes figuras muestran la pila antes y después de activar ese flag.

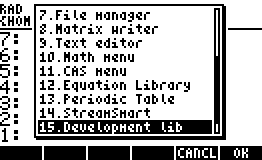
  

Esto significa que, si sólo veias External con la pila normal, ahora el nombre del comando (o PTR y la dirección, si ningún nombre es encontrado) será mostrado, si tienes instalada la biblioteca EXTABLE. Juega un poco con esto y verás cuan útil puede llegar a ser. Algunos objetos (como los reales y los enteros) aun mantienen su notación usual, pero en la pila interactiva todos los objetos son descompilados. Probablemente alternes entre los dos tipos de pila a cada momento. Sería una buena idea asignarle un programa simple a una tecla para alternar entre los dos tipos de pila. Por ejemplo, podemos asignar la acción a la tecla Shift derecho+MODE (END). Para esto, en User RPL podemos ejecutar

<< -85. IF DUP FS? THEN CF ELSE SF END >> 22.3 ASN

A.2.1 Biblioteca 256

La biblioteca 256 contiene algunas herramientas útiles para el programador. Esta biblioteca no aparece en el menú LIBS (debido a que no tiene título), pero puedes conseguir un menú con sus comandos escribiendo 256 MENU en la calculadora. Si la biblioteca está vinculada (como debería ser), puedes escribir los comandos, verlos en el catálogo y también una opción aparece en el menú APPS, la cual se llama “Development lib”, dandonos acceso a todos los comandos de la biblioteca 256.



Aquí está una descripción de todos los comandos presentes en esta biblioteca.

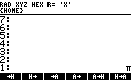
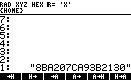
**Direcc. Nombre User Descripción**

000100 🡪H ( ob 🡺 $hex )

“Hacia hex”: Convierte un objeto en una cadena de caracteres hexadecimal.

Es una herramienta común para facilitar transferencia de objetos binarios.

Una cadena hexadecimal es una cadena que sólo tiene los caracteres 0 1 2 3 4 5 6 7 8 9 A B C D E F.

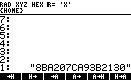
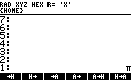
 

001100 🡪H ( $hex 🡺 ob )

“Desde hex”: Esta es la transformación opuesta: crea un objeto a partir de una cadena de caracteres hexadecimal.

La cadena debe ser válida.

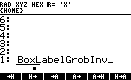
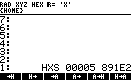
**Si la cadena no representa un objeto válido, esto puede dañar la memoria.**

002100 🡪A ( ob 🡺 hxs )

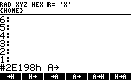
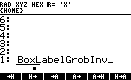
“Hacia address”: Dado un objeto, este comando retorna la dirección del objeto, el cual es siempre un hxs de 5 nibbles.

Los objetos cuyas direcciones sean menores que # 80000h están en ROM, y los objetos cuyas direcciones sean mayores están en RAM.

003100 A🡪 ( hxs 🡺 ob )

“Desde address”: Llama el objeto ubicado en la dirección especificada.

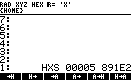
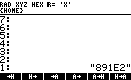
 

**Direcc. Nombre User Descripción**

004100 A🡪H ( hxs 🡺 $hex )

“Addres hacia hex”: Dada la dirección de un objeto, retorna la cadena de caracteres hexadecimal que representa al objeto (puedes luego usar esto con el comando POKE).

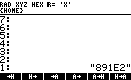
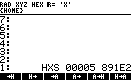
La cadena de caracteres hexadecimal de una dirección es una cadena de 5 caracteres donde la dirección está escrita hacia atrás.

005100 H🡪A ( $hex 🡺 hxs )

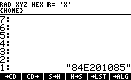
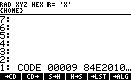
“Hex hacia address”: Dada la cadena de caracteres hexadecimal que representa a un objeto, retorna la dirección del objeto.

La cadena de caracteres hexadecimal de una dirección es una cadena de 5 caracteres donde la dirección está escrita hacia atrás.

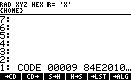
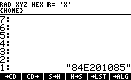
006100 🡪CD ( $hex 🡺 code )

“Hex hacia code”: Dada la cadena de caracteres hexadecimal que representa a un objeto, retorna el code (Assembly program) que representa al objeto.

007100 🡪CD ( code 🡺 $hex )

“Code hacia hex”: Dado el code (Assembly program) que representa al objeto, retorna la cadena de caracteres hexadecimal que representa al objeto.

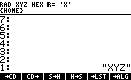
 

**Direcc. Nombre User Descripción**

008100 S🡪H ( $ 🡺 $’ )

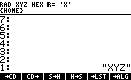
“Cadena hacia hex”: Dada una cadena, retorna la representación hexadecimal de sus caracteres.

Por ejemplo, al ser 58, 59 y 5A los códigos hexadecimales de X, Y y Z respectivamente, entonces "XYZ" se convierte en "8595A5" al usar el comando.

009100 H🡪S ( $’ 🡺 $ )

“Hex hacia cadena”: Dada la representación hexadecimal de unos caracteres, retorna una cadena.

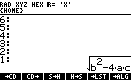
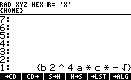
00A100 🡪LST ( symb 🡺 list )

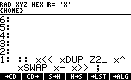
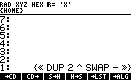
( prog 🡺 list )

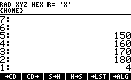
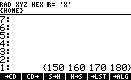
( list 🡺 list )

( metauser 🡺 list )

“Crea Lista”: Crea una lista a partir de un meta de User o de otro compuesto (programas y simbólicos). Un meta de User es cualquier número de objetos en la pila seguido por un contador representado como número real.

**Direcc. Nombre User Descripción**

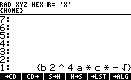
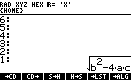
00B100 🡪ALG ( list 🡺 symb )

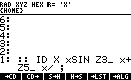
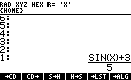
( prog 🡺 symb )

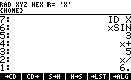
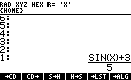
( symb 🡺 symb )

( metauser 🡺 symb )

“Crea Algebraico”: Crea un objeto simbólico a partir de un meta de User o de otro compuesto. El resultado puede llegar a ser 'Invalid Expression'.

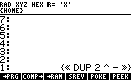
00C100 🡪PRG ( list 🡺 prog )

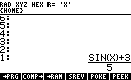
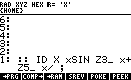
( symb 🡺 prog )

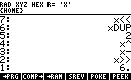
( prog 🡺 prog )

( metauser 🡺 prog )

“Crea programa”: Crea un programa a partir de un meta de User o a partir de otro compuesto (listas y simbólicos).

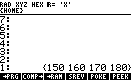
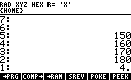
**Direcc. Nombre User Descripción**

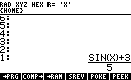
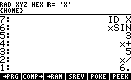
00D100 COMP🡪 ( list 🡺 metauser )

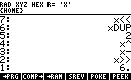
( symb 🡺 metauser )

( prog 🡺 metauser )

“Desde Compuesto”: Desintegra un objeto compuesto (listas, programas y simbólicos) y lo convierte en un meta de User.

00E100 🡪RAM ( ob 🡺 ob’ )

“A la RAM”: Hace una copia de un objeto en la RAM, donde sea que esté el objeto.

Este comando le permite volcar un objeto de la ROM en la RAM.

Puede extraer algunos comandos para su desensamblado, consulte la sección A.5 para obtener más información.

00F100 SREV ( $ 🡺 $’ )

“Reverse string”: Invierte los caracteres de una cadena.

Muy Rápido.

**Direcc. Nombre User Descripción**

010100 POKE ( hxs $hex 🡺 )

Comando de escritura de memoria: escribe nibbles en una dirección especificada en la RAM.

Ponga en el nivel 2 un hxs con la dirección, y en el nivel 1 una cadena de caracteres hexadecimal para escribir en esa dirección.

No se puede escribir datos en la Flash ROM usando este comando.

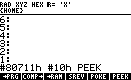
**Cuidado: escribir datos en la memoria aleatoriamente puede hacer que se pierda toda la memoria.**

011100 PEEK ( hxs hxs 🡺 $hex )

Comando de lectura de memoria: lee los nibbles de una dirección especificada en la RAM.

Ponga en el nivel 2 un hxs con la dirección, y en el nivel 1 un hxs con el número de nibbles que se desea conseguir.

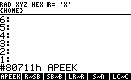
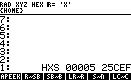
Debido al bank switching, es posible que los datos leídos desde la dirección # 40000h a # 7FFFFh no sean precisos.

012100 APEEK ( hxs 🡺 hxs’ )

Address PEEK: Lee la dirección almacenada en una dirección.

Este comando es parecido a PEEK, pero siempre devuelve cinco nibbles, retornandolos como un hxs.

013100 R~SB ( % 🡺 # )

( Z 🡺 # )

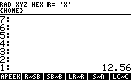
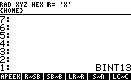
( # 🡺 % )

“Real↔Binario de System”: Convierte de real a bint y viceversa.

Si el real o entero es negativo, retorna BINT0.

Si el real o entero es mayor o igual a 1048575, retorna # FFFFF

Si el real entre 0 y 1048575 tiene parte decimal, retorna el BINT mas cercano.

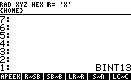
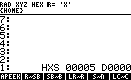
 

**Direcc. Nombre User Descripción**

014100 SB~B ( hxs 🡺 # )

( # 🡺 hxs )

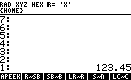
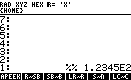
“Binario de System↔Binario de User”: Conviert bint a hxs y viceversa.

015100 LR~R ( % 🡺 %% )

( %% 🡺 % )

“Real Largo↔Real”: Convierte real largo a real y viceversa.

016100 S~N ( $ 🡺 id )

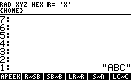
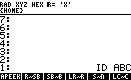
( id 🡺 $ )

“String↔Name”: Convierte cadena a nombre global y viceversa.

El tamaño máximo de la cadena es 127.

Con este comando se pueden crear nombres glovales inválidos y también un nombre global nulo.

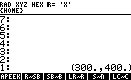
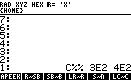
**Cuidado: No borrar o mover el directorio nulo que está en HOME. No modificar los datos que están en ese directorio.**

017100 LC~C ( C% 🡺 C%% )

( C%% 🡺 C% )

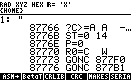
“Complejo Largo↔Complejo”: Convierte complejo largo a complejo y viceversa.

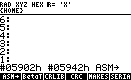
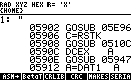
 

**Direcc. Nombre User Descripción**

018100 ASM🡪 ( code 🡺 $ )

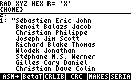
( hxsinicial hxsfinal 🡺 $ )

“Desde ASM”: Este comando desarma un objeto Code o un rango de direcciones de memoria que contiene el lenguaje de máquina Saturn para producir el código fuente del lenguaje ensamblador.  
 

019100 BetaTesting ( 🡺 $ )

“Cadena de prueba”: Retorna una cadena útil para fines de prueba.



01A100 CRLIB ( 🡺 library )

“Create Lib”: Crea una biblioteca basada en las variables que están en el directorio actual.

Ver el apéndice B para más información.

01B100 CRC ( library 🡺 hxs )

( bak 🡺 hxs )

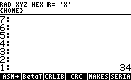
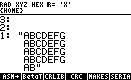
( $ 🡺 hxs )

“Cyclic redundance check”: Retorna el CRC de una biblioteca, un objeto de respaldo o una cadena.

01C100 MAKESTR ( % 🡺 $ )

( Z 🡺 $ )

“Make string”: Crea una cadena de prueba con la cantidad de caracteres dada en el nivel 1 de la pila.

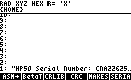
 

**Direcc. Nombre User Descripción**

01D100 SERIAL ( 🡺 $ )

“Número de serie”: Retorna una cadena con el número de serie interno de la calculadora.

Este es el número de serie del software y no coincide con el número de serie del hardware que está impreso en la parte posterior de la calculadora.



01E100 ASM ( $ 🡺 ob )

( $ 🡺 ob $debug )

( $ 🡺 $ {}error )

“Ensamblar/compilar”: compila una cadena que contiene código System RPL, Assembly Saturn o Assembly ARM en un objeto que la calculadora pueda usar.

Lea más adelante en este capítulo para obtener detalles sobre el formato de la cadena.

01F100 ER ( $ {}error 🡺 $editada )

“Error Checker”: inicia una sesión interactiva de comprobación de errores. Toma la salida de una ejecución fallida de ASM como entrada.

020100 🡪S2 ( ob 🡺 $ )

“Descompilación”: Descompila un objeto en el modo System RPL. Si la biblioteca "extable" está instalada, usará la tabla para buscar los mnemónicos adecuados para todos los comandos conocidos de System RPL.

Para la descompilación, 🡪S2 llama al comando ^FSTR2

021100 XLIB~ ( %lib %cmd 🡺 rompointer )

( hxslib hxscmd 🡺 rompointer )

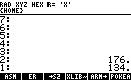
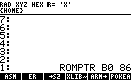
( hxslib %cmd 🡺 rompointer )

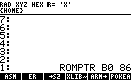
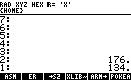
( %lib hxscmd 🡺 rompointer )

( rompointer 🡺 %lib %cmd )

“Conversión XLIB”: Crea un rompointer a partir del número de biblioteca en el nivel 2 y el número de comando en el nivel 1.

Si en la pila hay un rompointer, retorna el número de biblioteca y el número de comando como reales.

**Direcc. Nombre User Descripción**

022100 ARM🡪 ( code 🡺 $ )

( hxsinicial hxsfinal 🡺 $ )

“Desde ARM”: Este comando desarma un objeto Code o un rango de direcciones de memoria que contiene el lenguaje de máquina ARM para producir el código fuente del lenguaje ensamblador.

023100 POKEARM ( hxs $hex 🡺 )

Comando de escritura de memoria: escribe bytes en una dirección especificada en el espacio de la memoria ARM.

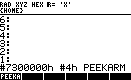
Ponga en el nivel 2 un hxs con la dirección, y en el nivel 1 una cadena de caracteres hexadecimal para escribir en esa dirección.

**Cuidado: escribir datos en la memoria aleatoriamente puede hacer que se pierda toda la memoria.**

024100 PEEKARM ( hxs hxs 🡺 $hex )

Comando de lectura de memoria: lee los nibbles de una dirección especificada en la dirección ARM.

Ponga en el nivel 2 un hxs con la dirección, y en el nivel 1 un hxs con el número de bytes que se desea conseguir.

**A.2.1 Operating Tools para la calculadora HP**

Wolfgang Rautenberg es el autor de una biblioteca llamada Operations Tools (llamada OT49) con varios comandos, algunos de los cuales son útiles para el programador System RPL. OT49 contiene un creador de bibliotecas y también puede convertir una biblioteca en directorio. Para convertir una biblioteca en directorio, sólo debes poner el número de la biblioteca en el nivel 1 de la pila y ejecutar el comando D↔L. Para crear una biblioteca, primero debes entrar en el directorio que será convertido, en el nivel 1 de la pila no debe haber un número real (o también puede estar vacía la pila), luego ejecutar el comando D↔L.

El comando Dtype muestra el tipo del objeto que está en el nivel 1 de la pila, si este objeto es un rompointer (XLIB) o un flashpointer, su contenido es llamado (a menos que esté escrito en puro lenguaje máquina) y el tipo del contenido es mostrado (con un asterisco al final).

Uno de los comandos más útiles es 3tog. Este comando alterna entre las tres representaciones de un objeto compuesto: como lista, como programa y como un objeto meta de User. Esto puede ser usado para manipular objetos System RPL sin descompilarlos realmente. 3tog descompone un programa en la pila y podrás reordenar sus elementos en la pila para luego llamara a 3tog nuevamente para reconstruir el programa.

Otro comando muy útil es Flg˜. Este comando alterna el estado de un flag. Sólo debes colocar en la pila el número del flag de sistema o de usuario y al llamar al comando, el estado del flag será cambiado. También será mostrado en la cabecera (header) el cambio que se ha realizado.

El comando Sys compila o descompila un objeto (dependiendo de si en el nivel 1 de la pila se encuentra una cadena u otro objeto).

A.3 El Compilador

El compilador incluido en la calculadora HP es MASD.

MASD es llamado con el comando ASM. Este comando espera una cadena en el nivel 1 de la pila y retorna el objeto compilado. Si hay algún error en la escritura de la cadena, entonces la cadena y una lista serán colocadas en la pila. Estos son los argumentos del comando ER que explicaremos más adelante.

El compilador MASD por alguna razón necesitá que la cadena que contiene al código **finalice** con el carácter “@”. Todas las cadenas con código fuente deberán de tener este carácter, pues de lo contrario MASD los rechazará. Dicho carácter debe estar sólo en una línea, al inicio de la línea y sin otros caracteres en esa línea (ni siquiera un salto de línea).

Otra cosa que debemos de tener en cuenta es el estado del flag -92. Si está activado MASD operará en modo System RPL. Si está desactivado, MASD operará en modo lenguaje ensamblador. Por lo tanto, si deseas hacer un programa System RPL (o compilar un objeto) en la calculadora, debes de tener activado el flag -92. Entonces, nada más necesitas hacer para compilar programas u otros objetos System RPL (sólo colocar @ en la última línea). Incluso es posible compilar código en lenguaje ensamblador cuando la calculadora está en modo System RPL: sólo envuelve el código entre las palabras CODE y ENDCODE.

Si estás en modo lenguaje ensamblador, es posible compilar código System RPL insertando estas dos líneas antes del código:

!NO CODE

!RPL

Ambas son llamadas directivas. La directiva !NO CODE le dice a MASD que compile la fuente como código System RPL y no como código de lenguaje máquina (una vez más, puedes insertar código de lenguaje ensamblador entre CODE y ENDCODE). Es buena idea poner siempre estas dos líneas al inicio de todos tus programas aun si estás en modo System RPL: de esta manera, el código podrá ser siempre compilado sin importar el estado de los flags.

Aquí está un código fuente que puede ser leído por MASD.

!NO CODE

!RPL

::

DUPTYPEZINT?

case

FPTR2 ^Z>R

DUPTYPEREAL? ?SEMI

SETTYPEERR

;

@

Lo mostrado arriba es el resultado de desensamblar el comando CKREAL. Como puedes ver, este convierte números enteros en reales.

Veamos que sucede al ejecutar el comando ASM si hay un error en la escritura de una cadena que contenga un código fuente. En este caso, la cadena original es retornada en el nivel 2 y una lista es colocada en el nivel 1. Ahora, con esos dos objetos en la pila,puedes ejecutar el comando ER. Con esto podrás ver un CHOOSEBOX donde se mostrarán los errores que hay y podrás escoger cada uno de estos para ir directamente al error en el código fuente. Corrige el error, presiona ENTER y luego escoge otro error, hasta que todos los errores hayan sido corregidos. Luego ejecuta ASM (y ER, si es necesario) nuevamente. Todavía mejor, usa el comando ASM2 de la biblioteca 257, el cual llama al comando ASM y luego, si hay algún error, llama también a ER.

**A.3.1 MASD y los Diferentes Tipos de Comandos**

Un programa System RPL puede llamar a tres diferentes tipos de comandos:

- Entradas **normales**, los cuales apuntan a alguna dirección de la ROM.

- Entradas **flashpointer**, los cuales apuntan a un comando en alguno de los bancos flash de la calculadora.

- Entradas **rompointer**, los cuales apuntan a un comando de una biblioteca (sea biblioteca ya incorporada o externa).

Para llamar a entradas normales “soportadas”, sólo debes escribir el nombre del comando. Para llamar a entradas normales no soportadas, tienes que escribirlas de la siguiente manera: PTR <address>, donde <address> es la direccion como está en la tabla de entradas.

Para entradas flashpointers soportadas (cuyos nombres comienzan con ^), tienes que usar el prefijo FPTR2. De esta manera, para llamar al comando ^Z>R, tienes que escribirlo asi en tu programa: FPTR2 ˆZ>R.

Una entrada flashpointer no soportada es llamada asi FPTR <bank> <cmd>. Donde <bank> son los tres últimos dígitos de la dirección como está listada en la tabla (en la práctica no es mayor que Fh), y <cmd> son los tres primeros dígitos.

Llamar a entradas rompointer (cuyos nombres empiezan con ~) es muy similar a llamar flashpointers. Si es un rompinter soportado, el prefijo debe ser ROMPTR2. Para entradas no soportadas, debes usar la sintaxis ROMPTR <lib> <cmd>. Donde <lib> son los tres últimos dígitos de la dirección, y <cmd> son las tres primeras.

**A.3.2 Características especiales de MASD**

El compilador MASD soporta algunas características especiales, que pueden ser útiles al programador. La primera característica que veremos es la que permite el uso de entradas no soportadas. Puedes definir un nombre para un comando no soportado, haciendo que se comporte como si fuera una entrada en extable (Esto sólo funciona para comandos normales, no para flashpointers ni rompointers). Para esto, usa la siguiente estructura:

EQU name address

donde name es el nombre de la entrada, y address es su dirección. Por ejemplo, la línea de abajo define el comando 2NELCOMPDROP, el cual retorna el segundo elemento de un compuesto:

EQU 2NELCOMPDROP 2825E

con esta definición puedes usar 2NELCOMPDROP en lugar de PTR 2825E para acceder al comando. Nota que esto sólo funciona para entradas normales.

Otra manera para facilitar la inclusión de entradas no soportadas (especialmente rompointers y flashpointers), pero que también es útil en otros casos, es usar los DEFINEs. Su estructura es la siguiente:

DEFINE name value

donde name es una palabra única, y value es el resto de la línea. Después de esta definición, donde sea que name sea encontrado en el archivo fuente, este será reemplazado por value. De esta manera, si por ejemplo usas el browser 49 (capítulo 34), este podría ser definido convenientemente de esta manera:

DEFINE ˆChoose3 FTPR 2 72

De tal manera que simplemente debes insertar ˆChoose3 cuando quieras llamar al browser.

A.3.2.1 Declarando Variables Locales Sin Nombre

Hay una estructura que te permite referirte a variables locales con nombres en el código fuente, pero que producen variables locales sin nombre, y de esta manera combina facilidad de uso con velocidad.

Las variables locales son declaradas asi:

{{ name1 name2 ... nameN }}

Después de esto, ingresando name1 se convertirá en 1GETLAM, name2 se convertirá en 2GETLAM. Precediendo el nombre de una variable con = o ! guarda algo en la variable, esto es, =name1 se convierte en 1PUTLAM, y asi sucesivamente. Presta atención a la manera como los nombres locales son declarados: el primer nombre de variable corresponde a 1GETLAM (esto es, el objeto que estaba en el nivel uno), y asi sucesivamente.

A.4 Desensamblando

Como se mencionó brevemente en la descripción de la Biblioteca 256 (véase la sección A.2), el comando →S2 es el desensamblador en System RPL. Este comando puede desensamblar cualquier objeto que esté en el nivel 1 de la pila en su código fuente, el cual puede ser ensamblado nuevamente con MASD. Por desgracia, todavía hay algunos errores en MASD, que impiden que algunos objetos desensamblados sean correctamente reensamblados. Todos esperamos que en una próxima versión de esta biblioteca, estos errores sean corregidos. A menudo, uno quiere ver cómo uno de los comandos integrados en la ROM de calculadora HP está construido. Por desgracia, es difícil hacerlo sólo con las herramientas incorporadas en la calculadora. Sin embargo, existen dos bibliotecas para este propósito: Nosy por Jurjen N. E. Boss, and CQIF, por Pierre Tardy. Ambas bibliotecas extraen y desensamblan código de la ROM.

**A.4.1 Usando Nosy**

La biblioteca Nosy, escrita por Jurjen N. E. Boss (j.bos@interpay-iss.demon.nl) es una herramienta que desensambla la ROM de la calculadora HP. Es muy fácil de usar.

Para usar Nosy, debes colocar en el nivel 1 de la pila una cadena que contenga el nombre del comando. También puedes colocar el comando, o su dirección (algunas otras entradas también serán aceptadas, ver la documentación de Nosy) y ejecutar el comando Nosy. Esto abrirá un browser interactivo donde podrás ver el objeto desensamblado y explorar en la ROM. Puedes usar las teclas de dirección para desplazarte. También puedes entrar a explorar otro comando dentro del código seleccionando el objeto y presionando ENTER o F6. Esto abrirá otro browser tal como abrió el primero. Para regresar al browser anterior debes presionar la tecla DROP (🡸) y para salir presiona ON.

Hay algunas otras funciones que puedes usar en el browser interactivo. Consulta su documentación para más detalles.

Apéndice B  
Creación de Bibliotecas

Las bibliotecas son una colección de comandos a los que el usuario puede acceder como si estos comandos estuvieran ya incorporados en la calculadora. Si has escrito un programa complejo con varias subrutinas, es mucho mejor distribuirlo como una biblioteca en lugar de distribuirlo como un directorio. Cuando el programa está en una biblioteca, el usuario no necesita navegar a través de las variables para poder acceder a tu programa; el usuario sólo debe tipear el nombre del comando desde cualquier lugar en donde se encuentre.

Los comandos de una biblioteca aparecen en el catálogo, y los usuarios pueden tener una ayuda disponible para cada comando de una biblioteca.

Hay un menú que muestra todas las bibliotecas instaladas. Pero también una biblioteca puede mostrarse toda o parte de esta a algunos de las cajas de selección propias de la calculadora, tales como APPS, STAT, NUM SOLVER, etc.

Por otra parte, se puede hacer que el usuario pueda acceder sólo a algunos de los comandos de la biblioteca. De esta manera, se evita que el usuario ejecute comandos que no debe y el programador de la biblioteca sólo debe preocuparse de proporcionar verificación de errores para los comandos accesibles al usuario.

Lo escrito debe ser suficiente para convencerte de distribuir tus programas como parte de bibliotecas.

B.1 Creación de bibliotecas desde la Calculadora

Si deseas crear una biblioteca desde la calculadora puedes usar el comando CRLIB de la biblioteca 256.

Sólo debes crear algunas variables especiales en un directorio, el cual especifica algunos aspectos de la biblioteca, y luego ejecutar el comando CRLIB.

Las variables especiales tienen nombres que comienzan con $. Estas variables configuran las características que tendrá la biblioteca que se creará:

**Variable Significado**

$ROMID Especifica el número de la biblioteca. Cada biblioteca debe tener un número único. En una calculadora no pueden estar instaladas 2 bibliotecas que tengan el mismo número.

**Debe ser un número real o entero**, en el rango 769-1791.

$TITLE Este es el título de la biblioteca. Los primeros 5 caracteres son mostrados en el título de la biblioteca.

**Debe ser una cadena**.

Si esta variable es una cadena vacía, entonces no se podrá acceder a la biblioteca desde el menú de bibliotecas, pero no evita hacer posible el acceso desde otro lugar como al presionar APPS, STAT, NUM SLV, etc.

$VISIBLE Esta es una lista de variables globales (nombres de los comandos visibles).

Las variables que figuran en esta lista se convertirán en comandos accesibles al usuario en la biblioteca resultante.

**Variable Significado**

$HIDDEN Esta es una lista de variables globales (nombres de los comandos invisibles).

Las variables que figuran en esta lista se convertirán en comandos que no serán accesibles al usuario en la biblioteca resultante.

$CONFIG Este objeto se encarga de configurar la biblioteca.

Este objeto es evaluado en cada warmstart.

Normalmente, es un programa que vincula la biblioteca al directorio HOME.

Por ejemplo, si el número de biblioteca es 1000:

En User RPL, la variable $CONFIG puede ser:

« 1000. ATTACH »

En System RPL, la variable $CONFIG puede ser:

:: HOMEDIR 1000 XEQSETLIB ;

O también:

:: 1000 TOSRRP ;

Si lo deseas, simplemente puedes guardar el número real 1. dentro de la variable $CONFIG para que un objeto de configuración por defecto sea producido, el cual vincula la biblioteca al directorio actual en cada warmstart.

$MESSAGE Esta es una lista de cadenas que estarán disponibles en la biblioteca para su uso como mensaje de error o para cualquier otro uso.

Si cada mensaje es usado sólo una vez, no vale la pena crear una tabla de mensajes. Pero si los mensajes son usados en muchos lugares, o si quieres que sea fácil cambiar el idioma de la biblioteca, entonces sería muy útil crear una tabla de mensajes.

La lista puede contener como máximo a 255 cadenas. En la calculadora cada mensaje es identificado como un bint único.

Por ejemplo, si el número de la biblioteca es 1000 (3E8 en hexadecimal), podrás acceder a un mensaje desde un programa usando:

# 3E801 JstGETTHEMSG

# 3E802 JstGETTHEMSG

.........

# 3E8FF JstGETTHEMSG

Para generar un mensaje de error mostrando una de esas cadenas desde un programa, puedes usar:

# 3E801 ERROROUT

O también:

# 3E801 DO#EXIT

En el capítulo 23 puedes ver más información sobre esto.

La variable $MESSAGE es opcional. No es obligatorio incluirla.

$EXTPRG Este debe ser un nombre global.

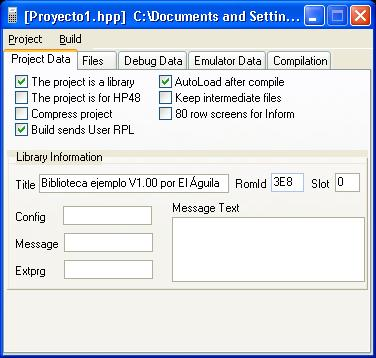
Es el nombre del comando que permite la personalización de algunos menús, adición de ayuda de comandos en el catálogo, y otras cosas.

Ver abajo para más información sobre esto.

Recuerda que sólo los comandos cuyos nombres se encuentran en las variables $VISIBLE o $HIDDEN serán convertidos en comandos de biblioteca. Por lo tanto, no debes olvidar que el nombre global contenido en la variable $EXTPRG también debe estar incluido en una de esas dos listas ($VISIBLE o $HIDDEN).

La variable $EXTPRG es opcional. No es obligatorio incluirla.

B.2 Creación de bibliotecas desde Debug 4x

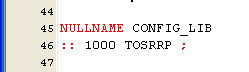
En el apéndice C se muestran los pasos para crear tu primera biblioteca en System RPL con Debug 4x. Debes leer ese apéndice para poder crear fácilmente bibliotecas de esa manera.

Cuando creas una biblioteca con Debug 4x, no es necesario crear variable en un directorio. En lugar de ello, debemos llenar algunos campos en la ventana Project.

Algunos elementos de la ventana Project son:

**RomID**: Especifica el número de la biblioteca. Debe contener a un número (259-1791) pero debes escribirlo en base hexadecimal (103-7FF). No está permitido el número 788 (314 en hexadecimal), pues esta es una biblioteca que contiene algunos comandos del CAS.

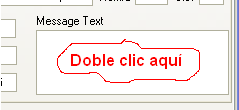
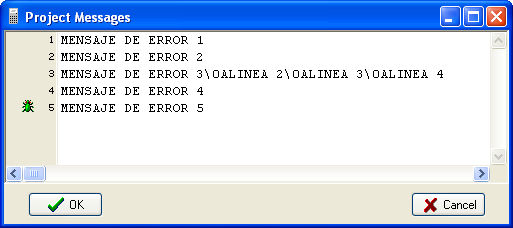
**Title**: Es el título de la biblioteca. Los primeros 5 caracteres son mostrados en el título de la biblioteca. Si dejas este campo en blanco, entonces no se podrá acceder a la biblioteca desde el menú de bibliotecas, pero esto no evita hacer posible el acceso desde otro lugar como al presionar APPS, STAT, NUM SLV, etc.

**Config**: Este objeto se encarga de configurar la biblioteca. Este objeto es evaluado en cada warmstart. Normalmente, es un programa que adjunta la biblioteca al directorio HOME. Se usa de la siguiente manera: escribe un programa en el editor y escribe el nombre de este programa en el campo Config.

Si dejas este campo en blanco, no hay ningún problema, pues en cada warmstart se ejecutará el programa por defecto, por ejemplo,

:: HOMEDIR 1000 XEQSETLIB ;

el cual vincula la biblioteca al directorio HOME.

**Message Text**: Haciendo doble clic en este campo, podrás escribir los mensajes de error de la biblioteca. Si cada mensaje es usado sólo una vez, no vale la pena crear una tabla de mensajes. Pero si los mensajes son usados en muchos lugares, o si quieres que sea fácil cambiar el idioma de la biblioteca, entonces sería muy útil crear una tabla de mensajes.

Cada mensaje de error debe ir en sólo una línea. Puedes poner caracteres en formato hexadecimal. Por ejemplo, si un mensaje de error lleva varios renglones, puedes usar el salto de línea escribiendo \0A como se muestra en la figura.

La lista puede contener como máximo a 255 cadenas. En la calculadora cada mensaje es identificado como un bint único. Por ejemplo, si el número de la biblioteca es 1000 (3E8 en hexadecimal), podrás acceder a un mensaje desde un programa usando:

# 3E801 JstGETTHEMSG

# 3E802 JstGETTHEMSG

# 3E803 JstGETTHEMSG

.........

# 3E8FF JstGETTHEMSG

Para generar un mensaje de error mostrando una de esas cadenas desde un programa, puedes usar:

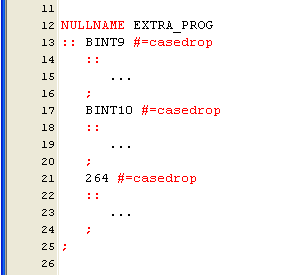
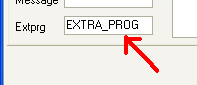
# 3E801 ERROROUT

O también:

# 3E801 DO#EXIT

En el capítulo 23 puedes ver más información sobre esto.

No hay ningún problema si este campo queda en blanco.

**Extprg**: Este es el message handler de la biblioteca. Se encarga de mostrar la biblioteca o partes de ella dentro de algunos menús (APPS, STAT, etc), mostrar ayuda de los comandos de la biblioteca en el catálogo, y otras cosas. Se usa de la siguiente manera: escribe un programa en el editor y escribe el nombre de este programa en el campo Extprg.

En la siguiente sección se explica detalladamente a este programa. No hay ningún problema si este campo queda en blanco.



**Slot**: Aquí debemos colocar el número de puerto (0,1,2) donde queremos que se instale la biblioteca. Por ejemplo para que se instale en el puerto cero, escribimos en el campo Slot el número 0.

En el editor de Debug 4x:

a) Comandos visibles se crean de esta manera:

\* Explicación opcional

xNAME NombreCom ( ... -> ... )

::

\* Aquí el contenido

;

b) Y se llaman anteponiendo la letra x:

...

xNombreCom

...

c) Comandos invisibles se crean de esta manera:

\* Explicación opcional

NULLNAME NombreSub ( ... -> ... )

::

\* Aquí el contenido

;

d) Y se llaman simplemente escribiendo su nombre:

...

NombreSub

...

e) Comandos visibles no permitidos en algebraicos se crean de esta manera:

\* Explicación opcional

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME NombreCom ( ... -> ... )

::

\* Aquí el contenido

;

En los documentos complementarios que vienen con el archivo de descarga se muestra como ver el diagrama de pila de cada comando al escribirlo en el editor de Debug 4x, como se autocompletan los nombres de los comandos, como insertar un grob en el editor, como depurar errores, etc.

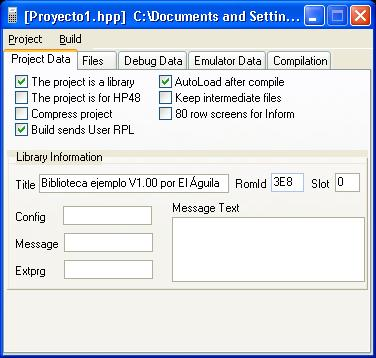
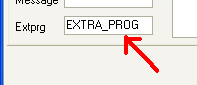
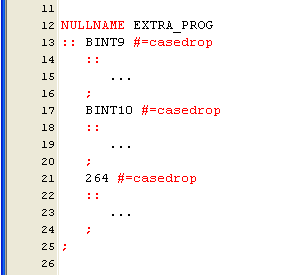
B.3 El Message Handler de la Biblioteca

Las bibliotecas pueden contener un message handler. Este programa es llamado por el sistema operativo en varias ocasiones, con el fin de que la biblioteca pueda modificar menús, proveer ayuda online para sus comandos o hacer otras acciones.

A) Cuando se crea una biblioteca desde un directorio (en la calculadora), la variable reservada puede contener un nombre global. Este es el nombre de una variable en el directorio, el cual más tarde será un rompointer en la biblioteca. Esta variable debe contener un **programa**.

B) Cuando se crea una biblioteca desde Debug 4x, en la ventana **Project/Project Data/Library Information** en el campo **ExtPrg** se puede colocar el nombre de un comando de la biblioteca. Este comando debe contener un programa.

Este programa debe aceptar un bint en el nivel 1 de la pila y dependiendo del mensaje específico, otros argumentos en los otros niveles de la pila.



B.3.1 Extensiones de Menús

La mayoría de mensajes pueden ser usados para extender algunos menús propios de la calculadora. Entre estos menús figuran el menú APPS, varios otros menús, el SEARCH, GOTO, submenús de herramientas (TOOLS) en el menú del editor, etc. Cuando el mensaje es llamado para extender un menú, la forma predefinida de ese menú está ya en la pila sea como una lista o como un meta. El programa (message handler) puede entonces modificar este menú y retornarlo en la pila.

Por lo tanto, el diagrama de pila para las extensiones de estos menús es:

( {key1...keyN} #msg 🡺 lista\_modificada #msg )

( key1...keyN #n #msg 🡺 meta\_modificado #msg )

El bint que representa el número de mensaje debe permanecer en la pila, para que el message handler de otra biblioteca pueda ser llamado inmediatamente después para que haga sus modificaciones respectivas.

Los siguientes menús puedes ser extendidos usando mensajes de biblioteca:

**#msg Menú Tipo de Menu**

0 APPS lista

1 STAT lista

2 Pruebas de Hipótesis (dentro de STAT) lista

3 Intervalos de confianza (dentro de STAT) lista

5 NUMERIC SOLVER lista

6 TIME lista

8 GAMES (dentro de APPS) meta

11 Editor SEARCH (flag -117 desactivado) lista

12 Editor TOOLS (flag -117 desactivado) lista

13 Editor GOTO (flag -117 desactivado) lista

14 Editor SEARCH (flag -117 activado) meta

15 Editor TOOLS (flag -117 activado) meta

16 Editor GOTO (flag -117 activado) meta

**Ejemplo de modificación del menú APSS**

NULLNAME EXTRA\_PROG ( lista/meta # -> lista'/meta' # )

:: ( ... # )

BINT0 #=casedrop

:: ( {{$ ob}} )

DUPLENCOMP ( {{$ ob}} #n )

#1+ ( {{$ ob}} #n+1 )

#>$ ( {{$ ob}} "n+1" )

tok. ( {{$ ob}} "n+1" "." )

&$ ( {{$ ob}} "n+1." )

"DESCRIPCIÓN" ( {{$ ob}} "n+1." "DESCRIPCIÓN" )

&$ ( {{$ ob}} $ )

' :: "Hola" FlashWarning ; ( {{$ ob}} $ prog )

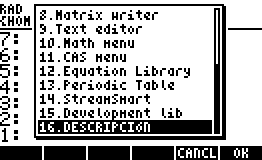
TWO{}N ( {{$ ob}} {$ prog} )

>TCOMP ( {{$ ob}}' )

BINT0 ( {{$ ob}}' #0 )

;

;



**Ejemplo de modificación del menú STAT**

NULLNAME EXTRA\_PROG ( lista/meta # -> lista'/meta' # )

:: ( ... # )

BINT1 #=casedrop

:: ( {{$ ob}} )

DUPLENCOMP ( {{$ ob}} #n )

#1+ ( {{$ ob}} #n+1 )

#>$ ( {{$ ob}} "n+1" )

tok. ( {{$ ob}} "n+1" "." )

&$ ( {{$ ob}} "n+1." )

"DESCRIPCIÓN" ( {{$ ob}} "n+1." "DESCRIPCIÓN" )

&$ ( {{$ ob}} $ )

' :: "Hola" FlashWarning ; ( {{$ ob}} $ prog )

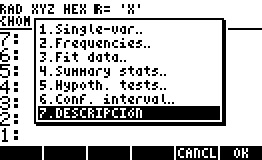
TWO{}N ( {{$ ob}} {$ prog} )

>TCOMP ( {{$ ob}}' )

BINT1 ( {{$ ob}}' #1 )

;

;



**Ejemplo de modificación del menú GAMES**

NULLNAME EXTRA\_PROG ( lista/meta # -> lista'/meta' # )

:: ( ... # )

BINT8 #=casedrop

:: ( ob1...obn #n )

#1+DUP ( ob1...obn #n+1 #n+1 )

#>$ ( ob1...obn #n+1 "n+1" )

tok. ( ob1...obn #n+1 "n+1" "." )

&$ ( ob1...obn #n+1 "n+1." )

"DESCRIPCIÓN" ( ob1...obn #n+1 "n+1." "DESCRIPCIÓN" )

&$ ( ob1...obn #n+1 $ )

' :: "Hola" FlashWarning ; ( ob1...obn #n+1 $ prog )

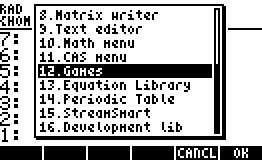
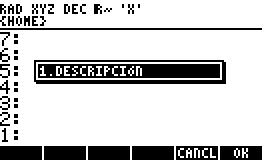
TWO{}N ( ob1...obn #n+1 {$ prog} )

SWAP ( ob1...obn,obn+1 #n+1 )

BINT8 ( meta #8 )

;

;

**Ejemplo de modificación de 2 menús**

NULLNAME EXTRA\_PROG ( lista/meta # -> lista'/meta' # )

:: ( ... # )

BINT0 #=casedrop

:: ( {{$ ob}} )

DUPLENCOMP ( {{$ ob}} #n )

#1+ ( {{$ ob}} #n+1 )

#>$ ( {{$ ob}} "n+1" )

tok. ( {{$ ob}} "n+1" "." )

&$ ( {{$ ob}} "n+1." )

"DESCRIPCIÓN" ( {{$ ob}} "n+1." "DESCRIPCIÓN" )

&$ ( {{$ ob}} $ )

' :: "Hola" FlashWarning ; ( {{$ ob}} $ prog )

TWO{}N ( {{$ ob}} {$ prog} )

>TCOMP ( {{$ ob}}' )

BINT0 ( {{$ ob}}' #0 )

;

BINT8 #=casedrop

:: ( ob1...obn #n )

#1+DUP ( ob1...obn #n+1 #n+1 )

#>$ ( ob1...obn #n+1 "n+1" )

tok. ( ob1...obn #n+1 "n+1" "." )

&$ ( ob1...obn #n+1 "n+1." )

"DESCRIPCIÓN" ( ob1...obn #n+1 "n+1." "DESCRIPCIÓN" )

&$ ( ob1...obn #n+1 $ )

' :: "Hola" FlashWarning ; ( ob1...obn #n+1 $ prog )

TWO{}N ( ob1...obn #n+1 {$ prog} )

SWAP ( ob1...obn,obn+1 #n+1 )

BINT8 ( meta #8 )

;

;

**Ejemplo de modificación del menú TOOLS**

NULLNAME EXTRA\_PROG ( lista/meta # -> lista'/meta' # )

:: ( ... # )

BINT12 #=casedrop

:: ( {{$ ob}} )

DUPLENCOMP ( {{$ ob}} #n )

#1+ ( {{$ ob}} #n+1 )

#>$ ( {{$ ob}} "n+1" )

tok. ( {{$ ob}} "n+1" "." )

&$ ( {{$ ob}} "n+1." )

"DESCRIPCIÓN" ( {{$ ob}} "n+1." "DESCRIPCIÓN" )

&$ ( {{$ ob}} $ )

' :: TakeOver ( )

SELECT.LINE ( ) ( Selecciona la línea actual )

CMD\_COPY ( ) ( Copia al portapapeles )

; ( {{$ ob}} $ prog )

TWO{}N ( {{$ ob}} {$ prog} )

>TCOMP ( {{$ ob}}' )

BINT12 ( {{$ ob}}' 12 )

;

BINT15 #=casedrop

:: ( ob1...obn #n )

{ "E.L.A"

{ :: TakeOver ( )

SELECT.LINE ( ) ( Selecciona la línea actual )

CMD\_COPY ( ) ( Copia al portapapeles )

;

Modifier

:: TakeOver

"Selecciona la línea actual y la copia al portapapeles"

FlashWarning

;

}

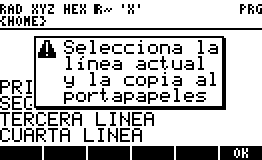
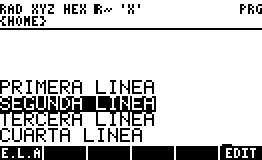
}

SWAP#1+ ( meta 15 )

BINT15 ( meta 15 )

;

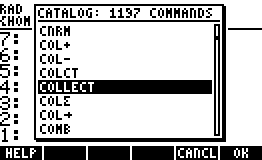
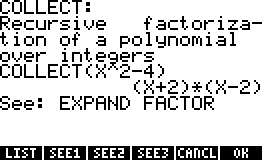
;

B.3.2 Ayuda en Línea para Comandos de Biblioteca

En la HP49G, todos los comandos CAS tienen un texto de ayuda corto el cual puede ser mostrado desde el catálogo.

Cuando en el CHOOSEBOX del catálogo es seleccionado un comando del CAS, el menú debajo del CHOOSEBOX tiene un botón adicional, el botón HELP. Al presionar ese botón se mostrará el texto de ayuda correspondiente al comando.

Las bibliotecas que creamos pueden proveer ayuda para sus propios comandos de una manera similar, usando los message handlers número 9 y 10.

El **mensaje 9** es para decidir si se mostrará o no el botón de ayuda correspondiente al comando rompointer. El diagrama de pila es:

( romptr FALSE #9 🡺 romptr TRUE/FALSE #9 )

Donde el TRUE/FALSE retornado en el nivel 2 de la pila indica si la biblioteca está preparada para mostrar la ayuda correspondiente al rompointer del nivel 3. Este mensaje es usado para determinar si el botón HELP en el catálogo debe mostrarse o no.

El **mensaje 10** es usado para mostrar realmente la ayuda cuando el usuario presiona la tecla de menú HELP. El diagrama de pila es:

( romptr TEN 🡺 FALSE )

Antes de colocar FALSE, el message handler debe mostrar el texto de ayuda correspondiente al comando rompointer.

El siguiente ejemplo es un message handler que provee un texto de ayuda corto para cada comando visible de la biblioteca.

NULLNAME EXTRA\_PROG

:: BINT9 #=casedrop

:: ( romptr F )

DROPTRUE ( romptr T )

BINT9 ( romptr T 9 ) ( todos los comandos tendrán ayuda )

;

BINT10 #=casedrop

:: ( romptr )

DUP ( romptr romptr )

DECOMP$ ( romptr "NombRomp" )

NEWLINE&$ ( romptr "NombRomp\0A" )

SWAP ( "NombRomp\0A" romptr )

ROMPTR># ( "NombRomp\0A" #lib #cmd )

SWAPDROP ( "NombRomp\0A" #cmd ) ( #cmd: 0,1,2... )

{ "Help text for romptr 0"

"Help text for romptr 1"

...

"Help text for romptr N"

}

( "NombRomp\0A" #cmd {$} )

SWAP#1+ ( "NombRomp\0A" {$} #cmd+1 ) ( #cmd+1: 1,2,3... )

NTHCOMPDROP ( "NombRomp\0A" $Help )

&$ ( "NombRomp\0AHelp" )

FALSE ( "NombRomp\0AHelp" F )

SWAP ( F "NombRomp\0AHelp" )

ViewStrObject ( F )

;

;

Puedes notar que el comando ViewStrObject coloca convenientemente FALSE en la pila, el cual es el valor requerido por el mensaje 10.

El message handler se vuelve un poco más complicado, si la ayuda sólo es proporcionada para unos pocos rompointers.

En este caso, el mensaje número 9 puede verificar el rompointer en una lista, y el mensaje 10 podría usar el comando Lookup o algo parecido para extraer el texto de ayuda.

En vez de sólo mostrar una cadena, el mensaje 10 puede también hacer cosas más complicadas, como ejecutar una aplicación que muestre ayuda de manera más detallada.

En el siguiente ejemplo, el message handler mostrará la ayudá sólo para algunos comandos de la biblioteca (comandos de números 0, 3 y 4).

NULLNAME EXTRA\_PROG

:: BINT9 #=casedrop

:: ( romptr F )

DROPDUP ( romptr romptr )

ROMPTR># ( romptr #lib #cmd )

SWAPDROP ( romptr #cmd )

LISTA\_HELPS ( romptr #cmd {} )

FLASHPTR ListPos ( romptr #i/#0 )

#0<> ( romptr T/F )

BINT9 ( romptr T/F 9 )

;

BINT10 #=casedrop

:: ( romptr )

DUP ( romptr romptr )

DECOMP$ ( romptr "NombRomp" )

NEWLINE&$ ( romptr "NombRomp\0A" )

SWAP ( "NombRomp\0A" romptr ) ( save cmd name as string )

ROMPTR># ( "NombRomp\0A" #lib #cmd )

SWAPDROP ( "NombRomp\0A" #cmd ) ( #cmd: 0,1,2... )

LISTA\_HELPS ( "NombRomp\0A" #cmd {} )

EQLookup ( "NombRomp\0A" $ T // "NombRomp\0A" #cmd F )

NOTcase2drop

:: DoBadKey ( )

FALSE ( F )

;

( "NombRomp\0A" $ )

&$ ( "NombRomp\0AHelp" )

FALSE ( "NombRomp\0AHelp" F )

SWAP ( F "NombRomp\0AHelp" )

ViewStrObject ( F )

;

;

NULLNAME LISTA\_HELPS

{ BINT0 "Help text for romptr 0"

BINT3 "Help text for romptr 3"

BINT4 "Help text for romptr 4"

}

B.3.3 El Mensaje que Maneja el Menú de Bibliotecas

Si el menu de una biblioteca es invocado a través del menú LIBS (Shift derecho+2), el número romid de la biblioteca es enviado al message handler de la biblioteca. Se puede hacer cualquier cosa aquí, por ejemplo, mostrar en pantalla un texto de bienvenida, hacer algo divertido con el menú o hacer sonar una melodía. También podrías cambiar la configuración del menú, por ejemplo, proporcionar funcionalidad a los botones Shift+tecla de menú, tal como hace la biblioteca Comandos de César Vásquez.

El diagrama de pila para este mensaje es:

( #romid 🡺 #romid )

El siguiente ejemplo es el message handler de una biblioteca cuyo número es 1000.

Este programa muestra un texto en la parte superior de la pantalla cuando la biblioteca es seleccionada desde el menú LIBS.

NULLNAME EXTRA\_PROG

:: 1000 #=casedrop

:: ( #romid )

ZEROZERO ( #romid #0 #0 )

"(c) 2011 Balán Gonzales\0A" ( #romid #0 #0 $ )

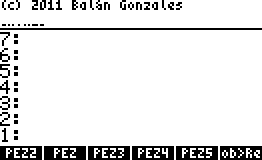
$>grobCR ( #romid #0 #0 grob )

XYGROBDISP ( #romid )

SetDA1Temp ( #romid )

;

;



Apéndice C  
Pasos Para Crear Bibliotecas con Debug 4x

Debug 4x permite hacer programas para la calculadora HP desde tu ordenador.

Con Debug 4x puedes hacer programas en User RPL, System RPL y en lenguaje ensamblador.

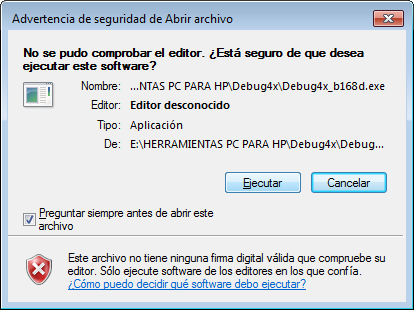
Si deseas escribir programas en System RPL la mejor manera de hacerlo es usando Debug 4x. Con Debug 4x puedes poner comentarios, poner en cada línea el diagrama de pila, depurar el código de un programa, ver en cada momento el contenido de las nombres locales y globales, el contenido de la pila virtual, la pila de retornos, probar los programas en el emulador y mucho más.

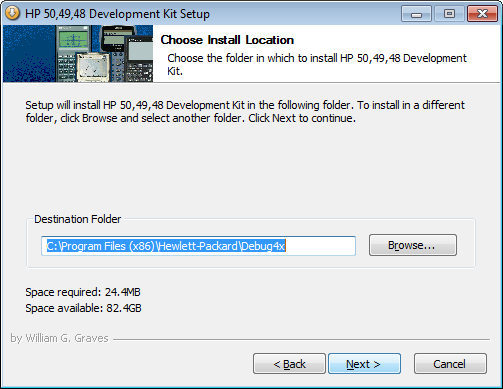
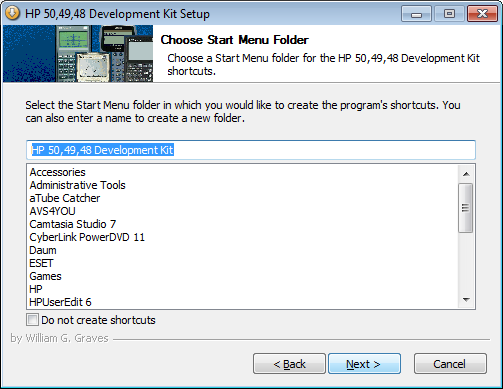
C.1 Instalar Debug 4x

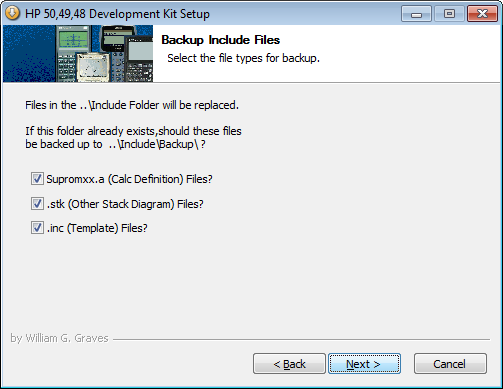
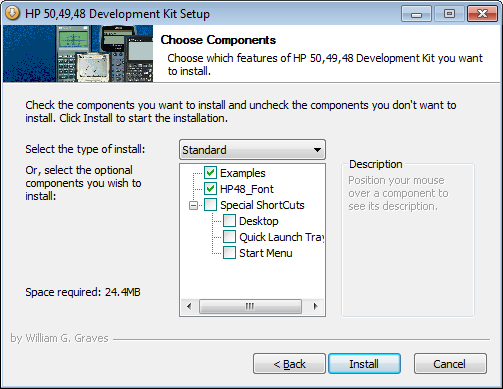
Debug 4x es totalmente gratis y puedes descargar la última versión desde la página de Bill Graves:

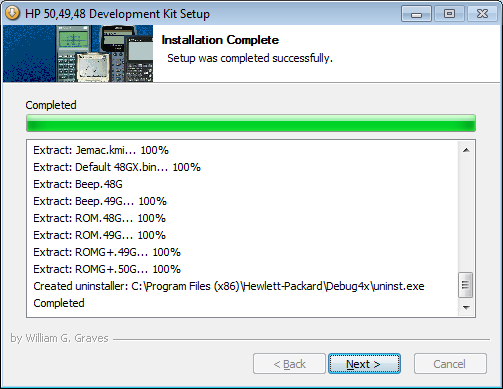
<http://www.debug4x.com/>

Sigue los pasos de instalación de Debug 4x.

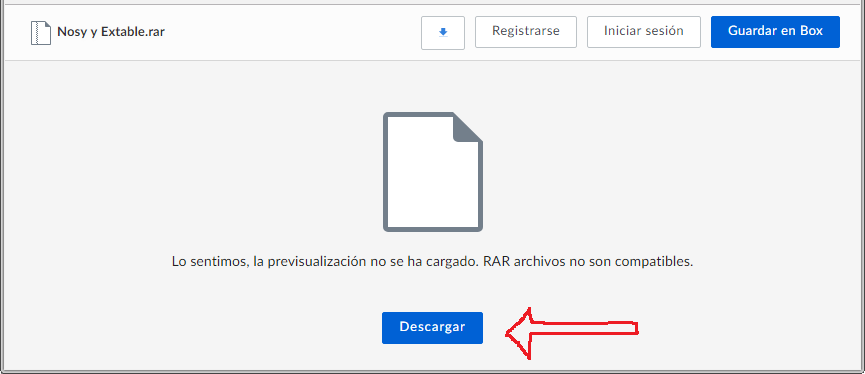
 

Es preferible que instales en tu calculadora y en el emulador estas 2 bibliotecas: Nosy y Extable:

<http://www.box.net/shared/y8jnglfyel>



C.2 Ventana Principal (Main Window)

Una vez que has instalado Debug 4x podrás ejecutarlo. Al hacerlo se abrirá la ventana principal (**Main Window**).



Desde la ventana principal se pueden abrir muchas otras, pero sólo tres son necesarias para hacer programas en System RPL y depurarlos:

1) **Project**

2) **Editor**

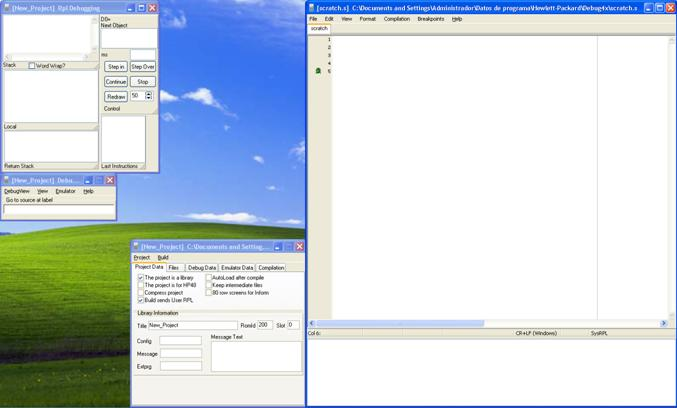
3) **Debugger RPL**

Abre esas 3 ventanas de la siguiente manera:

|  |  |  |
| --- | --- | --- |
| **Project** | **Editor** | **Debugger RPL** |
| Muestra la ventana de  Proyecto. | Muestra el Editor de código  fuente con resaltado de  sintaxis. | Muestra una ventana desde  donde se puede depurar  programas System RPL. |
|  |  |  |
|  |  |  |

C.3 Ordena las ventanas en la pantalla

Ahora debemos ordenar las ventanas en la pantalla del ordenador. El hueco dejado será para colocar más tarde el emulador.

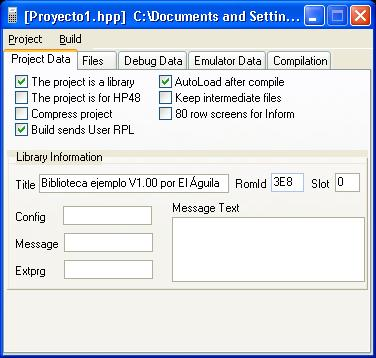


C.4 Crea un nuevo proyecto

|  |  |
| --- | --- |
| Ahora crearemos un nuevo proyecto.  Para esto ir a **Project** y seleccionar  **New Proyect** de la siguiente manera: | -Ir a la ruta que queramos.  -Crear una nueva carpeta que contendrá al  proyecto y sus archivos.  Por ejemplo llamémosla "carpeta proyecto 1".  -Entrar a esta carpeta y poner el nombre del proyecto. Por ejemplo lo llamaremos "Proyecto1" |
|  |  |

Listo. Ya tienes creado tu proyecto.

C.5 Selecciona opciones para tu proyecto.

Debemos seleccionar opciones para el proyecto en la ventana **Project**.

-Seleccionar The proyect is a library para que el proyecto sea una biblioteca y no sólo un programa.

-Seleccionar AutoLoad after compile para que al compilar la biblioteca esta se instale automáticamente en el emulador.

-Poner el nombre de la biblioteca a mostrarse en la calculadora. Para esto en el campo Title escribimos: Biblioteca ejemplo V1.00 por El Águila.

-Poner el número de la biblioteca. Este número debe estar en el rango 769-1791. Pero debemos colocarlo en hexadecimal (entre 301 y 6FF). Por ejemplo para que la biblioteca tenga el número 1000 en el campo RomId escribimos el número 3E8

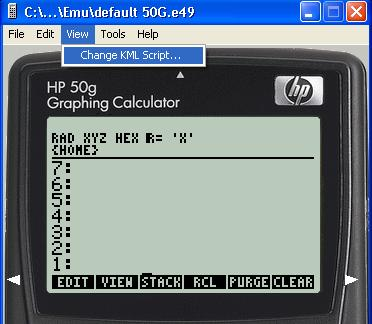
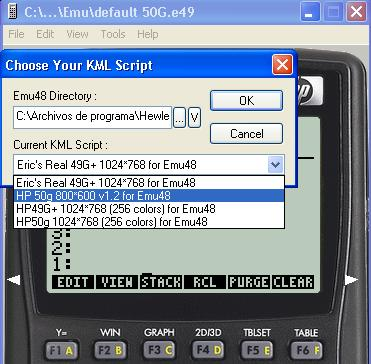
-Colocar el número de puerto (0,1,2) donde queremos que se instale la biblioteca. Por ejemplo para que se instale en el puerto cero, escribimos en el campo Slot el número 0

C.6 Abre el emulador que usará tu proyecto

Debug 4x trae incorporados varios emuladores. Solo debemos abrirlo desde la ventana principal (**Main Window**) de la siguienta manera:



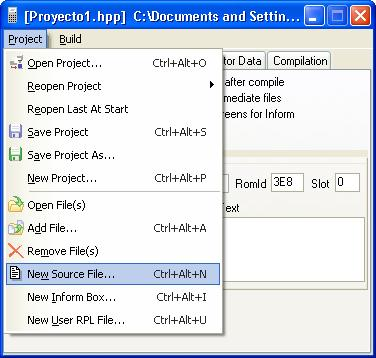
Seguramente se abrirá un emulador muy grandote. Para cambiarlo a uno más pequeño seleccionar en el emulador View - Change KML Script y a continuación seleccionar un emulador más pequeño, de tamaño 800x600 (puede ser de hp 49G, hp 49g+ o hp 50g). Esto se resume en estas imágenes:

C.7 Crea Archivo Fuente para tus programas

Ahora crearemos un archivo fuente (Source File) para los programas de tu biblioteca.

Para esto escoger en la ventana **Project** la opción New Source File.... (También se puede escoger esta opción desde la ventana Editor).



|  |  |
| --- | --- |
| No cambiarse de carpeta.  Dar un nombre al archivo fuente  (Source File).  Nosotros lo llamaremos Principal. | El archivo fuente (Source File) recien  creado tiene el siguiente aspecto en  el **editor**: |
|  |  |

- En un proyecto puedes agregar más archivos fuente (source files). Esto es de gran ayuda si haces bibliotecas grandes.

C.8 Tu primer programa en System RPL.

Ahora ya estamos listos para crear programas System RPL en la biblioteca.

|  |  |
| --- | --- |
| Copia el siguiente código y pegalo en tu  archivo fuente: | El archivo fuente se verá así: |
| xNAME PRIMER  :: CK1&Dispatch  BINT1  :: "Es un número real"  SUBPROGRAMA  ;  BINT2  :: "Es un número complejo"  SUBPROGRAMA  ;  ;  NULLNAME SUBPROGRAMA  :: FlashWarning  % 500 % .05 xBEEP  ; |  |

A continuación presionar **F9** lo cual guarda los cambios, compila el programa e instala la biblioteca en el emulador. Esto tarda unos dos segundos:

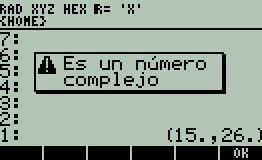


Luego ya tienes instalada la biblioteca en el emulador.

- Vemos que al anteponer xNAME el comando será visible.

- Vemos que al anteponer NULLNAME el comando será invisible.

A continuación se muestra la ejecución del comando PRIMER de la biblioteca creada cuando en la pila se encontraba un número complejo.

Listo, ya tienes funcionando tu primera biblioteca hecha con Debug 4x en System RPL para calculadoras Hp 49 series.

C.9 Uso de la ventana Debugger RPL para depurar programas en System RPL

La ventana del depurador (Debugger RPL), permite depurar o limpiar los posibles errores de la biblioteca.

|  |  |
| --- | --- |
| Crearemos otro programa en nuestra biblioteca. Copiar y pegar en tu editor el siguiente código que permite hallar el discriminante de una ecuación cuadrática teniendo como datos los coeficientes en la pila: | En la ventana del editor picar en la parte izquierda del renglón donde queramos que empiece la depuración. Aparecerá el punto de ruptura o breakpoint como una bolita roja. Se verá así: |
| xNAME DISCR  :: CK3&Dispatch  3REAL  :: { LAM A LAM B LAM C }  BIND  LAM B  %SQ\_  %4  LAM A  %\*  LAM C  %\*  %-  "Discr"  >TAG  ABND  ;  ; |  |

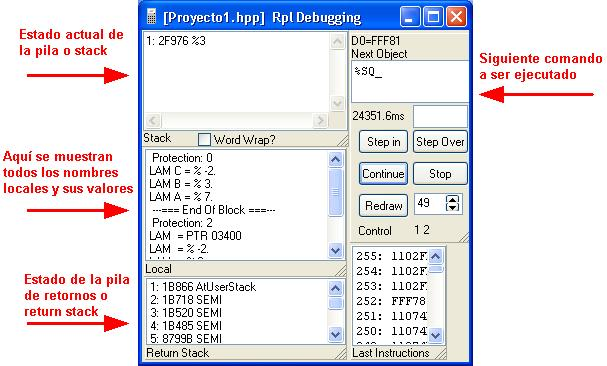
Presionar **F9** para guardar, compilar e instalar la biblioteca en el emulador.

A continuación, en el emulador colocar los datos en la pila y ejecutar el comando recién creado. El comando DISCR



Como hemos puesto un breakpoint en el editor entonces el programa se detendrá en este.

En la ventana Debugger RPL podemos ver el estado de la calculadora en ese momento:



Se ven unos botones:

**Step in**: Se usa para ir paso a paso a través de un programa. El siguiente comando se ejecuta paso a paso.

**Step Over**: Se usa para ir paso a paso a través de un programa. El siguiente comando se ejecuta inmediatamente.

**Continue**: Para continuar la ejecución del programa hasta su finalización o hasta encontrar el siguiente breakpoint.

Los valores que se muestran en esta ventana Debugger RPL se actualizan mientras vamos ejecutando paso a paso el programa.

Apéndice D  
Preguntas Frecuentes Sobre Debug 4x y System RPL

D.1 Ver Argumentos y Resultados de Comandos en Debug 4x

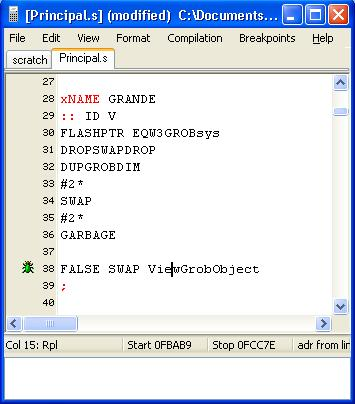
PREGUNTA:

Cuando escribo un comandos en Debug 4x.

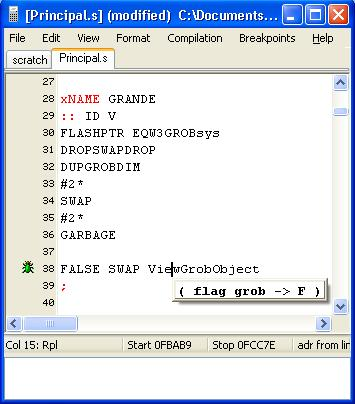
¿Se pueden ver los argumentos que requiere el comando y también los resultados que devuelve el comando?

RESPUESTA:

Primero, haz clic sobre el comando cuyos argumentos y resultados quieres conocer.

****

Luego presiona CTRL + J

****

En este ejemplo la información en la pantalla dice que el comando ViewGrobObject necesita dos argumentos:

- Un flag (TRUE o FALSE)

- Un grob (objeto gráfico)

El comando devuelve un objeto en la pila

- El flag FALSE

D.2 Autocompletar comandos en Debug 4x

PREGUNTA:

¿Es posible autocompletar el nombre de un comando cuando programo en Debug 4x, es decir que al escribir una parte del comando la máquina escriba el resto del comando?

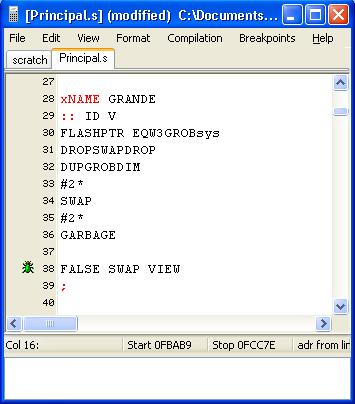
RESPUESTA:

Si, si se puede. Uno no puede recordar todos los comandos del System Rpl de la calculadora, pues son muchos.

Pero si tu recuerdas el comienzo del nombre del comando, la COMPLETACIÓN DINÁMICA de Debug 4X puede encontrar el resto del nombre por ti.

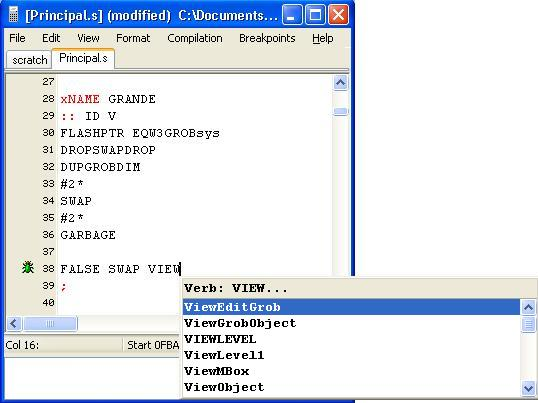
Primero, escribe la parte inicial del comando que deseas en el editor.

En este ejemplo escribimos VIEW



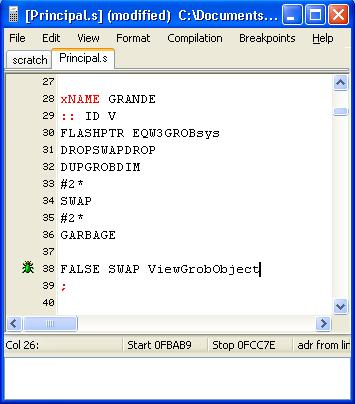
A continuación presionar CTRL + Barra Espaciadora.

Verás todos los comandos que **EMPIEZAN** con las letras que has escrito.

****

Ahora escoge el comando que deseas y presiona ENTER.

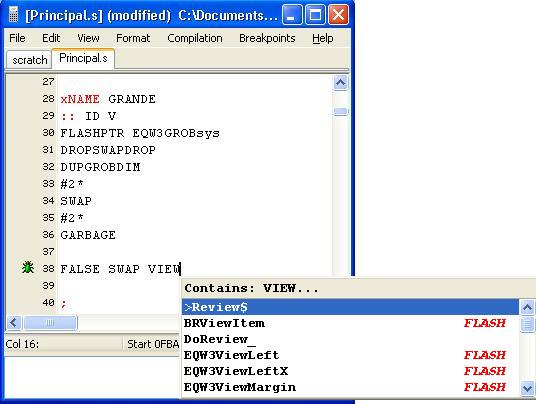
El comando ahora se ha escrito correctamente en la Ventana Editor.

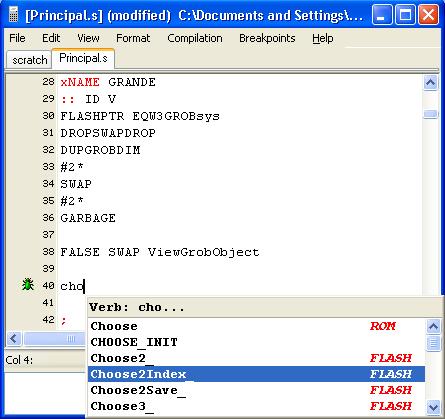


También se puede escoger entre todos los comandos que **CONTIENEN** a la palabra seleccionada.

En este caso luego de escribir la palabra debes presionar

ALT + CTRL + Barra Espaciadora



Los caracteres especiales ~ y ^ con los cuales comienza un ROMPTR o FLASHPTR no son reconocidos por Debug 4x.

En su lugar tu debes escribir antes del comando la palabra ROMPTR o FLASHPTR.

Por ejemplo, para estos comandos:

Debes escribirlos de la siguiente manera:

ROMPTR Choose

CHOOSE\_INIT

FLASHPTR Choose2\_

FLASHPTR Choose2Index\_

FLASHPTR Choose2Save\_

FLASHPTR Choose3\_

D.3 Comandos Permitidos y no Permitidos en Algebraicos con Debug4x

PREGUNTA:

En el manual de la calculadora, se ve que hay dos tipos de comandos en la HP.

1) Comandos permitidos en algebraicos (llamados también funciones). Por ejemplo: SIN, COS, DARCY.

2) Comandos no permitidos en algebraicos. Por ejemplo, PICK, ROOT y LIST🡪.

Si escribo un comando permitido en algebraicos como DARCY en el EQW y abro paréntesis, de manera automática se ve en esta forma: DARCY(■,■). Además estos comandos pueden ser seleccionados en el EQW.

Con los comandos no permitidos en algebraicos como PICK, eso no es posible.

Mi pregunta es si se puede crear comandos de los dos tipos al hacer bibliotecas con Debug4x y como puedo hacer esto.

RESPUESTA:

A) Para crear un comando permitido en algebraicos:

\* Comentarios

xNAME NombreDelComando ( ... -> ... )

::

\* Aquí el contenido

;

B) Para crear un comando no permitido en algebraicos:

\* Comentarios

ASSEMBLE

CON(1) 8 \* Tell parser 'Non algebraic'

RPL

xNAME NombreDelComando ( ... -> ... )

::

\* Aquí el contenido

;

D.4 Insertar un Grob en el Editor de Debug4x

PREGUNTA:

En el editor de Debug 4x quiero copiar la representación System RPL de un grob

Es decir, como escribo un grob como este

GROB 10 21 256546545644...

del User RPL en el editor de Debug 4x.

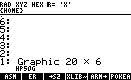
Se que en System RPL, cambia la escritura del grob, algo asi

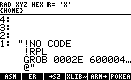
GROB 0003C 654654654654

¿Cómo se obtiene su equivalente listo para copiar al editor sin la necesidad de llamarlo como un id, sino con su escritura en System RPL.

RESPUESTA:

Sigue los siguientes pasos:



* En el emulador debes de tener el grob en la pila.
* En el emulador ejecuta el comando 🡪S2 de la biblioteca 256.

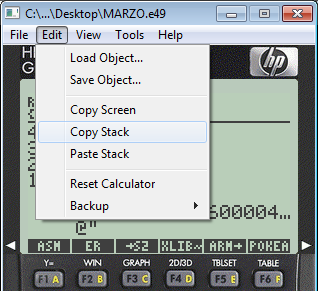
En la pila se obtiene una cadena de esta forma:

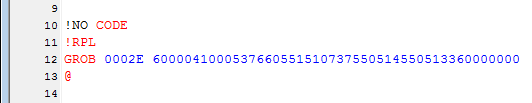
"!NO CODE

!RPL

GROB 0002E 6000041000537660551510737550514550513360000000

@"



* En el emulador, en la barra de menús, presiona Edit, luego de la lista desplegable escoge Copy Stack. Eso copiará la cadena en el portapapeles de tu PC.
* En el Editor de Debug4x presiona CTRL+V.

La cadena ahora aparece en el editor de Debug4x.

* Borra la 1º, 2º y 4º lineas de la cadena pegada para que tengas solamente el grob en el editor de Debug4x:

GROB 0002E 6000041000537660551510737550514550513360000000

D.5 Más Preguntas y Respuestas

Para ver más preguntas y respuestas sobre Debug 4x y System RPL visita el foro Honrados HP en la siguiente dirección:

[www.honradoshp.com](http://www.honradoshp.com)

Apéndice E  
Comandos de User RPL

La lista mostrada en este capítulo es de todos los comandos accesibles al usuario (excepto 3 comandos ', DIR y GROB), con sus direcciones. En la mayoría de los casos, el nombre User RPL de un comando es igual al nombre System RPL que puedes colocar en el editor de Debug4x con una x adelante y a veces también con ˜ adelante. Las pocas excepciones también se indican.

En la versión 2.15, el número de comandos User RPL es **771** contando a los comandos de la biblioteca 227 (EQLIB: Equation Lib) y de la biblioteca 229 (PRTBL: Periodic Table) y sin contar a los comandos de las bibliotecas 256 y 257.

Por Ejemplo, el comando →ROW de User RPL en System RPL (editor de Debug 4x) se llama como:

ROMPTR x\->ROW

Si deseas llamar a ese comando sin guardarlo como el comando actual y sin que quede marcada la pila, debes usar el comando 'EvalNoCK: (capítulo 30) antes del comando de User RPL. Por ejemplo asi:

...

'EvalNoCK:

ROMPTR x\->ROW

...

Mejor aun, si deseas evitar la verificación de argumentos, puedes usar la biblioteca Nosy para ver cual es la entrada interna de ese comando User RPL en System RPL. Para esto, coloca en el nivel 1 de la pila la lista: {→ROW}. Luego ejecuta el comando Nosy de la biblioteca Nosy y verás la siguiente pantalla:



La cual indica que para ejecutar lo mismo que el comando →ROW pero sin verificación de argumentos es suficiente escribir en el editor de Debug 4x:

...

FLASHPTR 003 065

...

El BINT4 en la pantalla indica que el comando →ROW espera un arreglo o una matriz simbólica en el nivel 1 de la pila.

E.1 Referencia

**Direcc. Nombre Descripción**

3ABAF xFACT ( x 🡺 x' )

001: !

3B251 x% ( x y 🡺 xy/100 )

002: %

3B362 x%CH ( x1 x2 🡺 x3 )

003: %CH

3B2DC x%T ( x y 🡺 100y/x )

004: %T

39DE8 x\* ( x y 🡺 x\*y )

006: \*

39B58 x+ ( x y 🡺 x+y )

007: +

39CFC x- ( x y 🡺 x-y )

008: -

39F49 x/ ( x y 🡺 x/y )

009: /

3F053 x; ( ob 🡺 )

( 🡺 )

010: ;

3CE42 x< ( x y 🡺 1 )

011: <

398B9 x= ( x y 🡺 x=y )

012: =

3CBF6 x== ( x y 🡺 1 )

( x y 🡺 0 )

013: ==

3CEE1 x> ( x y 🡺 1 )

( x y 🡺 0 )

014: >

089314 ˜x? ( 🡺 ob )

015: ?

030314 ˜xABCUV ( pa pb c 🡺 u v )

016: ABCUV

39A07 xABS ( x 🡺 x' )

017: ABS

390E4 xACK ( 🡺 )

018: ACK

390C9 xACKALL ( 🡺 )

019: ACKALL

3A7DC xACOS ( x 🡺 x' )

020: ACOS

025314 ˜xACOS2S ( symb 🡺 symb' )

021: ACOS2S

3A8D8 xACOSH ( x 🡺 x' )

022: ACOSH

05C0AB xADD ( {} {}' 🡺 {}'' )

( {} ob 🡺 {}' )

( ob {} 🡺 {}' )

023: ADD

**Direcc. Nombre Descripción**

06E314 ˜xADDTMOD ( symb1 symb2 🡺 symb3 )

024: ADDTMOD

0000DE xADDTOREAL ( var 🡺 )

025: ADDTOREAL

080314 xALGB ( 🡺 )

026: ALGB

aka: ˜xBASE

3AAE5 xALOG ( x 🡺 x' )

027: ALOG

04B0AB xAMORT ( n 🡺 princ intr bal )

028: AMORT

3CA07 xAND ( x1 x2 🡺 x3 )

029: AND

0140AB ( grob1...grobn n 🡺 SameStack )

( grob1...grobn {n {x y} delay rep} 🡺 SameStack )

030: ANIMATE

3F033 xANS ( n 🡺 ob )

031: ANS

3D7AC xAPPLY ( {symb1 .. symbn} f 🡺 f(symb1...symbn) )

032: APPLY

3C8C6 xARC ( c r θ1 θ2 🡺 )

( {#x #y} #r θ1 θ2 🡺 )

033: ARC

3EAC7 xARCHIVE ( :port:name 🡺 )

( :IO:name 🡺 )

034: ARCHIVE

3A390 xARG ( c 🡺 θ )

035: ARG

085314 ˜xARIT ( 🡺 )

036: ARIT

3BEC5 xARRY> ( [] 🡺 x1...xn {n} )

( [[]] 🡺 x11...xnm {n m} )

037: ARRY🡪

3A756 xASIN ( x 🡺 x' )

038: ASIN

024314 ˜xASIN2C ( symb 🡺 symb' )

039: ASIN2C

023314 ˜xASIN2T ( symb 🡺 symb' )

040: ASIN2T

3A88E xASINH ( x 🡺 x' )

041: ASINH

3EEE7 xASN ( obj key 🡺 )

( 'SKEY' 🡺 )

042: ASN

38DE1 xASR ( # 🡺 #' )

043: ASR

0260DE ( ob 🡺 ob' )

044: ASSUME

3A844 xATAN ( x 🡺 x' )

045: ATAN

022314 ˜xATAN2S ( symb 🡺 symb' )

046: ATAN2S

**Direcc. Nombre Descripción**

3A94F xATANH ( x 🡺 x' )

047: ATANH

0100AB ( x 🡺 )

( n 🡺 )

( {x y} 🡺 )

( {n m} 🡺 )

048: ATICK

3EB64 xATTACH ( n 🡺 )

049: ATTACH

0130DE ( list ob 🡺 ob' )

( vector ob 🡺 ob' )

( matrix ob 🡺 ob' )

( $ ob 🡺 ob' )

050: AUGMENT

3C49F xAUTO ( 🡺 )

051: AUTO

3C3B2 xAXES ( c 🡺 )

( {c tick $x $y } 🡺 )

052: AXES

04A314 ˜xAXL ( {} 🡺 [] )

( [] 🡺 () )

053: AXL

049314 ˜xAXM ( [A] 🡺 [M] )

054: AXM

04C314 ˜xAXQ ( [nxn] [n] 🡺 [nxn]' [n] )

055: AXQ

3C9D3 xBAR ( 🡺 )

056: BAR

3E196 xBARPLOT ( 🡺 )

057: BARPLOT

0110DE ( {vectors} 🡺 {vectors}' )

058: BASIS

3EDCC xBAUD ( n 🡺 )

059: BAUD

39765 xBEEP ( freq dur 🡺 )

060: BEEP

3E2C1 xBESTFIT ( 🡺 )

061: BESTFIT

3B655 xBIN ( 🡺 )

062: BIN

3E171 xBINS ( min width n 🡺 [[]] [] )

063: BINS

3C70A xBLANK ( #width #height 🡺 grob )

064: BLANK

3C6E0 xBOX ( {#n1 #m1} {#n2 #m2} 🡺 )

( c1 c2 🡺 )

065: BOX

3EE47 xBUFLEN ( 🡺 nchars 0/1 )

066: BUFLEN

39480 xBYTES ( obj 🡺 chksum size )

067: BYTES

38F21 xB>R ( # 🡺 R )

068: B🡪R

**Direcc. Nombre Descripción**

01E0DE xC2P ( {} 🡺 ????? )

069: C2P

07E314 ˜xCASCFG ( 🡺 )

070: CASCFG

0330DE xCASCMD ( 🡺 ? )

071: CASCMD

38B28 xCASE ( 🡺 )

072: CASE

3AD1B xCEIL ( x 🡺 n )

073: CEIL

3C3DC xCENTR ( (x,y) 🡺 )

( x 🡺 )

074: CENTR

3B4E9 xCF ( n 🡺 )

075: CF

03A314 ˜xCHINREM ( []1 []2 🡺 []3 )

076: CHINREM

00B0DE ( matrix 🡺 matrix' )

077: CHOLESKY

04D0AB xCHOOSE ( title {elems} pos 🡺 ob 1 )

( title {elems} pos 🡺 0 )

078: CHOOSE

3BC19 xCHR ( n 🡺 $ )

079: CHR

01D0DE xCIRC ( prg {} 🡺 ????? )

080: CIRC

3EDAC xCKSM ( n\_type 🡺 )

081: CKSM

3DD4E xCLEAR ( ob1 .. obn 🡺 )

082: CLEAR

39144 xCLKADJ ( ticks 🡺 )

083: CLKADJ

39839 xCLLCD ( 🡺 )

084: CLLCD

3EC95 xCLOSEIO ( 🡺 )

085: CLOSEIO

3E91A xCLUSR ( 🡺 )

086: CLVAR

3DD8E xCLSIGMA ( 🡺 )

087: CL∑

081314 ˜xCMPLX ( 🡺 )

088: CMPLX

3B193 xCNRM ( [] 🡺 col\_norm )

089: CNRM

03F0AB xCOL+ ( [[]] [[]]' n 🡺 [[]]'' )

( [] x n 🡺 []' )

090: COL+

03E0AB xCOL- ( [] n 🡺 []' xn )

( [[]] n 🡺 [[]]' [vn] )

091: COL-

3E5A0 xCOLCT ( symb 🡺 symb' )

092: COLCT

**Direcc. Nombre Descripción**

0300DE xCOLLECT ( symb 🡺 symb' )

093: COLLECT

3E0FD xSIGMACOL ( x\_col y\_col 🡺 )

094: COL∑

0390AB xCOL→ ( [v1]...[vn] n 🡺 [[]] )

( x1...xn n 🡺 [] )

095: COL🡪

3B423 xCOMB ( n k 🡺 Cn,k )

Symbolic argument allowed.

096: COMB

3BF77 xCON ( { n } x 🡺 [] )

( { n k } x 🡺 [[]] )

( [] x 🡺 []' )

097: CON

0260AB xCOND ( [[n\*n]] 🡺 x )

098: COND

3C967 xCONIC ( 🡺 )

099: CONIC

39A6C xCONJ ( x 🡺 x' )

100: CONJ

0180AB xCONLIB ( 🡺 )

101: CONLIB

0190AB xCONST ( name 🡺 x )

102: CONST

02A0DE ( 🡺 )

103: CONSTANTS

3989C xCONT ( 🡺 )

104: CONT

38F41 xCONVERT ( x1\_u1 x2\_u2 🡺 x3\_u2 )

105: CONVERT

3DE24 xCORR ( 🡺 x\_correlation )

106: CORR

3A5D0 xCOS ( x 🡺 x' )

107: COS

3A6C2 xCOSH ( x 🡺 x' )

108: COSH

3DE3F xCOV ( 🡺 x\_covariance )

109: COV

3D128 xCR ( 🡺 )

110: CR

393CA xCRDIR ( name 🡺 )

111: CRDIR

3B208 xCROSS ( [1] [2] 🡺 [3] )

112: CROSS

0410AB ( matrix i j 🡺 matrix' )

( vector i j 🡺 vector' )

113: CSWP

057314 ˜xCURL ( [func] [vars] 🡺 [] )

114: CURL

0150DE ( n 🡺 poly )

115: CYCLOTOMIC

0120AB xCYLIN ( 🡺 )

116: CYLIN

**Direcc. Nombre Descripción**

3C58E xC>PX ( (x,y) 🡺 {#n #m} )

117: C🡪PX

3BAF5 xC>R ( (x,y) 🡺 x y )

118: C🡪R

0610AB xDARCY ( xe/D yRe 🡺 xDarcy )

119: DARCY

39078 xDATE ( 🡺 date )

120: DATE

39238 xDATE+ ( date ndays 🡺 date' )

121: DATE+

0150DD xDBUG ( prog 🡺 )

( name 🡺 )

122: DBUG

39218 xDDAYS ( date1 date2 🡺 days )

123: DDAYS

3B670 xDEC ( 🡺 )

124: DEC

3E576 xDECR ( name 🡺 x\_new )

125: DECR

0370DE ( 🡺 $ )

126: DEDICACE

0250DE ( 'name=expr' 🡺 'name=expr' )

( 'name(name1...)=expr(name1...)' 🡺 'name(n1...)=expr(n1...)' )

127: DEF

3E85C xDEFINE ( 'name=expr' 🡺 )

( 'name(name1...)=expr(name1...)' 🡺 )

128: DEFINE

3B549 xDEG ( 🡺 )

129: DEG

0360DE ( poly 🡺 n )

130: DEGREE

391D8 xDELALARM ( n 🡺 )

131: DELALARM

3D1C7 xDELAY ( x\_delay 🡺 )

132: DELAY

3EF3B xDELKEYS ( rc.p 🡺 )

( 0 🡺 )

( 'S' 🡺 )

133: DELKEYS

3C51F xDEPND ( name 🡺 )

( {name y1 y2} 🡺 )

( {y1 y2} 🡺 )

( y1 y2 🡺 )

134: DEPND

3DCA7 xDEPTH ( 🡺 n )

135: DEPTH

00E314 ˜xDERIV ( symb var 🡺 symb' )

136: DERIV

003314 ˜xDERVX ( symb 🡺 symb' )

137: DERVX

00F314 ˜xDESOLVE ( eq func 🡺 func' )

138: DESOLVE

**Direcc. Nombre Descripción**

3B1BA xDET ( [[]] 🡺 x )

139: DET

3EB84 xDETACH ( n 🡺 )

( :port:n 🡺 )

140: DETACH

00C0DE ( matrix expr 🡺 matrix' )

141: DIAGMAP

03B0AB xDIAG→ ( [] { dims } 🡺 [[]] )

142: DIAG🡪

084314 ˜xDIFF ( 🡺 )

143: DIFF

00E0AB xDIFFEQ ( 🡺 )

144: DIFFEQ

39725 xDISP ( obj n\_line 🡺 )

146: DISP

0160DD xDISPXY ( ob {#x #y} %size 🡺 )

Display o (decompiled if nexessary) at the given display coordinates, using either the system font (%size=2) or the minifont (%size=1).

147: DISPXY

0190DE ( ob 🡺 ob' )

148: DISTRIB

056314 ˜xDIV ( [func] [vars] 🡺 func )

149: DIV

026314 ˜xDIV2 ( symb1 symb2 🡺 squot srem )

150: DIV2

072314 ˜xDIV2MOD ( symb1 symb2 🡺 squot srem )

151: DIV2MOD

044314 ˜xDIVIS ( symb 🡺 {} )

152: DIVIS

071314 ˜xDIVMOD ( symb1 symb2 🡺 sq )

153: DIVMOD

062314 ˜xDIVPC ( symb1 symb2 n 🡺 symb3 )

154: DIVPC

3816B xDO ( 🡺 )

155: DO

39527 xDOERR ( n 🡺 )

( $ 🡺 )

( 0 🡺 )

156: DOERR

05B0AB xDOLIST ( {1}...{n} n prog 🡺 {} )

( {1}...{n} prog 🡺 {} (n=1) )

157: DOLIST

0210DE ( func 🡺 {} )

158: DOMAIN

0540AB xDOSUBS ( {} n prog 🡺 {}' )

( {} prog 🡺 {}' (n=1) )

159: DOSUBS

3B1E1 xDOT ( [1] [2] 🡺 x )

160: DOT

3C484 xDRAW ( 🡺 )

161: DRAW

**Direcc. Nombre Descripción**

06B0AB xDRAW3DMATRIX ( [[]] v\_min v\_max 🡺 )

162: DRAW3DMATRIX

3C4BA xDRAX ( 🡺 )

163: DRAX

0230DE ( (x1 y1) (x2 y2) 🡺 ecuacrecta )

164: DROITE

3DC3B xDROP ( ob 🡺 )

165: DROP

3DC56 xDROP2 ( ob1 ob2 🡺 )

166: DROP2

3DCC7 xDROPN ( ob1...obn n 🡺 )

167: DROPN

3EFEF xDTAG ( tag:obj 🡺 obj )

168: DTAG

3DBEA xDUP ( ob 🡺 ob ob )

169: DUP

3DC05 xDUP2 ( 1 2 🡺 1 2 1 2 )

170: DUP2

3F29A xDUPDUP ( 1 🡺 1 1 )

171: DUPDUP

3DCE2 xDUPN ( 1...n n 🡺 1...n 1...n )

172: DUPN

3B06E xD>R ( x 🡺 (π/180)x )

173: D🡪R

0070DD xEDIT ( ob 🡺 ob' )

174: EDIT

0090DD xEDITB ( ob 🡺 ob' )

175: EDITB

02E314 ˜xEGCD ( symb1 symb2 🡺 symb3 symb4 symb5 )

176: EGCD

02C0AB xEGV ( [[]] 🡺 [[evect]]' [evals] )

177: EGV

02D0AB xEGVL ( [[]] 🡺 [egval] )

178: EGVL

3805D xELSE ( 🡺 )

179: ELSE

38A54 xENDDO ( 1/0 🡺 )

180: END

0570AB xENDSUB ( 🡺 x )

Number of lists in DOSUBS.

181: ENDSUB

3B5DA xENG ( n 🡺 )

182: ENG

088314 ˜xEPSX0 ( symb1 🡺 symb2 )

183: EPSX0

0000E3 ( 🡺 )

184: EQNLIB

Forma parte de la biblioteca 227 EQLIB: Equation Lib

00B0DD xEQW ( symb 🡺 symb' )

185: EQW

3BDE6 xEQ> ( 'l=r' 🡺 l r )

186: EQ🡪

**Direcc. Nombre Descripción**

3C553 xERASE ( 🡺 )

187: ERASE

3955B xERR0 ( 🡺 )

188: ERR0

39591 xERRM ( 🡺 $msg )

189: ERRM

39576 xERRN ( 🡺 $nerr )

190: ERRN

038314 ˜xEULER ( z1 🡺 z2 )

191: EULER

395AC xEVAL ( ob 🡺 ? )

192: EVAL

06C314 ˜xEXLR ( symb 🡺 symb1 symb2 )

193: EXLR

3A9B7 xEXP ( x 🡺 x' )

194: EXP

087314 ˜xEXP&LN ( 🡺 )

195: EXP&LN

03E0DE ( ob 🡺 ob' )

196: EXP2HYP

01A0DE ( ob 🡺 ob' )

197: EXP2POW

3E5E9 xEXPAN ( symb1 🡺 symb2 )

198: EXPAN

000314 ˜xEXPAND ( symb1 🡺 symb2 )

( [symb1] 🡺 [symb2] )

199: EXPAND

076314 ˜xEXPANDMOD ( symb1 🡺 symb2 )

200: EXPANDMOD

3E25E xEXPFIT ( 🡺 )

201: EXPFIT

017314 ˜xEXPLN ( symb1 🡺 symb2 )

202: EXPLN

3AB6F xEXPM ( x 🡺 x' )

203: EXPM

0050AB xEYEPT ( xx xy xz 🡺 )

204: EYEPT

0620AB xF0λ ( y\_lambda xT 🡺 x\_power )

205: F0λ

3ABAF xFACT ( x 🡺 x' )

206: FACT

001314 ˜xFACTOR ( symb 🡺 symb1\*symb2... )

( z 🡺 z1\*z2... )

207: FACTOR

077314 ˜xFACTORMOD ( symb 🡺 symb1\*symb2... )

208: FACTORMOD

043314 ˜xFACTORS ( z 🡺 {z1 m1...} )

( symb 🡺 {symb1 m1...} )

209: FACTORS

0600AB xFANNING ( x\_x/D y\_Re 🡺 x\_fanning )

210: FANNING

3F2DF xFAST3D ( 🡺 )

211: FAST3D

**Direcc. Nombre Descripción**

3B529 xFC? ( n 🡺 0/1 )

212: FC?

3B635 xFC?C ( n 🡺 0/1 )

213: FC?C

041314 ˜xFCOEF ( [] 🡺 symb )

214: FCOEF

0180DE ( ob 🡺 ob' )

215: FDISTRIB

01A0AB xFFT ( [] 🡺 []' )

216: FFT

00C0DD xFILER ( 🡺 )

217: FILER

391AE xFINDALARM ( date 🡺 n )

( {date time} 🡺 n )

( 0 🡺 n )

218: FINDALARM

3ED76 xFINISH ( 🡺 )

219: FINISH

3B59A xFIX ( n 🡺 )

220: FIX

0170AB xFLASHEVAL ( # 🡺 ? )

221: FLASHEVAL

3ACD1 xFLOOR ( x 🡺 n )

222: FLOOR

00F0DD xFONT6 ( 🡺 font )

223: FONT6

00E0DD xFONT7 ( 🡺 font )

224: FONT7

00D0DD xFONT8 ( 🡺 font )

225: FONT8

0030DD xFONT→ ( 🡺 font )

226: FONT🡪

38252 xSTARTVAR ( start finish 🡺 )

227: FOR

05E314 ˜xFOURIER ( symb z 🡺 c\_z )

228: FOURIER

3AC87 xFP ( x 🡺 x' )

229: FP

3EB2C xFREE ( 🡺 )

230: FREE

39745 xFREEZE ( n 🡺 )

231: FREEZE

042314 ˜xFROOTS ( symb 🡺 [] )

232: FROOTS

3B615 xFS?C ( n 🡺 0/1 )

233: FS?

3B509 xFS? ( n 🡺 0/1 )

234: FS?C

3C955 xFUNCTION ( 🡺 )

235: FUNCTION

06B314 ˜xFXND ( 'x/y' 🡺 x y )

236: FXND

**Direcc. Nombre Descripción**

0070DE xGAMMA ( x 🡺 x' )

237: GAMMA

04D314 ˜xGAUSS ( symb [vars] 🡺 [diag] [P] symb' [vars] )

238: GAUSS

03A0DE ( [poly] [vars] 🡺 [] )

239: GBASIS

02C314 ˜xGCD ( x1 x2 🡺 x3 )

240: GCD

075314 ˜xGCDMOD ( x1 x2 🡺 x3 )

241: GCDMOD

3C1C7 xGET ( ob n 🡺 elm )

ob = [] or [[]] or {} or name

pos = n or {n} or {n m}

242: GET

3C22D xGETI ( ob pos 🡺 ob' pos' elm )

ob = [] or [[]] or {} or name

pos = n or {n} or {n m}

243: GETI

3C74A xGOR ( g\_targ {#n #m} grob 🡺 g\_targ' )

( g\_targ (x,y) grob 🡺 g\_targ' )

( PICT ... ... 🡺 )

244: GOR

3B57F xGRAD ( 🡺 )

245: GRAD

0090DE ( vector func 🡺 vector' )

246: GRAMSCHMIDT

03B0DE ( [] [] [vars] 🡺 [] )

247: GREDUCE

00A0AB xGRIDMAP ( 🡺 )

248: GRIDMAP

07C314 ˜xGROBADD ( gr1 gr2 🡺 gr3 )

250: GROBADD

3C7D8 xGXOR ( g\_targ {#n #m} g\_src 🡺 g\_targ' )

( g\_targ (x,y) g\_src 🡺 g\_targ' )

( PICT ... ... 🡺 )

251: GXOR

046314 ˜xHADAMARD ( [M1] [M2] 🡺 [M3] )

252: HADAMARD

020314 ˜xHALFTAN ( symb 🡺 symb' )

253: HALFTAN

3880D xHALT ( 🡺 )

254: HALT

0510AB ( list 🡺 ob )

( $ 🡺 $' )

255: HEAD

0050DD xHEADER→ ( 🡺 n )

256: HEADER🡪

0320DE ( 🡺 ? )

257: HELP

05C314 ˜xHERMITE ( z 🡺 symb )

258: HERMITE

059314 ˜xHESS ( symb [vars] 🡺 [M] [grad] [vars] )

259: HESS

**Direcc. Nombre Descripción**

3B68B xHEX ( 🡺 )

260: HEX

054314 ˜xHILBERT ( z 🡺 [M] )

261: HILBERT

3C9C1 xHISTOGRAM ( 🡺 )

262: HISTOGRAM

3E1CA xHISTPLOT ( 🡺 )

263: HISTPLOT

3B12C xHMS+ ( hms1 hms2 🡺 hms3 )

264: HMS+

3B14C xHMS- ( hms1 hms2 🡺 hms3 )

265: HMS-

3B10C xHMS> ( x 🡺 x' )

266: HMS🡪

39405 xHOME ( 🡺 )

267: HOME

037314 ˜xHORNER ( symb1 x 🡺 symb2 x symb3 )

268: HORNER

02B0DE ( 🡺 )

269: HYPERBOLIC

031314 ˜xIABCUV ( n1 n2 n3 🡺 n4 n5 )

270: IABCUV

0120DE ( {[]} {[]} 🡺 {[]} )

271: IBASIS

0060DE xIBERNOULLI ( n 🡺 x )

272: IBERNOULLI

00B314 ˜xIBP ( uv' v 🡺 uv -u'v )

273: IBP

03B314 ˜xICHINREM ( []1 []2 🡺 []3 )

274: ICHINREM

027314 ˜xIDIV2 ( n1 n2 🡺 quot rem )

275: IDIV2

3C02E xIDN ( n 🡺 [[]] )

( [[]] 🡺 [[]]' )

( name 🡺 [[]] )

276: IDN

02F314 ˜xIEGCD ( n1 n2 🡺 c b a )

277: IEGCD

37F48 xIF ( 🡺 )

278: IF

387AC xIFERR ( 🡺 )

279: IFERR

01B0AB xIFFT ( [] 🡺 []' )

280: IFFT

396A4 xIFT ( 0/1 obj 🡺 ? )

281: IFT

395F3 xIFTE ( 0/1 objT objF 🡺 ? )

282: IFTE

011314 ˜xILAP ( symb 🡺 symb' )

283: ILAP

3B87E xIM ( (x,y) 🡺 y )

( [] 🡺 []' )

284: IM

**Direcc. Nombre Descripción**

0100DE ( matrix 🡺 {[]} )

285: IMAGE

3E54C xINCR ( name 🡺 x' )

286: INCR

3C33E xINDEP ( name 🡺 )

( {name x1 x2} 🡺 )

( {x1 x2} 🡺 )

( x1 x2 🡺 )

287: INDEP

04C0AB xINFORM ( $ {flds} fmt {rst} {init} 🡺 {} 1 )

( $ {flds} fmt {rst} {init} 🡺 0 )

288: INFORM

3EEBD xINPUT ( $prompt $ 🡺 $' )

( $prompt {specs} 🡺 $' )

289: INPUT

3F007 xINT ( f(var) var x0 🡺 F(x0) )

290: INT

0290DE ( 🡺 )

291: INTEGER

004314 ˜xINTVX ( f(x) 🡺 F(x) )

292: INTVX

3A32B xINV ( x 🡺 1/x )

( [[]] 🡺 [[]]' )

293: INV

074314 ˜xINVMOD ( x 🡺 x' )

294: INVMOD

3AC3D xIP ( x 🡺 n )

295: IP

029314 ˜xIQUOT ( n1 n2 🡺 n3 )

296: IQUOT

02B314 ˜xIREMAINDER ( n1 n2 🡺 n3 )

297: IREMAINDER

3E648 xISOL ( symb var 🡺 symb' )

298: ISOL

00D0DE ( matrix 🡺 {} )

299: ISOM

03C314 ˜xISPRIME? ( n 🡺 1 )

( n 🡺 0 )

300: ISPRIME

3F0B7 xI>R ( n 🡺 x )

301: I🡪R

050314 ˜xJORDAN ( [nxn] 🡺 minpol chrpol {} [] )

302: JORDAN

00F0DE ( matrix 🡺 {} )

303: KER

3EE2C xKERRM ( 🡺 msg )

304: KERRM

39854 xKEY ( 🡺 rc 1 )

( 🡺 0 )

305: KEY

07B314 ˜xKEYEVAL ( rc.p 🡺 ? )

306: KEYEVAL

**Direcc. Nombre Descripción**

06D0AB xKEYTIME→ ( 🡺 ticks )

307: KEYTIME🡪

3ECE4 xKGET ( name 🡺 )

( "name" 🡺 )

( {names} 🡺 )

( {{old new}...} 🡺 )

308: KGET

394F1 xKILL ( 🡺 )

309: KILL

3C5C9 xLABEL ( 🡺 )

310: LABEL

05D314 ˜xLAGRANGE ( [2xn] 🡺 pol )

311: LAGRANGE

0010DD xLANGUAGE→ ( 🡺 n )

312: LANGUAGE🡪

010314 ˜xLAP ( symb 🡺 symb' )

313: LAP

058314 ˜xLAPL ( symb [vars] 🡺 symb' )

314: LAPL

397E5 xLAST ( 🡺 ob1 .. obn )

315: LASTARG

3C866 xLCD> ( 🡺 grob )

316: LCD🡪

02D314 ˜xLCM ( symb1 symb2 🡺 symb3 )

317: LCM

055314 ˜xLCXM ( n1 n2 prog 🡺 [] )

318: LCXM

012314 ˜xLDEC ( symb1 symb2 🡺 symb3 )

319: LDEC

05A314 ˜xLEGENDRE ( n 🡺 pol )

320: LEGENDRE

032314 ˜xLGCD ( {symb...} 🡺 {} gcd )

321: LGCD

0160AB xLIBEVAL ( # 🡺 ? )

322: LIBEVAL

3EB42 xLIBS ( 🡺 {title nlib nport ...} )

323: LIBS

014314 ˜xLIN ( symb 🡺 symb' )

324: LIN

3C68C xLINE ( (x1,y1) (x2,y2) 🡺 )

( {#n1 #m1} {#n2 #m2} 🡺 )

325: LINE

3E214 xLINFIT ( 🡺 )

326: LINFIT

0150AB xLININ ( symb var 🡺 0/1 )

327: LININ

052314 ˜xLINSOLVE ( [eqs] [vars] 🡺 [eqs] {pp} sol )

328: LINSOLVE

3BAC1 xLIST> ( {} 🡺 ob1...obn n )

329: LIST🡪

3AA01 xLN ( x 🡺 x' )

330: LN

**Direcc. Nombre Descripción**

06D314 ˜xLNAME ( symb 🡺 [vars] )

331: LNAME

016314 ˜xLNCOLLECT ( symb 🡺 symb' )

332: LNCOLLECT

3AB2F xLNP1 ( x 🡺 x' )

333: LNP1

03C0DE ( {} 🡺 )

334: LOCAL

3AA73 xLOG ( x 🡺 x' )

335: LOG

3E239 xLOGFIT ( 🡺 )

336: LOGFIT

0320AB xLQ ( [[]] 🡺 [[L]] [[Q]] [[P]] )

337: LQ

3DF83 xLR ( 🡺 Intercept Slope )

338: LR

02B0AB xLSQ ( [B] [[A]] 🡺 []' )

( [[B]] [[A]] 🡺 [[]]' )

339: LSQ

0300AB xLU ( [[]] 🡺 [[L]] [[U]] [[P]] )

340: LU

06A314 ˜xLVAR ( symb 🡺 symb [vars] )

341: LVAR

051314 ˜xMAD ( [] 🡺 det inv coeff cpol )

342: MAD

07F314 ˜xMAIN ( 🡺 )

343: MAIN

3B02E xMANT ( x 🡺 x' )

344: MANT

066314 ˜xMAP ( {} prog 🡺 {}' )

345: MAP

02F0DE ( 🡺 )

346: MATHS

083314 ˜xMATR ( 🡺 )

347: MATR

3ADA5 xMAX ( x y 🡺 x' )

348: MAX

39AE4 xMAXR ( 🡺 MAXR )

349: MAXR

3DEE1 xMAXSIGMA ( 🡺 xmax )

( 🡺 [x1...xn] )

350: MAX∑

0760AB xMCALC ( var 🡺 )

( {vars} 🡺 )

( "ALL" 🡺 )

351: MCALC

3DEFC xMEAN ( 🡺 xmean )

( 🡺 [x1...xn] )

352: MEAN

3E8C1 xMEM ( 🡺 x )

353: MEM

3E9D4 xMENU ( % 🡺 )

354: MENU

**Direcc. Nombre Descripción**

07A314 ˜xMENUXY ( n1 n2 🡺 )

355: MENUXY

3EB16 xMERGE ( 🡺 )

356: MERGE

3AE2B xMIN ( x y 🡺 x' )

357: MIN

0030E3 ( 🡺 )

358: MINEHUNT

Forma parte de la biblioteca 227 EQLIB: Equation Lib

0120DD xMINIFONT→ ( 🡺 font )

359: MINIFONT🡪

0730AB xMINIT ( 🡺 )

360: MINIT

39B01 xMINR ( 🡺 MINR )

361: MINR

3DF17 xMINSIGMA ( 🡺 xmin )

( 🡺 [x1...xn] )

362: MIN∑

0740AB xMITM ( title {vars} 🡺 )

363: MITM

00E0DE ( {} 1/-1 🡺 matrix )

364: MKISOM

3AFCB xMOD ( x y 🡺 x' )

365: MOD

079314 ˜xMODSTO ( mod 🡺 )

366: MODSTO

02C0DE ( 🡺 )

367: MODULAR

0020E5 ( id/$ 🡺 PM )

368: MOLWT

Forma parte de la biblioteca 229 PRTBL: Periodic Table

0770AB xMROOT ( var 🡺 x )

( "ALL" 🡺 )

369: MROOT

04E0AB xMSGBOX ( $ 🡺 )

370: MSGBOX

0200DE ( [eqs] [vars] [inic] 🡺 [eqs] [vars] [sol] )

371: MSLV

0720AB xMSOLVR ( 🡺 )

372: MSOLVR

0020E3 ( 🡺 )

373: MSOLVR2

Forma parte de la biblioteca 227 EQLIB: Equation Lib

070314 ˜xMULTMOD ( symb1 symb2 🡺 symb3 )

374: MULTMOD

0750AB xMUSER ( var 🡺 )

( {vars} 🡺 )

( "ALL" 🡺 )

375: MUSER

01C0AB xNDIST ( xq v x 🡺 x' )

376: NDIST

3F2B5 xNDUPN ( ob n 🡺 ob .. ob n )

377: NDUPN

**Direcc. Nombre Descripción**

39976 xNEG ( x 🡺 x' )

378: NEG

394AA xNEWOB ( ob 🡺 ob )

379: NEWOB

3831C xNEXT ( 🡺 )

380: NEXT

03D314 ˜xNEXTPRIME ( n 🡺 n' )

381: NEXTPRIME

3F264 xNIP ( ob1 ob2 🡺 ob2 )

382: NIP

3CB13 xNOT ( x 🡺 x' )

383: NOT

3F0FC xNOVAL ( 🡺 )

384: NOVAL

0560AB xNSUB ( 🡺 npos )

385: NSUB

3BBF9 xNUM ( $ 🡺 n )

386: NUM

0060AB xNUMX ( n 🡺 )

387: NUMX

0070AB xNUMY ( n 🡺 )

388: NUMY

3DE09 xNSIGMA ( 🡺 nrows )

389: N∑

3BE38 xOBJ> ( ob 🡺 ? )

390: OBJ🡪

3B6A6 xOCT ( 🡺 )

391: OCT

3950C xOFF ( 🡺 )

392: OFF

3D0BC xOLDPRT ( 🡺 )

393: OLDPRT

3EC75 xOPENIO ( 🡺 )

394: OPENIO

3CA8D xOR ( x y 🡺 x' )

395: OR

3E8F0 xORDER ( {names} 🡺 )

396: ORDER

3DC8C xOVER ( 1 2 🡺 1 2 1 )

397: OVER

01F0DE ( {} 🡺 {}' )

398: P2C

039314 ˜xPA2B2 ( n 🡺 n' )

399: PA2B2

3C98B xPARAMETRIC ( 🡺 )

400: PARAMETRIC

3EDEC xPARITY ( n 🡺 )

401: PARITY

0090AB xPARSURFACE ( 🡺 )

402: PARSURFACE

034314 ˜xPARTFRAC ( symb 🡺 symb' )

403: PARTFRAC

**Direcc. Nombre Descripción**

393EA xPATH ( 🡺 {HOME dir1 .. dirn} )

404: PATH

04F314 ˜xPCAR ( [nxn] 🡺 pol )

405: PCAR

0450AB xPCOEF ( [roots] 🡺 [coefs] )

406: PCOEF

00D0AB xPCONTOUR ( 🡺 )

407: PCONTOUR

01F0AB xPCOV ( 🡺 xpcovariance )

408: PCOV

3C4F5 xPDIM ( (xmin,ymin) (xmax,ymax) 🡺 )

( #width #height 🡺 )

409: PDIM

0030E5 ( 🡺 )

410: PERINFO

Forma parte de la biblioteca 229 PRTBL: Periodic Table

3B477 xPERM ( n k 🡺 n' )

411: PERM

0000E5 ( 🡺 )

412: PERTBL

Forma parte de la biblioteca 229 PRTBL: Periodic Table

0460AB xPEVAL ( [coefs] x 🡺 x' )

413: PEVAL

3EAA7 xPGDIR ( name 🡺 )

414: PGDIR

3DCFD xPICK ( 1...n n 🡺 1..n 1 )

415: PICK

3F27F xPICK3 ( 1 2 3 🡺 1 2 3 1 )

416: PICK3

3C72A xPICT ( 🡺 PICT )

417: PICT

3C5AE xGRAPH ( 🡺 )

418: PICTURE

06A0AB xPINIT ( 🡺 )

419: PINIT

3C662 xPIX? ( (x,y) 🡺 1/0 )

( {#n #m} 🡺 1/0 )

420: PIX?

3C638 xPIXOFF ( (x,y) 🡺 )

( {#n #m} 🡺 )

421: PIXOFF

3C60E xPIXON ( (x,y) 🡺 )

( {#n #m} 🡺 )

422: PIXON

3EE9D xPKT ( data type 🡺 response )

423: PKT

009314 ˜xPLOT ( expr 🡺 expr )

424: PLOT

00A314 ˜xPLOTADD ( f 🡺 )

425: PLOTADD

3C392 xPMAX ( (x,y) 🡺 )

426: PMAX

**Direcc. Nombre Descripción**

3C372 xPMIN ( (x,y) 🡺 )

427: PMIN

0140DE ( matrix 🡺 matrix' )

428: PMINI

3C979 xPOLAR ( 🡺 )

429: POLAR

02D0DE ( 🡺 )

430: POLYNOMIAL

0350DE ( 🡺 )

431: POP

3BB94 xPOS ( str substring 🡺 n/0 )

( {} ob 🡺 n/0 )

432: POS

0380DE ( [expr] [vars] 🡺 funcpotencial )

433: POTENTIAL

01B0DE ( expr 🡺 expr' )

434: POWEXPAND

073314 ˜xPOWMOD ( symb exp 🡺 symb' )

435: POWMOD

3D0D7 xPR1 ( ob 🡺 ob )

436: PR1

3DFDD xPREDV ( x 🡺 y )

437: PREDV

3E01D xPREDX ( y 🡺 x )

438: PREDX

3DFFD xPREDY ( x 🡺 y )

439: PREDY

00C314 ˜xPREVAL ( f x1 x2 🡺 symb )

( f x1 x2 🡺 x )

440: PREVAL

03E314 ˜xPREVPRIME ( n 🡺 n' )

441: PREVPRIME

3D1E7 xPRLCD ( 🡺 )

442: PRLCD

38BBF xPROMPT ( $ 🡺 )

443: PROMPT

08B314 ˜xPROMPTSTO ( var 🡺 )

444: PROMPTSTO

0440AB xPROOT ( [coefs] 🡺 [roots] )

445: PROOT

035314 ˜xPROPFRAC ( x 🡺 symb' )

446: PROPFRAC

3D10D xPRST ( 🡺 )

447: PRST

3D0F2 xPRSTC ( 🡺 )

448: PRSTC

3D143 xPRVAR ( name 🡺 )

( {names} 🡺 )

( :port:name 🡺 )

449: PRVAR

01D0AB xPSDEV ( 🡺 xpsdev )

( 🡺 {x1...xn} )

450: PSDEV

**Direcc. Nombre Descripción**

0040DE xPSI ( symb 🡺 symb' )

451: PSI

036314 ˜xPTAYL ( pol x 🡺 pol' )

452: PTAYL

0010E5 ( symb/y x 🡺 $/unit/id/% )

453: PTPROP

Forma parte de la biblioteca 229 PRTBL: Periodic Table

3E87C xPURGE ( name 🡺 )

{names} 🡺 )

:port:name 🡺 )

:port:nlib 🡺 )

454: PURGE

0340DE ( 🡺 )

455: PUSH

3C0BF xPUT ( ob pos obj 🡺 ob' )

ob = [] or [[]] or {} or name

pos = n or {n} or {n m}

456: PUT

3C139 xPUTI ( ob pos obj 🡺 [] pos' )

ob = [] or [[]] or {} or name

pos = n or {n} or {n m}

457: PUTI

01E0AB ( 🡺 varianza )

458: PVAR

3EA49 xPVARS ( nport 🡺 {} mem )

459: PVARS

3C5E4 xPVIEW ( (x,y) 🡺 )

( {#n #m} 🡺 )

460: PVIEW

3E283 xPWRFIT ( 🡺 )

461: PWRFIT

3C56E xPX>C ( {#m #n} 🡺 (x,y) )

462: PX🡪C

0030DE xPsi ( symb n 🡺 symb' )

463: Psi

0310AB xQR ( [[]] 🡺 [[Q]] [[R]] [[P]] )

464: QR

3E66F xQUAD ( symb var 🡺 symb' )

465: QUAD

028314 ˜xQUOT ( p1 p2 🡺 p3 )

466: QUOT

3D6F6 xQUOTE ( ob 🡺 'ob )

467: QUOTE

04B314 ˜xQXA ( symb [vars] 🡺 [[]] [vars] )

468: QXA

3B564 xRAD ( 🡺 )

469: RAD

3B3E6 xRAND ( 🡺 x )

470: RAND

02A0AB xRANK ( [[]] 🡺 n )

471: RANK

0350AB xRANM ( {m n} 🡺 [[]] )

472: RANM

**Direcc. Nombre Descripción**

3DBCA xPREDIV ( x y 🡺 x/y )

473: RATIO

3D393 xRCEQ ( 🡺 EQ )

474: RCEQ

0420AB ( matrix x i 🡺 matrix' )

475: RCI

0430AB ( matrix x i j 🡺 matrix' )

476: RCIJ

3E6F1 xRCL ( var 🡺 x )

( :port:nlib 🡺 lib )

( :port:name 🡺 ob )

( :port:{path} 🡺 ob )

477: RCL

3918E xRCLALARM ( n 🡺 {date time action rep} )

478: RCLALARM

3B715 xRCLF ( 🡺 {#s1 #u1 #s2 #u2} )

479: RCLF

3EF79 xRCLKEYS ( 🡺 {ob ... key ...} )

480: RCLKEYS

3EA2E xRCLMENU ( 🡺 x )

481: RCLMENU

03F0DE xRCLVX ( 🡺 name )

Recall the current content of the reserved

CAS variable VX.

482: RCLVX

3DDA9 xRCLSIGMA ( 🡺 [[]] )

483: RCL∑

3B6FA xRCWS ( 🡺 n )

484: RCWS

3BEEC xRDM ( ob size 🡺 ob' )

( name size 🡺 )

ob= [] or [[]]

size = {n} or {n m}

485: RDM

3B401 xRDZ ( x 🡺 )

486: RDZ

3B819 xRE ( (x,y) 🡺 x )

( [] 🡺 []' )

487: RE

3ED22 xRECN ( name 🡺 )

( $name 🡺 )

488: RECN

0110AB xRECT ( 🡺 )

489: RECT

3ED56 xRECV ( 🡺 )

490: RECV

048314 ˜xREF ( [[]] 🡺 [[]]' )

491: REF

02A314 ˜xREMAINDER ( p1 p2 🡺 p3 )

492: REMAINDER

0130DD xRENAME ( name name' 🡺 )

493: RENAME

**Direcc. Nombre Descripción**

069314 ˜xREORDER ( pol var 🡺 pol' )

494: REORDER

38105 xREPEAT ( 1/0 🡺 )

495: REPEAT

3B9D2 xREPL ( ob pos new 🡺 ob' )

ob= [[]] or [] or {} or $ or PICT

pos= N or {n m} or (n,m)

496: REPL

3C41A xRES ( n\_int 🡺 )

497: RES

3EAE7 xRESTORE ( :port:name 🡺 )

498: RESTORE

0050DE xRESULTANT ( p1 p2 🡺 res )

499: RESULTANT

05D0AB xREVLIST ( {1...n} 🡺 {n...1}' )

500: REVLIST

0280DE ( 🡺 )

501: REWRITE

00D314 ˜xRISCH ( f var 🡺 F )

502: RISCH

0200AB xRKF ( {} xtol xTf 🡺 {} xtol )

( {} {xtol step} xTf 🡺 {} xtol )

503: RKF

0220AB xRKFERR ( {} h 🡺 {} h dy err )

504: RKFERR

0210AB xRKFSTEP ( {} tol h 🡺 {} tol h' )

505: RKFSTEP

38E01 xRL ( # 🡺 #' )

506: RL

38E21 xRLB ( # 🡺 #' )

507: RLB

3AEB1 xRND ( x n 🡺 x' )

508: RND

3B16C xRNRM ( [] 🡺 x )

509: RNRM

3DD18 xROLL ( 1...n n 🡺 2...n 1 )

510: ROLL

3DD33 xROLLD ( n ... 1 n 🡺 1 n...2 )

511: ROLLD

06F0AB xROMUPLOAD ( 🡺 )

512: ROMUPLOAD

3D3CE xROOT ( prog/s var guess 🡺 x )

( prog/s var {guesses} 🡺 x )

513: ROOT

3DC71 xROT ( 1 2 3 🡺 2 3 1 )

514: ROT

03D0AB xROW+ ( [[]] [[]]' n 🡺 [[]]'' )

( [[]] [] n 🡺 [[]]' )

( [] n n' 🡺 [] )

515: ROW+

03C0AB xROW- ( [[]] nrow 🡺 [[]]' [] )

( [] n 🡺 []' elt )

516: ROW-

**Direcc. Nombre Descripción**

0370AB xROW\-> ( [1]...[n] n 🡺 [] )

( x1...xn 🡺 [] )

517: ROW🡪

3F218 xRPL> ( 🡺 ob )

518: RPL>

38E41 xRR ( # 🡺 x' )

519: RR

38E61 xRRB ( # 🡺 x' )

520: RRB

0340AB xRREF ( [[]] 🡺 [[]]' )

521: RREF

078314 ˜xRREFMOD ( [[]] 🡺 [[]]' )

522: RREFMOD

0230AB xRRK ( {} xtol xTfinal 🡺 {} xtol )

523: RRK

0240AB xRRKSTEP ( {} xtol h last 🡺 {} xtol h' cur )

524: RRKSTEP

0250AB xRSBERR ( {} h 🡺 {} h dy err )

525: RSBERR

3B22F xRSD ( [B] [[A]] [Z] 🡺 []' )

526: RSD

0400AB xRSWP ( []/[[]] i j 🡺 []/[[]] )

527: RSWP

3E632 xRULES ( 🡺 )

528: RULES

38F01 xR>B ( x 🡺 # )

529: R🡪B

3B7ED xR>C ( x y 🡺 (x,y) )

530: R🡪C

3B0AE xR>D ( x 🡺 (180/π)x )

531: R🡪D

3F070 xR>I ( x 🡺 n )

532: R🡪I

3C9E5 xSAME ( ob1 ob2 🡺 1/0 )

533: SAME

3EE82 xSBRK ( 🡺 )

534: SBRK

3C4D5 xSCALE ( xs ys 🡺 )

535: SCALE

3C444 x\*H ( xf 🡺 )

536: SCALEH

3C464 x\*W ( yf 🡺 )

537: SCALEW

3E1EF xSCATRPLOT ( 🡺 )

538: SCATRPLOT

3C9AF xSCATTER ( 🡺 )

539: SCATTER

0330AB xSCHUR ( [[]] 🡺 [[Q]] [[T]] )

540: SCHUR

3B5BA xSCI ( n 🡺 )

541: SCI

3E127 xSCLSIGMA ( 🡺 )

542: SCL∑

**Direcc. Nombre Descripción**

3E385 xSCONJ ( name 🡺 )

543: SCONJ

07D314 ˜xSCROLL ( ob 🡺 )

544: SCROLL

3DF32 xSDEV ( 🡺 xsdev )

( 🡺 [x1...xn] )

545: SDEV

3ECB0 xSEND ( name 🡺 )

( {names} 🡺 )

( {{old new}...} 🡺 )

546: SEND

0530AB xSEQ ( prog var start end incr 🡺 {} )

547: SEQ

007314 ˜xSERIES ( func var order 🡺 {} symb' )

548: SERIES

3ED91 xSERVER ( 🡺 )

549: SERVER

064314 ˜xSEVAL ( symb 🡺 symb' )

550: SEVAL

3B4C9 xSF ( n 🡺 )

551: SF

3E696 xSHOW ( symb name 🡺 symb' )

( symb {names} 🡺 symb' )

552: SHOW

0630AB xSIDENS ( x 🡺 x' )

553: SIDENS

0020DE xSIGMA ( f var 🡺 F )

554: SIGMA

0010DE xSIGMAVX ( f(x) 🡺 F(x) )

555: SIGMAVX

3A3EE xSIGN ( x 🡺 x' )

556: SIGN

05F314 ˜xSIGNTAB ( symb 🡺 {} )

557: SIGNTAB

033314 ˜xSIMP2 ( x y 🡺 x/gcd y/gcd )

558: SIMP2

0220DE xSIMPLIFY ( symb 🡺 symb' )

559: SIMPLIFY

3A57C xSIN ( x 🡺 x' )

560: SIN

018314 ˜xSINCOS ( symb 🡺 symb' )

561: SINCOS

3A678 xSINH ( x 🡺 x' )

562: SINH

3E331 xSINV ( name 🡺 )

563: SINV

3BB1F xSIZE ( ob 🡺 n )

( ob 🡺 {N m} )

564: SIZE

38E81 xSL ( # 🡺 #' )

565: SL

38EA1 xSLB ( # 🡺 #' )

566: SLB

**Direcc. Nombre Descripción**

00C0AB xSLOPEFIELD ( 🡺 )

567: SLOPEFIELD

3E35B xSNEG ( name 🡺 )

568: SNEG

0290AB xSNRM ( [] 🡺 x )

569: SNRM

03F314 ˜xSOLVE ( symb var 🡺 {zeros} )

570: SOLVE

0010E3 ( n m 0/1 🡺 )

571: SOLVEQN

Forma parte de la biblioteca 227 EQLIB: Equation Lib

086314 ˜xSOLVER ( 🡺 )

572: SOLVER

008314 ˜xSOLVEVX ( symb 🡺 {zeros} )

573: SOLVEVX

05E0AB xSORT ( {} 🡺 {}' )

574: SORT

0130AB xSPHERE ( 🡺 )

575: SPHERE

3A4EF xSQ ( x 🡺 x' )

576: SQ

38EC1 xSR ( # 🡺 #' )

577: SR

0280AB xSRAD ( [[]] 🡺 x )

578: SRAD

38EE1 xSRB ( # 🡺 #' )

579: SRB

3EC55 xSRECV ( n 🡺 $ 0/1 )

580: SRECV

0100DD xSREPL ( str find repl 🡺 str' )

581: SREPL

381AB xSTART ( start finish 🡺 )

582: START

3B5FA xSTD ( 🡺 )

583: STD

3851F xSTEP ( n 🡺 )

( symb 🡺 )

584: STEP

3D3AE xSTEQ ( ob 🡺 )

585: STEQ

3EE62 xSTIME ( x 🡺 )

586: STIME

3E739 xSTO ( ob name 🡺 )

( ob :port:name 🡺 )

( lib port 🡺 )

( ob 'name(i)' 🡺 )

587: STO

3E4D2 xSTO\* ( ob name 🡺 )

588: STO\*

3E3AF xSTO+ ( ob name 🡺 )

589: STO+

3E406 xSTO- ( ob name 🡺 )

590: STO-

**Direcc. Nombre Descripción**

3E46C xSTO/ ( ob name 🡺 )

591: STO/

39164 xSTOALARM ( time 🡺 n )

( {date time act rep} 🡺 n )

592: STOALARM

3B749 xSTOF ( {#s1 #u1 #s2 #u2} 🡺 )

593: STOF

3EF07 xSTOKEYS ( {ob key ...} 🡺 )

( {'S' ob key ...} 🡺 )

( 'S' 🡺 )

594: STOKEYS

0240DE ( expr id 🡺 result )

595: STORE

0400DE xSTOVX ( name 🡺 )

Store object into the reserved CAS variable VX.

596: STOVX

3DD6E xSTOSIGMA ( ob 🡺 )

597: STO∑

0580AB xSTREAM ( {} prog 🡺 x )

598: STREAM

0000FF ( 🡺 )

599: STRM

3BBD9 xSTR> ( $ 🡺 ob )

600: STR🡪

0160AB ( poly 🡺 {} )

601: STURM

0170AB ( poly 🡺 {} )

602: STURMAB

3B6C1 xSTWS ( n 🡺 )

603: STWS

3B8D7 xSUB ( ob start end 🡺 ob' )

ob= [[]], $, {}, grob

start,end = n, {n m}, (n,m)

604: SUB

002314 ˜xSUBST ( symb var=s1 🡺 symb' )

605: SUBST

06F314 ˜xSUBTMOD ( x1 x2 🡺 x3 )

606: SUBTMOD

02E0AB xSVD ( [[]] 🡺 [[U]] [[V]] [S] )

607: SVD

02F0AB xSVL ( [[]] 🡺 [] )

608: SVL

3DC20 xSWAP ( ob1 ob2 🡺 ob2 ob1 )

609: SWAP

04E314 ˜xSYLVESTER ( [[]] 🡺 [D] [P] )

610: SYLVESTER

39705 xSYSEVAL ( # 🡺 ? )

611: SYSEVAL

00A0DE ( [eqs] [vars] 🡺 matrix )

612: SYST2MAT

061314 ˜xTABVAL ( symb(x) {vals} 🡺 symb(x) {{vals} {res}} )

613: TABVAL

**Direcc. Nombre Descripción**

060314 ˜xTABVAR ( symb(x) 🡺 symb(x) {{}{}} grob )

614: TABVAR

0520AB xTAIL ( {} 🡺 {}' )

( $ 🡺 $' )

615: TAIL

3A624 xTAN ( x 🡺 x' )

616: TAN

01C0DE xTAN2CS2 ( symb 🡺 symb' )

617: TAN2CS2

01F314 ˜xTAN2SC ( symb 🡺 symb' )

618: TAN2SC

021314 ˜xTAN2SC2 ( symb 🡺 symb' )

619: TAN2SC2

3A70C xTANH ( x 🡺 x' )

620: TANH

006314 ˜xTAYLOR0 ( symb 🡺 symb' )

621: TAYLOR0

3E6CA xTAYLR ( symb var n 🡺 symb' )

622: TAYLR

05B314 ˜xTCHEBYCHEFF ( n 🡺 pol )

623: TCHEBYCHEFF

01A314 ˜xTCOLLECT ( symb 🡺 symb' )

624: TCOLLECT

0640AB xTDELTA ( x y 🡺 x' )

625: TDELTA

02E0DE ( 🡺 )

626: TESTS

065314 ˜xTEVAL ( ob 🡺 ? time )

627: TEVAL

013314 ˜xTEXPAND ( symb 🡺 symb' )

628: TEXPAND

3C8FA xTEXT ( 🡺 )

629: TEXT

37F7F xTHEN ( 0/1 🡺 )

630: THEN

39093 xTICKS ( 🡺 # )

631: TICKS

3905D xTIME ( 🡺 time )

632: TIME

0650AB xTINC ( x y 🡺 x' )

633: TINC

019314 ˜xTLIN ( symb 🡺 symb' )

634: TLIN

3C6B6 xTLINE ( (x1,y1) (x2,y2) 🡺 )

( {#n1 #m1} {#n2 #m2} 🡺 )

635: TLINE

3E97B xTMENU ( % 🡺 [InitMenu%] )

( Ob 🡺 [@LIST InitMenu] )

636: TMENU

3DF4D xTOT ( 🡺 xsum )

( 🡺 {x1...xn} )

637: TOT

**Direcc. Nombre Descripción**

0270AB xTRACE ( [[]] 🡺 x )

638: TRACE

045314 ˜xTRAN ( [[]] 🡺 [[]]' )

( name 🡺 )

639: TRAN

3EE0C xTRANSIO ( n 🡺 )

640: TRANSIO

01B314 ˜xTRIG ( symb 🡺 symb' )

641: TRIG

01C314 ˜xTRIGCOS ( symb 🡺 symb' )

642: TRIGCOS

082314 ˜xTRIGO ( 🡺 )

643: TRIGO

01D314 ˜xTRIGSIN ( symb 🡺 symb' )

644: TRIGSIN

01E314 ˜xTRIGTAN ( symb 🡺 symb' )

645: TRIGTAN

3C084 xTRN ( [[]] 🡺 [[]]' )

( name 🡺 )

646: TRN

3AF3E xTRNC ( x n 🡺 )

647: TRNC

063314 ˜xTRUNC ( symb1 symb2 🡺 symb3 )

648: TRUNC

3C99D xTRUTH ( 🡺 )

649: TRUTH

015314 ˜xTSIMP ( symb 🡺 symb' )

650: TSIMP

391F8 xTSTR ( date time 🡺 $ )

651: TSTR

39456 xTVARS ( ntype 🡺 {} )

( {n...} 🡺 {} )

652: TVARS

0470AB xTVM ( 🡺 )

653: TVM

0480AB xTVMBEG ( 🡺 )

654: TVMBEG

0490AB xTVMEND ( 🡺 )

655: TVMEND

04A0AB xTVMROOT ( var 🡺 x )

656: TVMROOT

3BC39 xTYPE ( ob 🡺 %type )

657: TYPE

38FD7 xUBASE ( u 🡺 u' )

658: UBASE

3900B xUFACT ( u1 u2 🡺 u3 )

659: UFACT

0140DD xUFL1\->MINIF ( ob n 🡺 font )

660: UFL1🡪MINIF

0310DE ( var/{var} 🡺 ob/{} )

661: UNASSIGN

0270DE ( var/{var} 🡺 ob/{} )

662: UNASSUME

**Direcc. Nombre Descripción**

03D0DE ( 🡺 {} )

663: UNBIND

3F249 xUNPICK ( obn...ob1 ob n 🡺 ob...ob2 )

664: UNPICK

3F22E xUNROT ( 1 2 3 🡺 3 1 2 )

665: UNROT

38195 xUNTIL ( 🡺 )

666: UNTIL

39420 xUPDIR ( 🡺 )

667: UPDIR

3E07D xUTPC ( n x 🡺 x' )

668: UTPC

3E0BD xUTPF ( n1 n2 x 🡺 x' )

669: UTPF

3E09D xUTPN ( n v x 🡺 x' )

670: UTPN

3E0DD xUTPT ( n x 🡺 x' )

671: UTPT

38F81 xUVAL ( u 🡺 x )

672: UVAL

053314 ˜xVANDERMONDE ( {} 🡺 [[]] )

673: VANDERMONDE

3DF68 xVAR ( 🡺 x )

( 🡺 [x1...xn] )

674: VAR

3943B xVARS ( 🡺 {} )

675: VARS

08C314 ˜xVER ( 🡺 $ )

676: VER

00F0AB xVERSION ( 🡺 $ $ )

677: VERSION

0080DD xVISIT ( name 🡺 )

678: VISIT

00A0DD xVISITB ( name 🡺 )

679: VISITB

0390DE ( [curlU] [vars] 🡺 [U] )

680: VPOTENTIAL

3BDB2 xVTYPE ( name 🡺 n )

681: VTYPE

3C2AC xV> ( []/() 🡺 x y )

( []/() 🡺 x y z )

(in current co-system)

682: V🡪

39819 xWAIT ( sec 🡺 )

( 0 🡺 rc.p )

683: WAIT

380DB xWHILE ( 🡺 )

684: WHILE

0080AB xWIREFRAME ( 🡺 )

685: WIREFRAME

390AE xWSLOG ( 🡺 $ $ $ $ )

686: WSLOG

**Direcc. Nombre Descripción**

3E03D xXCOL ( n 🡺 )

687: XCOL

0700AB xXGET ( name 🡺 )

688: XGET

3EC35 xXMIT ( $ 🡺 1 )

( $ 🡺 $rest 0 )

689: XMIT

067314 ˜xXNUM ( x 🡺 x' )

690: XNUM

3CB7A xXOR ( # #' 🡺 #'' )

( $ $' 🡺 $'' )

( 1/0 1/0 🡺 1/0 )

691: XOR

3AD65 xXPON ( % 🡺 )

( symb 🡺 )

692: XPON

0710AB xXPUT ( name 🡺 )

693: XPUT

068314 ˜xXQ ( x 🡺 x' )

694: XQ

0500AB xXRECV ( name 🡺 )

695: XRECV

3C915 xXRNG ( x1 x2 🡺 )

696: XRNG

3A278 xXROOT ( y x 🡺 Y' )

697: XROOT

04F0AB xXSEND ( name 🡺 )

698: XSEND

06E0AB xXSERV ( 🡺 )

699: XSERV

0000AB xXVOL ( x1 x2 🡺 )

700: XVOL

0030AB xXXRNG ( x1 x2 🡺 )

701: XXRNG

3E05D xYCOL ( n 🡺 )

702: YCOL

3C935 xYRNG ( y1 y2 🡺 )

703: YRNG

00B0AB xYSLICE ( 🡺 )

704: YSLICE

0010AB xYVOL ( y1 y2 🡺 )

705: YVOL

0040AB xYYRNG ( y1 y2 🡺 )

706: YYRNG

040314 ˜xZEROS ( symb var 🡺 {zeros} )

707: ZEROS

05F0AB xZFACTOR ( xTr yPr 🡺 xZf )

708: ZFACTOR

0020AB xZVOL ( x1 x2 🡺 )

709: ZVOL

3A097 xˆ ( y x 🡺 yˆx )

710: ^

**Direcc. Nombre Descripción**

3DB62 xFORMUNIT ( x y 🡺 x\_y )

711: \_

0690AB xdB ( 🡺 %1 )

712: dB

39B1E xCONSTANTe ( 🡺 e )

713: e

0660AB xgmol ( 🡺 u )

714: gmol

39B3B xi ( 🡺 i )

715: i

0670AB xlbmol ( 🡺 u )

716: lbmol

005314 ˜xLIMIT ( func point 🡺 lim )

717: lim

0080DE ( matrix 🡺 matrix' matrix' )

718: qr

0680AB xrpm ( 🡺 u )

719: rpm

047314 ˜xrref ( [[]] 🡺 [pp] [[]]' )

720: rref

3D56B x| ( symb {var val ...} 🡺 x' )

721: |

3A442 xSQRT ( x 🡺 x' )

722: √

3D434 x∫ ( x1 x2 symb var 🡺 symb' )

723: ∫

3D503 xSUM ( var n1 n2 symb 🡺 x )

724: ∑

3DDC4 xSIGMA+ ( x 🡺 )

( x1...xn 🡺 )

725: ∑+

3DDEE xSIGMA- ( 🡺 x )

( 🡺 [] )

726: ∑-

3E156 xSIGMALINE ( 🡺 symb )

727: ∑LINE

0590AB x∑LIST ( {} 🡺 x )

728: ∑LIST

3DE5A xSUMX ( 🡺 xsum )

729: ∑X

3DE90 xSUMX2 ( 🡺 xsum )

730: ∑X2

3DEC6 xSUMXY ( 🡺 xsum )

731: ∑XY

3DE75 xSUMY ( 🡺 xsum )

732: ∑Y

3DEAB xSUMY2 ( 🡺 xsum )

733: ∑Y2

**Direcc. Nombre Descripción**

3E823 xSTO> ( ob id 🡺 )

( ob symb 🡺 )

Like xSTO, but if the level 1 argument is symbolic, use the first element of it as the variable to write to.

734: ►

39AC7 xPI ( 🡺 π )

735: π

3D202 x∂ ( symb var 🡺 symb' )

736: ∂

3CF80 x<=? ( x y 🡺 1 )

( x y 🡺 0 )

737: ≤

3D01F x>=? ( x y 🡺 1 )

( x y 🡺 0 )

738: ≥

3CD21 x#? ( x y 🡺 1 )

( x y 🡺 0 )

739: ≠

3885C xRPN-> ( ob1 .. obn 🡺 )

740: 🡪

3BE9B x>ARRY ( x1..xn n 🡺 [] )

( x11...xnm {n m} 🡺 [[]] )

741: 🡪ARRY

0380AB x\->COL ( [[]] 🡺 [v1]...[vn] n )

( [] 🡺 x1...xn n )

742: 🡪COL

39104 xSETDATE ( date 🡺 )

743: 🡪DATE

03A0AB x\->DIAG ( [[]] 🡺 vec )

744: 🡪DIAG

0020DD x\->FONT ( font 🡺 )

745: 🡪FONT

3C8A1 x>GROB ( ob n\_chrsize 🡺 grob )

746: 🡪GROB

0040DD x\->HEADER ( n 🡺 )

747: 🡪HEADER

3B0EC x>HMS ( x 🡺 x' )

748: 🡪HMS

06C0AB x\->KEYTIME ( ticks 🡺 )

749: 🡪KEYTIME

0000DD x\->LANGUAGE ( n 🡺 )

750: 🡪LANGUAGE

3C881 x>LCD ( grob 🡺 )

751: 🡪LCD

3B7D2 x>LIST ( ob1 .. obn n 🡺 {} )

752: 🡪LIST

0110DD x\->MINIFONT ( font 🡺 )

753: 🡪MINIFONT

0060DD x\->NDISP ( n 🡺 )

754: 🡪NDISP

**Direcc. Nombre Descripción**

39785 x>NUM ( x 🡺 x' )

755: 🡪NUM

3DA3E x->Q ( x 🡺 a/b )

756: 🡪Q

3DA63 x->QPI ( x 🡺 symb )

757: 🡪Qπ

0360AB x\->ROW ( [[]] 🡺 [1]...[n] n )

( [] 🡺 x1...xn n )

758: 🡪ROW

3BBBE x>STR ( ob 🡺 $ )

759: 🡪STR

3EFB1 x->TAG ( ob tag 🡺 :tag:ob )

760: 🡪TAG

39124 xSETTIME ( time 🡺 )

761: 🡪TIME

38FB5 x>UNIT ( x u 🡺 u' )

762: 🡪UNIT

3C2D6 x>V2 ( x y 🡺 [] )

( x y 🡺 () )

763: 🡪V2

3C30A x>V3 ( x y z 🡺 [] )

764: 🡪V3

3DB04 xMATCHDN ( symb {spat srepl} 🡺 symb' 0/1 )

( symb {spat srepl scond} 🡺 symb' 0/1 )

765: ↓MATCH

3DAD0 xMATCHUP ( symb {spat srepl} 🡺 symb' 0/1 )

( symb {spat srepl scond} 🡺 symb' 0/1 )

766: ↑MATCH

0550AB xΔLIST ( {} 🡺 {}' )

767: ΔLIST

05A0AB ( {} 🡺 x )

768: ΠLIST

08A314 ˜x∞ ( 🡺 '+∞' )

769: ∞

389B9 x<< ( 🡺 )

770: «

389D4 x>> ( 🡺 )

771: »

Apéndice F  
Mensajes de Error

En este apéndice se muestran todos los mensajes de error de la calculadora HP, incluso cuando la mayoría no tiene nada que ver con errores. Es posible generar errores con estos mensajes directamente usando los comandos ERRORSTO o ERROROUT, o llamarlos a la pila como cadenas con el comando JstGetTHEMESG. (capítulo 23). Los números listados están en base hexadecimal. Están actualizados a la versión de ROM 2.15 y en idioma español.

Para generar una lista de mensajes de error como esta, puedes ejecutar el siguiente programa en el emulador y luego pegarlo a un documento como éste.

::

BINT2 ( #2 )

>LANGUAGE ( )

DOHEX ( )

NULL$ ( $ )

1000 ( $ #final ) ( Puedes cambiarlo )

BINT1 ( $ #final #inicial ) ( Puedes cambiarlo )

DO

INDEX@ ( $ #i )

DO>STR ( $ "x #####h" )

BINT3 ( $ "x #####h" #3 )

OVERLEN$ ( $ "x #####h" #3 #long )

#1- ( $ "x #####h" #3 #long-1 )

SUB$ ( $ "#####" )

DUP ( $ "#####" "#####" )

DISPROW1 ( $ "#####" )

"\09" &$ ( $ "#####\09" )

INDEX@ ( $ "#####\09" #i )

JstGETTHEMSG ( $ "#####\09" $msj )

DUPNULL$? ( $ "#####\09" $msj flag )

ITE

2DROP

:: ( $ "#####\09" $msj )

"\0A"

"\0A\09" ( $ "#####\09" $msj "\0A" "\0A\09" )

FLASHPTR 00F 01A ( $ "#####\09" $msj' % )

DROP ( $ "#####\09" $msj' )

&$ ( $ "#####\09mensaje" )

&$ ( "...i\09mensaje" )

NEWLINE$&$ ( "...i\09mensaje\0A" )

;

LOOP

( $ )

;

**#err Mensaje**

1 Memoria insuficiente

2 Directorio recursivo

3 Nombre local indefin.

4 Nombre XLIB indefinido

5 Memoria borrada

6 Corte de corriente

7 Atención:

8 Tarjeta: datos invál.

9 Objeto en uso

A Puerta no disponible

B Sin espacio en puerta

C No objeto en puerta

D Recuperando memoria

E Intento recup.memoria?

F Reinserte RAM,pulse ON

10 No mem. p.config. todo

11 Nombre FPTR indefin.

12 Invalid Bank Datos

13 Full Check Bad CRC

14 Cmprs: not a user bank

15 No or 2 system bank

16 Invalid bank

17 Invalid bank number

18 Inexisting pack

19 Pack twice

1A Ins. Mem.

1B Erase Fail, Rom faulty

1C Erase Fail, Low bats

1D Erase Fail, Locked Block

1E Write Adr outside ROM

1F Write Fail, Rom Faulty

20 Write Fail, Low bats

21 Write Fail, Locked Block

22 Invalid DOS Name

23 File already opened

24 Invalid File Handle

25 Invalid File Index

26 Invalid File Mode

27 Disk Full

28 Disk Format Error

29 Disk Change

2A No SD card inserted

2B Not enough ARM memory

2C DOS call unsupported

2D DOS unknown error

2E Disk Protected

101 No puedo guardar pila

102 Impos.revisar caráct.0

103 Función usuario incorr

104 Ecuación inexistente

106 Sintaxis incorrntos

107 Número real

108 Número complejo

109 Cadena

10A Forma.núms.reales

10B Forma.núms.complej

10C Lista

10D Global Name

10E Local Name

10F Program

110 Algebraic

111 Binary Integer

112 Graphic

113 Tagged

114 Unit

115 XLIB Name

116 Directory

117 Library

118 Backup

119 Function

11A Command

11B Val.bin.de sistema

11C Número real largo

11D Núm.complejo largo

11E Formación encaden.

11F Carácter

120 Cóodigo

121 Datos, biblioteca

122 Externo

124 LAST STACK desactivado

125 LAST CMD desactivado

126 HALT no permitido

127 Formación

128 Argumentos:núm incorr.

129 Referencia circular

12A Director. no permitido

12B Directorio no vacío

12C Definición incorrecta

12D Falta biblioteca

12E PPAR inválido

12F Resultado: núm.no real

130 Imposible aislar

131 No espacio para pila

132 Attención:

133 Error:

134 ¿Borrar?

135 No queda memoria

136 Pila

137 Última pila

138 Últimos comandos

139 Asignación de teclas

13A Alarmas

13B Últimos argumentos

13C Conflicto de nombre

13D Línea de comando

13F Interrupted

140 Integer

141 Symbolic Matrix

142 Font

143 Aplet

144 Extended Real

145 Extended Complex

146 FlashPtr

147 Extended Ptr

148 MiniFont

149 Extended 1

14A Extended 2

14B Extended 3

14C YES

14D NO

14E TRUE

14F FALSE

150 Are you sure?

151 Low Memory Condition

Please Wait...

152 CATALOG

153 Nonexistent Find Pattern

154 Not Found

155 Nonexistent Replace Pattern

156 Can't Find Selection

157 Y= not available

158 Warning:

Changes will not be saved

159 Result not editable in EQW

201 Muy pocos argumentos

202 Argumento incorrecto

203 Argumento:valor incorr

204 Nombre no definido

205 LASTARG desactivado

206 Subexpresión

incompleta

207 ( ) implícitos apagados

208 ( ) implícitos activados

301 Desborde p.defecto,pos

302 Desborde p.defecto,neg

303 Desborde por exceso

304 Resultado indefinido

305 Resultado infinito

501 Dimensión inválida

502 Elemento inválido

503 Suprimiendo línea

504 Suprimiendo columna

505 Insertando línea

506 Insertando columna

601 Datos ∑ inválidos

602 ∑DAT inexistente

603 Datos ∑ insuficient.

604 ∑PAR inválido

605 ∑DAT inválidos:LN(Neg)

606 ∑DAT inválidos: LN(0)

607 EQ inválido

608 Ecuación actual

609 No hay ecuación actual

60A Introd.ecuac,pulse NEW

60B Nombre la ecuación,

pulse ENTER

60C Elija tipo de trazado

60D Catálogo vacío

60E indefinido

60F No hay datos en ∑DAT

610 Auto-ajuste de escala

611 Resolviendo:

612 Sin datos. Introduzca

613 los valores, pulse∑+

614 Seleccione un modelo

615 No alarmas pendientes

616 Pulse ALRM para crear

617 Próxima alarma:

618 Alarme pasada:

619 Alarma reconocida

61A Intro.alarma,pulse SET

61B Selecc.interval.repet.

61C Menú,configuración E/S

61D Traza.tipo:

61E ""

61F (FUERA DE PANTALLA)

620 PTYPE INVÁLIDO

621 Nombre datos estadíst.

pulse ENTER

622 Intro.valor(Zoom fuera

si >1), pulse ENTER

623 Copiado en la pila

624 Zoom al eje X con AUTO

625 Zoom al eje X.

626 Zoom al eje Y.

627 Zoom a los ejes X y Y.

628 IR/Cable:

629 ASCII/binario:

62A Baud:

62B Paridad:

62C Tipo de Checksum:

62D Código de traducción:

62E Intro.matriz,luego NEW

62F No Associated Numeric View

701 Algebraic

702 RPN

703 Standard

704 Std

705 Fixed

706 Fix

707 Scientific

708 Sci

709 Engineering

70A Eng

70B Degrees

70C Radians

70D Grads

70E Rectangular

70F Polar

710 Spherical

711 Operating Mode¬

712 Number Format¬¬

713 Angle Measure¬¬

714 Coord System¬¬¬

715 FM,

716 Beep

717 Key Click

718 Last Stack

719 Choose calculator operating mode

71A Choose number display format

71B Choose decimal places to display

71C Choose angle measure

71D Choose coordinate system

71E Use comma as fraction mark?

71F Enable standard beep?

720 Enable key click?

721 Save last stk for UNDO and ANS?

722 CALCULATOR MODES

723 Font:

724 Stack:

725 Small

726 Textbook

727 Edit:

728 Small

729 Full Page

72A Indent

72B EQW:

72C Small

72D Small Stack Disp

72E Header:

72F Clock

730 Analog

731 Choose system font

732 Display stack using small font?

733 Use pretty print in the stack?

734 Edit using small font?

735 Edit in full page?

736 Automatically indent new lines?

737 Edit in EQW using small font?

738 Display EQW using small font?

739 Choose header height

73A Display ticking clock?

73B Analog clock?

73C DISPLAY MODES

73D Indep var:

73E Modulo:

73F Verbose

740 Step/Step

741 Complex

742 Approx

743 Incr Pow

744 Simp Non-Rational

745 Rigorous

746 Numeric

747 Enter independent variable name

748 Enter modulo value

749 Display calculus information?

74A Perform operations step by step?

74B Allow complex numbers?

74C Perform approx calculations?

74D Increasing polynomial ordering?

74E Simplify non rational expr?

74F Don't simplify |X| to X?

750 Replace constants by values?

751 CAS MODES

752 Goto row:

753 Goto column:

754 Specify a row to go to

755 Specify a column to go to

756 Matrix Writer

757 Bad range value

758 Start:

759 Step:

75A Type:

75B Zoom:

75C Small Font

75D File:

75E Enter starting value

75F Enter increment value

760 Choose table format

761 Enter zoom factor

762 Display table using small font?

763 Enter a filename to save data

764 TABLE SETUP

765 Automatic

766 Build Your Own

767 Function

768 Polar

769 Parametric

76A Diff Eq

76B Conic

76C Truth

76D Histogram

76E Bar

76F Scatter

770 Slopefield

771 Fast3D

772 Wireframe

773 Ps-Contour

774 Y-Slice

775 Gridmap

776 Pr-Surface

777 Deg

778 Rad

779 Grad

77A Type:

77B ∠:

77C EQ:

77D Indep:

77E Connect

77F Simult

780 H-Tick:

781 V-Tick:

782 Pixels

783 Depnd:

784 Save Animation

785 ∑DAT:

786 Col:

787 Cols:

788 F:

789 H-Var:

78A V-Var:

78B Stiff

78C ∂F∂Y:

78D ∂F∂T:

78E Choose type of plot

78F Choose angle measure

790 Enter function(s) to plot

791 Enter independent variable name

792 Connect plot points?

793 Plot functions simultaneously?

794 Enter horizontal tick spacing

795 Enter vertical tick spacing

796 Tick spacing units are pixels?

797 Enter dependent variable name

798 Save slices animation?

799 Enter data to plot

79A Enter col to use for horizontal

79B Enter col to use for vertical

79C Enter horizontal variable

79D Enter vertical variable

79E Use stiff diff eq solver?

79F Enter derivative w.r.t. soln

7A0 Enter derivative w.r.t. indep

7A1 PLOT SETUP

7A2 H-View:

7A3 V-View:

7A4 Indep Low:

7A5 High:

7A6 Step:

7A7 Pixels

7A8 Depnd Low:

7A9 High:

7AA X-Left:

7AB X-Right:

7AC Y-Near:

7AD Y-Far:

7AE Step Indep:

7AF Depnd:

7B0 Bar Width:

7B1 Z-Low:

7B2 Z-High:

7B3 XE:

7B4 YE:

7B5 ZE:

7B6 Init:

7B7 Final:

7B8 Init-Soln:

7B9 Tol:

7BA XXLeft:

7BB XXRight:

7BC YYNear:

7BD YYFar:

7BE Enter minimum horizontal value

7BF Enter maximum horizontal value

7C0 Enter minimum vertical value

7C1 Enter maximum vertical value

7C2 Enter minimum indep var value

7C3 Enter maximum indep var value

7C4 Enter indep var increment

7C5 Indep step units are pixels?

7C6 Enter minimum depend var value

7C7 Enter maximum depend var value

7C8 Enter bar width

7C9 Enter minimum Z view-volume val

7CA Enter maximum Z view-volume val

7CB Enter X eyepoint coordinate

7CC Enter Y eyepoint coordinate

7CD Enter Z eyepoint coordinate

7CE Enter absolute error tolerance

7CF Enter minimum XX range value

7D0 Enter maximum XX range value

7D1 Enter minimum YY range value

7D2 Enter maximum YY range value

7D3 PLOT WINDOW

7D4 Default

7D5 FUNCTION

7D6 POLAR

7D7 PARAMETRIC

7D8 DIFF EQ

7D9 CONIC

7DA TRUTH

7DB HISTOGRAM

7DC BAR

7DD SCATTER

7DE SLOPEFIELD

7DF FAST3D

7E0 WIREFRAME

7E1 PS-CONTOUR

7E2 Y-SLICE

7E3 GRIDMAP

7E4 PR-SURFACE

7E5 PLOT WINDOW -

7E6 Enter minimum X view-volume val

7E7 Enter maximum X view-volume val

7E8 Enter minimum Y view-volume val

7E9 Enter maximum Y view-volume val

7EA Enter indep var sample count

7EB Enter depnd var sample count

7EC Goto Level:

7ED Specify a level to go to

7EE HISTORY

801 Must be >= 0

802 Must be bewteen 0 and 1

803 µ0:

804 :

805 N:

806 α:

807 σ:

808 Null hypothesis population mean

809 Sample mean

80A Sample Size

80B Significance level

80C Population standard deviation

80D Z-TEST: 1 µ, KNOWN σ

80E Alternative Hypothesis

80F 1:

810 σ1:

811 N1:

812 α:

813 2:

814 σ2:

815 N2:

816 Sample mean for population 1

817 Std deviation for population 1

818 Sample size for population 1

819 Significance level

81A Sample mean for population 2

81B Std deviation for population 2

81C Sample size for population 2

81D Z-TEST: 2 µ, KNOWN σ

81E π0:

81F x:

820 N:

821 α:

822 Null hyp. population proportion

823 Success count

824 Sample size

825 Significance level

826 Z-TEST: 1 P

827 X1:

828 N1:

829 α:

82A X2:

82B N2:

82C Success count for sample 1

82D Size of sample 1

82E Significance level

82F Success count for sample 2

830 Size of sample 2

831 Z-TEST: 2 P

832 :

833 Sx:

834 µ0:

835 α:

836 N:

837 Null hypothesis population mean

838 Sample Standard deviation

839 Sample Mean

83A Significance level

83B Sample size

83C T-TEST: 1 µ, UNKNOWN σ

83D 1:

83E S1:

83F N1:

840 α:

841 2:

842 S2:

843 N2:

844 Pooled?

845 Sample mean for population 1

846 Std deviation for sample 1

847 Sample size for population 1

848 Significance level

849 Sample mean for population2

84A Std deviation for sample 2

84B Sample size for population 2

84C "Pooled" if checked

84D T-TEST: 2 µ, UNKNOWN σ

84E :

84F σ:

850 N:

851 C:

852 Sample mean

853 Population standard deviation

854 Sample size

855 Confidence level

856 CONF. INT.: 1 µ, KNOWN σ

857 1:

858 σ1:

859 N1:

85A C:

85B 2:

85C σ2:

85D N2:

85E Sample mean for population 1

85F Std deviation for sample 1

860 Size of sample 1

861 Sample mean for population 2

862 Std deviation for sample 2

863 Size of sample 2

864 Confidence level

865 CONF. INT.: 2 µ, KNOWN σ

866 x:

867 N:

868 C:

869 Sample success count

86A Sample size

86B Confidence level

86C CONF. INT.: 1 P

86D 1:

86E N1:

86F C:

870 2:

871 N2:

872 Sample 1 success count

873 Sample 1 size

874 Sample 2 success count

875 Sample 2 size

876 Confidence level

877 CONF. INT.: 2 P

878 :

879 Sx:

87A N:

87B C:

87C Sample mean

87D Sample standard deviation

87E Sample size

87F Confidence level

880 CONF. INT.: 1 µ, UNKNOWN σ

881 1:

882 S1:

883 N1:

884 C:

885 2:

886 S2:

887 N2:

888 Pooled

889 Sample 1 mean

88A Std deviation for sample 1

88B Sample 1 size

88C Sample 2 mean

88D Std deviation for sample 2

88E Sample 2 size

88F Confidence level

890 Pooled if checked

891 CONF. INT.: 2 µ, UNKNOWN σ

892 Search for:

893 Replace by:

894 Case Sensitive

895 Search For:

896 Enter search pattern

897 Enter replace pattern

898 Case sensitive search?

899 Enter search pattern

89A FIND REPLACE

89B FIND

89C Goto Line:

89D Specify a line to go to

89E GOTO LINE

89F Goto Position:

8A0 Specify a position to go to

8A1 GOTO POSITION

8A2 H-Factor:

8A3 V-Factor:

8A4 Recenter on cursor

8A5 Enter horizontal zoom factor

8A6 Enter vertical zoom factor

8A7 Recenter plot on cursor?

8A8 ZOOM FACTOR

8A9 Object:

8AA Name:

8AB Directory

8AC Enter New Object

8AD Enter variable name

8AE Create a new directory?

8AF NEW VARIABLE

8B0 Select Object

901 Tests the null hypothesis that the population mean is a given value, H0: µ= µ0, against an alternative hypothesis.

Example data

A set of 50 random numbers from 0 to 1, generated by a calculator, has a mean of 0.461368. The population should have:µ = 0.5 and σ = 0.2887

Calculation

Assume that the standard deviation of the population is 0.2887. Test the null hypothesis, H0: µ = 0.5 against the alternative hypothesis that the mean is less than 0.5, H1:µ < 0.5. Test at the 5% level.

Results

p > 0.05. Accept H0, There is insufficient evidence that the calculator is not functioning properly.

902 Tests the null hypothesis that the population means are equal, H0: µ1=µ2, against an alternative hypothesis. The population standard deviation must be known.

Example data

A set of 50 random numbers from 0 to 1, generated by one calculator, has a mean of 0.461368. A second calculator generates a set of 50 numbers, with a mean of 0.522851. The populations should have µ = 0.5 and σ = 0.2887. Test that these samples indicate that the calculators are operating differently.

Calculation

Test the null hypothesis, H0:µ1 = µ2, against the alternative hypothesis that the means are different, H1: µ1 ‹ µ2. Test at the 5% level.

Results

Since p > 0.05, accept the null hypothesis. Too little evidence to suspect that the calculators are operating differently.

903 Tests the null hypothesis that the proportion of successes in the population is a given value, H0: π=p0, against an alternative hypothesis.

Example data

A set of 50 random numbers between 0 and 1, generated by a calculator. 21 of the numbers are less than 0.5. The population should have π=0.5.

Calculation

Test the alternative hypotheses H1: π < 0.5 against the null hypothesis H0: π = 0.5 at the 5% level.

Result

The test returns a Z-value of -1.1313..., with a probability of 0.1289.... Since this probability is greater than α = 0.05, accept the null hypothesis Evidence is not strong enough to suspect the random number generator is faulty.

904 Tests the null hypothesis that the proportions of success in two populations are equal, H0: π1=π2, against an alternative hypothesis.

Example data

A set of 50 random numbers, between 0 and 1, generated by one calculator. 21 of the numbers are less than 0.5. A second set of 50 random numbers generated by another calculator. 26 of them are less than 0.5.

Calculation

Test the alternative hypotheses H1: π1 < π2 against the null hypothesis H0: π1 = π2 at the 5% level.

Result

The test returns a Z-value of -1.0018..., with a probability of 0.1582.... Since this probability is greater than α = 0.05, accept the null hypothesis. Evidence is not strong enough to suspect that the two calculators are functioning differently.

905 Used when the population standard deviation is not known. Tests the null hypothesis that the population mean is a given value, H0: µ=µ0, against an alternative hypothesis.

Example data

A set of 50 random numbers, between 0 to 1, is generated by a calculator. The sample mean is 0.461368 and the sample standard deviation is 0.2776. Ideally, the mean of the population should be 0.5. Is this sample evidence that the calculator is producing random numbers that are too small?

Calculation

Use a t-test to test the null hypothesis that the mean is 0.5, H0: µ = 0.5, against the alternative hypothesis that the mean is less than 0.5, H1: µ < 0.5. Test at the 5% level.

Results

Since p > 0.05, we accept the null hypothesis. Insufficient evidence to suspect the calculator of improper functioning.

906 Used when the population standard deviation is not known. Tests the null hypothesis that the population means are equal, H0: µ1=µ2, against an alternative hypothesis.

Example data

A set of 50 random numbers from 0 to 1, generated by one calculator, has a mean of 0.461368 and a sample standard deviation of 0.2776. A set of 50 random numbers generated by a second calculator has a mean of 0.522851 and a sample standard deviation of 0.2943.

Calculation

Use a t-test to test the null hypothesis that the means are equal, H0: µ1 = µ2, against the alternative hypothesis that the means are different, H1: µ1 ‹ µ2. Test at the 5% level.

Results

Since p > 0.05, accept the null hypothesis Insufficient evidence to suspect the calculators of behaving differently.

907 Uses Normal distribution to calculate a confidence interval for µ, the true mean of a population, when the true standard deviation, σ is known.

Example data

A set of 50 random numbers between 0 to 1, generated by a calculator.

Sample mean = 0.461368.

The population should have:

µ = 0.5

σ = 0.2887

Calculation

Calculate a 99% true mean confidence interval from the data. The confidence interval should contain 0.5 if the random number generator is true.

Results

The calculated confidence interval is [0.3562, 0.5665]. The probability is .99 that the population mean is in this interval.

908 Uses Normal distribution to calculate a confidence interval for the difference in the means of two populations, when the standard deviations are known.

Example data

Two sets of 50 random numbers between 0 to 1, each generated by a different calculator.

Calculator 1 sample mean

1 =0.461368

Calculator 2 sample mean

2= 0.522851

Each population should have:

µ = 0.5

σ = 0.2887

Calculation

Calculate a 99% confidence interval for the difference in the means of two populations. The confidence interval should contain 0 if the random number generators are operating properly.

Results

The calculated confidence interval is [-.2102, 0.0872]. The probability is .99 that the difference between the population means is in this interval.

909 Uses the Normal distribution to calculate a confidence interval for π, the true proportion of successes in a population, based on the number of successes, X, in a sample of size n.

Example data

A set of 50 random numbers, between 0 and 1, generated by a calculator. 21 of the numbers are less than 0.5.

The population should have π =0.5.

Calculation

Calculate a 99% confidence interval for the true proportion of numbers less than 0.5 produced by this generator. Interval should contain 0.5 if the generator is true.

Results

The calculated confidence interval is [0.2402, 0.6000]. The probability is .99 that the true proportion of numbers less than .5 is in this interval.

90A Uses the Normal distribution to calculate a confidence interval for π1 - π2, the difference of the true proportion of successes in two populations. Calculation is based on the number of successes, X1, in a sample of size n1 from the first population, and the number of successes, X2, in a sample of size n2 from the second population.

Example data

A set of 50 random numbers, between 0 and 1, generated by one calculator. 21 of the numbers are less than 0.5. A second set of 50 random numbers generated by another calculator. 26 numbers are less than 0.5.

Calculation

Calculate a 99% confidence interval for the difference of the true proportions of numbers less than 0.5 produced. The interval should contain 0 if there is no significant difference between the calculators.

Results

The calculated confidence interval is [-.3558, .1558]. The probability is .99 that the difference between the proportions of success of the two populations is in this interval.

90B Uses the Student's t-distribution to calculate a confidence interval for the true mean of a population, when the true population standard deviation, is unknown. The calculation is based on the sample mean and sample standard deviation.

Example data

A set of 50 random numbers from 0 to 1, generated by a calculator. The sample mean is 0.461368 and the sample standard deviation is 0.2776.

Calculation

Calculate a 99% confidence interval for the true mean of population of random numbers generated. If the calculator is operating properly, this interval should contain 0.5.

Results

The calculated confidence interval is [0.3562, 0.5666]. The probability is .99 that the population mean is in this interval.

90C Uses the Student's t-distribution to calculate a confidence interval for the difference in the means of two populations when standard deviations are unknown. The calculation is based on the sample means and the sample standard deviations.

Example data

A set of 50 random numbers from 0 to 1, generated by one calculator, has a mean of 0.461368 and a sample standard deviation of 0.2776. A set of 50 random numbers generated by a second calculator has a mean of 0.522851 and a sample standard deviation of 0.2943.

Calculation

Calculate a 99% confidence interval for the true difference in the means of the populations of random numbers generated by these two calculators.

Results

The calculated confidence interval is [-0.2118, 0.0888]. The probability is .99 that the difference between the population means is in this interval.

90D Inconclusive result

A01 Mala Estimación

A02 ¿Constante?

A03 Interrupción

A04 Cero

A05 Cambio de signo

A06 Extremo

B01 Unidad inválida

B02 Unidades incompatibles

C01 Checksum discorde

C02 Tiempo excedido

C03 Error de recepción

C04 Mem.intermed. excedida

C05 Error de paridad

C06 Transferencia fallida

C07 Error de protocolo

C08 Comando Servidor Invál

C09 Puerta cerrada

C0A Conectando

C0B Nuevo intento n°

C0C Espero comand.Servidor

C0D Enviando

C0E Recibiendo

C0F Objeto desechado

C10 Paquete n°

C11 Procesando el comando

C12 IOPAR inválido

C13 PRTPAR inválido

C14 E/S: pilas, baja carga

C15 Pila vacía

C16 Línea

C17 Nombre inválido

D01 Fecha incorrecta

D02 Hora incorrecta

D03 Repetición inválida

D04 Alarma inexistante

B901 Press [CONT] for menu

B902 reset/delete this field

B903 Reset value

B904 Delete value

B905 Reset all

B906 Valid object types:

B907 Valid object type:

B908 Any object

B909 Real number

B90A (Complex num)

B90B "String"

B90C [ Real array ]

B90D [(Cmpl array)]

B90E { List }

B90F Name

B910 « Program »

B911 'Algebraic'

B912 # Binary int

B913 \_Unit object

B914 Invalid object type

B915 Invalid object value

B916 Calculator Modes

B917 Number Format:

B918 Angle Measure:

B919 Coord System:

B91A Beep

B91B Clock

B91C FM,

B91D Choose number display format

B91E Enter decimal places to display

B91F Choose angle measure

B920 Choose coordinate system

B921 Enable standard beep?

B922 Display ticking clock?

B923 Use comma as fraction mark?

B924 Standard

B925 Std

B926 Fixed

B927 Fix

B928 Scientific

B929 Sci

B92A Engineering

B92B Eng

B92C Degrees

B92D Deg

B92E Radians

B92F Rad

B930 Grads

B931 Grad

B932 Rectangular

B933 Polar

B934 Spherical

B935 SYSTEM FLAGS

B936 01 General solutions

B937 02 Constant → symb

B938 03 Function → symb

B939 14 Payment at end

B93A 19 →V2 → vector

B93B 20 Underflow → 0

B93C 21 Overflow → ±9E499

B93D 22 Infinite → error

B93E 27 'X+Y\*i' → '(X,Y)'

B93F 28 Sequential plot

B940 29 Draw axes too

B941 31 Connect points

B942 32 Solid cursor

B943 33 Transfer via wire

B944 34 Print via IR

B945 35 ASCII transfer

B946 36 RECV renames

B947 37 Single-space prnt

B948 38 Add linefeeds

B949 39 Show I/O messages

B94A 40 Don't show clock

B94B 41 12-hour clock

B94C 42 mm/dd/yy format

B94D 43 Reschedule alarm

B94E 44 Delete alarm

B94F 51 Fraction mark: .

B950 52 Show many lines

B951 53 No extra parens

B952 54 Tiny element → 0

B953 55 Save last args

B954 56 Standard beep on

B955 57 Alarm beep on

B956 58 Show INFO

B957 59 Show variables

B958 60 [α][α] locks

B959 61 [USR][USR] locks

B95A 62 User keys off

B95B 63 Custom ENTER off

B95C 65 All multiline

B95D 66 Stack:x lines str

B95E 67 Digital clock

B95F 68 No AutoIndent

B960 69 Line edit

B961 70 →GROB 1 line str

B962 71 Show addresses

B963 72 Stack:current fnt

B964 73 Edit:current font

B965 74 Right stack disp

B966 75 Key click off

B967 76 Purge confirm

B968 79 Textbook on

B969 80 EQW cur stk font

B96A 81 GRB Alg cur font

B96B 82 EQW edit cur font

B96C 83 Display grobs on

B96D 85 Normal stk disp

B96E 90 CHOOSE:cur font

B96F 91 MTRW:matrix

B970 92 MASD asm mode

B971 94 Result = LASTCMD

B972 95 RPN mode

B973 97 List:horiz disp

B974 98 Vector:horiz disp

B975 99 CAS:quiet

B976 100 Step by step off

B977 103 Complex off

B978 105 Exact mode on

B979 106 Simp. in series

B97A 109 Sym. factorize

B97B 110 Normal matrices

B97C 111 Simp non rat.

B97D 112 i simplified

B97E 113 Linear simp on

B97F 114 Disp 1+x → x+1

B980 115 SQRT simplified

B981 116 Prefer cos()

B982 117 CHOOSE boxes

B983 119 Rigorous on

B984 120 Silent mode off

B985 123 Allow Switch Mode

B986 125 Accur. Sign-Sturm

B987 126 rref w/ last col

B988 127 IrDA mode

B989 128 Cmplx var allowed

B98A 01 Principal value

B98B 02 Constant → num

B98C 03 Function → num

B98D 14 Payment at begin

B98E 19 →V2 → complex

B98F 20 Underflow → error

B990 21 Overflow → error

B991 22 Infinite → ±9E499

B992 27 'X+Y\*i' → 'X+Y\*i'

B993 28 Simultaneous plot

B994 29 Don't draw axes

B995 31 Plot points only

B996 32 Inverse cursor

B997 33 Transfer via IR

B998 34 Print via wire

B999 35 Binary transfer

B99A 36 RECV overwrites

B99B 37 Double-space prnt

B99C 38 No linefeeds

B99D 39 No I/O messages

B99E 40 Show clock

B99F 41 24-hour clock

B9A0 42 dd.mm.yy format

B9A1 43 Don't reschedule

B9A2 44 Save alarm

B9A3 51 Fraction mark: ,

B9A4 52 Show one line

B9A5 53 Show all parens

B9A6 54 Use tiny element

B9A7 55 No last args

B9A8 56 Standard beep off

B9A9 57 Alarm beep off

B9AA 58 Don't show INFO

B9AB 59 Show names only

B9AC 60 [α] locks Alpha

B9AD 61 [USR] locks User

B9AE 62 User keys on

B9AF 63 Custom ENTER on

B9B0 65 Level 1 multiline

B9B1 66 Stk: 1 line str

B9B2 67 Analog clock

B9B3 68 AutoIndent

B9B4 69 Infinite line edit

B9B5 70 →GROB x lines str

B9B6 71 No addresses

B9B7 72 Stack:mini font

B9B8 73 Edit:mini font

B9B9 74 Left stack disp

B9BA 75 Key click on

B9BB 76 No purge confirm

B9BC 79 Textbook off

B9BD 80 EQW mini stk font

B9BE 81 GRB Alg mini font

B9BF 82 EQW edit mini fnt

B9C0 83 Display grobs off

B9C1 85 SysRPL stk disp

B9C2 90 CHOOSE:mini font

B9C3 91 MTRW:list of list

B9C4 92 MASD SysRPL mode

B9C5 94 Result <> LASTCMD

B9C6 95 Algebraic mode

B9C7 97 List:vert disp

B9C8 98 Vector:vert disp

B9C9 99 CAS:verbose

B9CA 100 Step by step on

B9CB 103 Complex on

B9CC 105 Approx. mode on

B9CD 106 !Simp. in series

B9CE 109 Num. factorize

B9CF 110 Large matrices

B9D0 111 !Simp non rat.

B9D1 112 i not simplified

B9D2 113 Linear simp off

B9D3 114 Disp x+1 → 1+x

B9D4 115 SQRT !simplified

B9D5 116 Prefer sin()

B9D6 117 Soft MENU

B9D7 119 Rigorous off

B9D8 120 Silent mode on

B9D9 123 Forb. Switch Mode

B9DA 125 FastSign-no Sturm

B9DB 126 rref w/o last col

B9DC 127 HP-IR mode

B9DD 128 Vars are reals

B9DE Object:

B9DF Obs in

B9E0 Name:

BA01 1.Send to Calculator¬

BA02 2.Get from Calculator

BA03 3.Print display

BA04 4.Print¬

BA05 5.Transfer¬

BA06 6.Start Server

BA07 Enter names of vars to send

BA08 Vars in

BA09 SEND TO CALCULATOR

BA0A Port:

BA0B Dbl-Space

BA0C Delay:

BA0D Xlat:

BA0E Linef

BA0F Baud:

BA10 Parity:

BA11 Len:

BA12 Choose print port

BA13 Enter object(s) to print

BA14 Print extra space between lines?

BA15 Enter delay between lines

BA16 Choose character translations

BA17 Print linefeed between lines?

BA18 Choose baud rate

BA19 Choose parity

BA1A Enter printer line length

BA1B PRINT

BA1C Type:

BA1D OvrW

BA1E Fmt:

BA1F Chk:

BA20 Choose transfer port

BA21 Choose type of transfer

BA22 Enter names of vars to transfer

BA23 Choose transfer format

BA24 Choose checksum type

BA25 Overwrite existing variables?

BA26 TRANSFER

BA27 Local vars

BA28 Remote PC files

BA29 Files in

BA2A Enter name of dir to change to

BA2B Choose Remote Directory

BA2C Infrared

BA2D IR

BA2E Wire

BA2F Kermit

BA30 XModem

BA31 Odd

BA32 Even

BA33 Mark

BA34 Space

BA35 Spc

BA36 ASCII

BA37 ASC

BA38 Binary

BA39 Bin

BA3A None

BA3B Newline (Ch 10)

BA3C Newl

BA3D Chr 128-159

BA3E →159

BA3F →255

BA40 Chr 128-255

BA41 One-digit arith

BA42 Two-digit arith

BA43 Three-digit CRC

BA44 HP-IR

BA45 IrDA

BA46 14K

BA47 19K

BA48 38K

BA49 57K

BA4A 115K

BA4B 15K

BA4C 1200

BA4D 2400

BA4E 4800

BA4F 9600

BA50 USB

BA51 Serial

BB01 1.Single-var¬

BB02 2.Frequencies¬

BB03 3.Fit data¬

BB04 4.Summary stats¬

BB05 SINGLE-VARIABLE STATISTICS

BB06 ∑DAT:

BB07 Type:

BB08 Mean

BB09 Std Dev

BB0A Variance

BB0B Total

BB0C Maximum

BB0D Minimum

BB0E Enter statistical data

BB0F Enter variable column

BB10 Choose statistics type

BB11 Calculate mean?

BB12 Calculate standard deviation?

BB13 Calculate variance?

BB14 Calculate column total?

BB15 Calculate column maximum?

BB16 Calculate column minimum?

BB17 Sample

BB18 Population

BB19 FREQUENCIES

BB1A X-Min:

BB1B Bin Count:

BB1C Bin Width:

BB1D Enter minimum first bin X value

BB1E Enter number of bins

BB1F Enter bin width

BB20 FIT DATA

BB21 X-Col:

BB22 Y-Col:

BB23 Model:

BB24 Enter indep column number

BB25 Enter dependent column number

BB26 Choose statistical model

BB27 Correlation

BB28 Covariance

BB29 PREDICT VALUES

BB2A Y:

BB2B Enter indep value or press PRED

BB2C Enter dep value or press PRED

BB2D SUMMARY STATISTICS

BB2E Calculate:

BB2F ∑X

BB30 ∑Y

BB31 ∑X2

BB32 ∑Y2

BB33 ∑XY

BB34 N∑

BB35 Calculate sum of X column?

BB36 Calculate sum of Y column?

BB37 Calculate sum of squares of X?

BB38 Calculate sum of squares of Y?

BB39 Calculate sum of products?

BB3A Calculate number of data points?

BB3B Linear Fit

BB3C Logarithmic Fit

BB3D Exponential Fit

BB3E Power Fit

BB3F Best Fit

BB40 5.Hypoth. tests¬

BB41 6.Conf. interval¬

BC01 1.Browse alarms¬

BC02 2.Set alarm¬

BC03 3.Set time, date¬

BC04 SET ALARM

BC05 Message:

BC06 Time:

BC07 Date:

BC08 Repeat:

BC09 Enter "message" or « action »

BC0A Enter hour

BC0B Enter minute

BC0C Enter second

BC0D Choose AM, PM, or 24-hour time

BC0E Enter month

BC0F Enter day

BC10 Enter year

BC11 Enter alarm repeat multiple

BC12 Enter alarm repeat unit

BC13 SET TIME AND DATE

BC14 Choose date display format

BC15 Monday

BC16 Tuesday

BC17 Wednesday

BC18 Thursday

BC19 Friday

BC1A Saturday

BC1B Sunday

BC1C None

BC1D AM

BC1E PM

BC1F 24-hour time

BC20 24-hr

BC21 1 January

BC22 2 February

BC23 3 March

BC24 4 April

BC25 5 May

BC26 6 June

BC27 7 July

BC28 8 August

BC29 9 September

BC2A 10 October

BC2B 11 November

BC2C 12 December

BC2D Week

BC2E Day

BC2F Hour

BC30 Minute

BC31 Second

BC32 Weeks

BC33 Days

BC34 Hours

BC35 Minutes

BC36 Seconds

BC37 Month/Day/Year

BC38 M/D/Y

BC39 Day.Month.Year

BC3A D.M.Y

BC3B ALARMS

BD01 1.Integrate¬

BD02 2.Differentiate¬

BD03 3.Taylor poly¬

BD04 4.Isolate var¬

BD05 5.Solve quad¬

BD06 6.Manip expr¬

BD07 INTEGRATE

BD08 Expr:

BD09 Var:

BD0A Result:

BD0B Enter expression

BD0C Enter variable name

BD0D Enter lower limit

BD0E Enter upper limit

BD0F Choose result type

BD10 Choose disp format for accuracy

BD11 DIFFERENTIATE

BD12 Value:

BD13 Enter variable value

BD14 Expression

BD15 TAYLOR POLYNOMIAL

BD16 Order:

BD17 Enter Taylor polynomial order

BD18 ISOLATE A VARIABLE

BD19 Principal

BD1A Get principal solution only?

BD1B SOLVE QUADRATIC

BD1C MANIPULATE EXPRESSION

BD1D MATCH EXPRESSION

BD1E Pattern:

BD1F Replacement:

BD20 Subexpr First

BD21 Cond:

BD22 Enter pattern to search for

BD23 Enter replacement object

BD24 Search subexpressions first?

BD25 Enter conditional expression

BD26 Symbolic

BD27 Numeric

BE01 Plot

BE02 Type:

BE03 ∠:

BE04 H-View:

BE05 Autoscale

BE06 V-View:

BE07 Choose type of plot

BE08 Choose angle measure

BE09 Enter function(s) to plot

BE0A Enter minimum horizontal value

BE0B Enter maximum horizontal value

BE0C Autoscale vertical plot range?

BE0D Enter minimum vertical value

BE0E Enter maximum vertical value

BE0F Plot (x(t), y(t))

BE10 Enter complex-valued func(s)

BE11 Plot y'(t)=f(t,y)

BE12 Enter function of INDEP and SOLN

BE13 Enter derivative w.r.t. SOLN

BE14 Enter derivative w.r.t. INDEP

BE15 Use Stiff diff eq solver?

BE16 ∑Dat:

BE17 Col:

BE18 Wid:

BE19 Enter data to plot

BE1A Arrays in

BE1B Enter column to plot

BE1C Enter bar width

BE1D Cols:

BE1E Enter col to use for horizontal

BE1F Enter col to use for vertical

BE20 Steps:

BE21 Enter indep var sample count

BE22 Enter dep var sample count

BE23 Plot Options

BE24 Lo:

BE25 Hi:

BE26 Axes

BE27 Simult

BE28 Connect

BE29 Pixels

BE2A H-Tick:

BE2B V-Tick:

BE2C Enter minimum indep var value

BE2D Enter maximum indep var value

BE2E Draw axes before plotting?

BE2F Connect plot points?

BE30 Plot functions simultaneously?

BE31 Enter indep var increment

BE32 Indep step units are pixels?

BE33 Enter horizontal tick spacing

BE34 Enter vertical tick spacing

BE35 Tick spacing units are pixels?

BE36 Depnd:

BE37 Enter dependent var name

BE38 Enter minimum dep var value

BE39 Enter maximum dep var value

BE3A H-Var:

BE3B V-Var:

BE3C Enter max indep var increment

BE3D Choose horizontal variable

BE3E Choose vertical variable

BE3F 0 INDEP

BE40 1 SOLN

BE41 SOLN(

BE42 X-Left:

BE43 X-Right:

BE44 Y-Near:

BE45 Y-Far:

BE46 Z-Low:

BE47 Z-High:

BE48 Enter minimum X view-volume val

BE49 Enter maximum X view-volume val

BE4A Enter minimum Y view-volume val

BE4B Enter maximum Y view-volume val

BE4C Enter minimum Z view-volume val

BE4D Enter maximum Z view-volume val

BE4E XE:

BE4F YE:

BE50 ZE:

BE51 Enter X eyepoint coordinate

BE52 Enter Y eyepoint coordinate

BE53 Enter Z eyepoint coordinate

BE54 Save Animation

BE55 Save animation data after plot?

BE56 XX-Left:

BE57 XX-Rght:

BE58 YY-Near:

BE59 YY-Far:

BE5A Enter minimum XX range value

BE5B Enter maximum XX range value

BE5C Enter minimum YY range value

BE5D Enter maximum YY range value

BE5E XX and YY Plot Options

BE5F Zoom Factors

BE60 H-Factor:

BE61 V-Factor:

BE62 Recenter at Crosshairs

BE63 Enter horizontal zoom factor

BE64 Enter vertical zoom factor

BE65 Recenter plot at crosshairs?

BE66 Reset plot

BE67 Dflt

BE68 Auto

BE69 Function

BE6A Polar

BE6B Conic

BE6C Truth

BE6D Parametric

BE6E Diff Eq

BE6F Histogram

BE70 Bar

BE71 Scatter

BE72 Slopefield

BE73 Wireframe

BE74 Ps-Contour

BE75 Y-Slice

BE76 Gridmap

BE77 Pr-Surface

BF01 1.Solve equation¬

BF02 2.Solve diff eq¬

BF03 3.Solve poly¬

BF04 4.Solve lin sys¬

BF05 5.Solve finance¬

BF06 SOLVE EQUATION

BF07 Enter value or press SOLVE

BF08 Eq:

BF09 Enter function to solve

BF0A Funcs in

BF0B Solver Variable Order

BF0C Variables:

BF0D Enter order of vars to display

BF0E SOLVE Y'(T)=F(T,Y)

BF0F f:

BF10 ∂f∂y:

BF11 ∂f∂t:

BF12 Indep:

BF13 Init:

BF14 Final:

BF15 Soln:

BF16 Tol:

BF17 Step:

BF18 Stiff

BF19 Enter function of INDEP and SOLN

BF1A Enter derivative w.r.t. SOLN

BF1B Enter derivative w.r.t. INDEP

BF1C Enter independent var name

BF1D Enter initial indep var value

BF1E Enter final indep var value

BF1F Enter solution var name

BF20 Enter initial solution var value

BF21 Press SOLVE for final soln value

BF22 Enter absolute error tolerance

BF23 Enter initial step size

BF24 Calculate stiff differential?

BF25 f

BF26 Tolerance

BF27 Solution

BF28 SOLVE AN•X^N+¬+A1•X+A0

BF29 Coefficients [ an ¬ a1 a0 ]:

BF2A Roots:

BF2B Enter coefficients or press SOLVE

BF2C Enter roots or press SOLVE

BF2D Coefficients

BF2E Roots

BF2F SOLVE SYSTEM A•X=B

BF30 A:

BF31 B:

BF32 X:

BF33 Enter coefficients matrix A

BF34 Enter constants or press SOLVE

BF35 Enter solutions or press SOLVE

BF36 Constants

BF37 Solutions

BF38 N:

BF39 I%YR:

BF3A PV:

BF3B PMT:

BF3C P/YR:

BF3D FV:

BF3E Enter no. of payments or SOLVE

BF3F Enter yearly int rate or SOLVE

BF40 Enter present value or SOLVE

BF41 Enter payment amount or SOLVE

BF42 Enter no. of payments per year

BF43 Enter future value or SOLVE

BF44 Choose when payments are made

BF45 TIME VALUE OF MONEY

BF46 N

BF47 I%/YR

BF48 PV

BF49 PMT

BF4A FV

BF4B End

BF4C Begin

BF4D Beg

BF4E AMORTIZE

BF4F Payments:

BF50 Principal:

BF51 Interest:

BF52 Balance:

BF53 Enter no. of payments to amort

BF54 Principal

BF55 Interest

BF56 Balance

C001 Unable to find root

DE01 denominator(s)

DE02 root(s)

DE03 last

DE04 obvious

DE05 factorizing

DE06 value

DE07 test(s)

DE08 searching

DE09 TAYLR of ↓ at

DE0A nth

DE0B is

DE0C numerator(s)

DE0D Less than

DE0E multiplicity

DE0F list of

DE10 at

DE11 factor(s)

DE12 Eigenvalues

DE13 Computing for

DE14 Root mult <

DE15 Numerical to symbolic

DE16 Invalid operator

DE17 Result:

DE18 Pivots

DE19 Press CONT to go on

DE1A Test

DE1B To be implemented

DE1C Unable to factor

DE1D Z is not = 1 mod 4

DE1E Z is not prime

DE1F Empty {} of equations

DE20 Not reducible to a rational expression

DE21 Non unary operator

DE22 User function

DE23 Non isolable operator

DE24 Not exact system

DE25 Parameters not allowed

DE26 CAS internal error

DE27 Invalid ^ for SERIES

DE28 Operator not implemented (SERIES)

DE29 No variable in expr.

DE2A No solution found

DE2B Invalid derivation arg

DE2C No solution in ring

DE2D Not a linear system

DE2E Can't derive int. var

DE2F Diff equation order>2

DE30 INT:invalid var change

DE31 Mode switch cancelled

DE32 No name in expression

DE33 Invalid user function

DE34 Can't find ODE type

DE35 Integer too large

DE36 Unable to find sign

DE37 Non-symmetric matrix

DE38 ATAN insufficent order

DE39 ASIN at infinity undef

DE3A Unsigned inf error

DE3B LN[Var] comparison err

DE3C Undef limit for var

DE3D Bounded var error

DE3E Got expr. indep of var

DE3F Can't state remainder

DE40 LN of neg argument

DE41 Insufficient order

DE42 ABS of non-signed 0

DE43 Numeric input

DE44 Singularity! Continue?

DE45 Cancelled

DE46 Negative integer

DE47 Parameter is cur. var. dependent

DE48 Unsimplified sqrt

DE49 Non polynomial system

DE4A Unable to solve ODE

DE4B Array dimension too large

DE4C Unable to reduce system

DE4D Complex number not allowed

DE4E Polyn. valuation must be 0

DE4F Mode switch not allowed here

DE50 Non algebraic in expression

DE51 Purge current variable

DE52 Reduction result

DE53 Matrix not diagonalizable

DE54 Int[u'\*F(u)] with u=

DE55 Int. by part u'\*v, u=

DE56 Square root

DE57 Rational fraction

DE58 Linearizing

DE59 Risch alg. of tower

DE5A Trig. fraction, u=

DE5B Unknown operator (DOMAIN)

DE5C Same points

DE5D Unsigned inf. Solve?

DE5E CAS not available

DE5F Can not store current var

DE60 Not available on the HP40G

DE61 Not available on the HP49G

DE62 SERIES remainder is O(1) at order 3

DE63 Delta/Heaviside not available from HOME

DE64 Warning, integrating in approx mode

DE65 Function is constant

DE66 Can not unbind local vars

DE67 Replacing strict with large inequality

DE68 No valid environment stored

DF01 Administrador archivos

DF02 NO

DF03 ABORT

DF04 TODO

DF05 SI

DF06 REN

DF07 Ya existe

DF08 Sobreescribir ?

DF09 Renombrar

DF0A ELEGIR DESTINO

DF0B Estás Seguro?

DF0C Modo Búsqueda Inactivo

DF0D Modo Búsqueda Activo

DF0E Nuevo Directorio?

DF0F Clasificar por:

DF10 Original

DF11 Tipo

DF12 Nombre

DF13 Tamaño

DF14 Inv. Tipo

DF15 Inv. Nombre

DF16 Inv. Tamaño

DF17 Enviando con Xmodem:

DF18 EDITA

DF19 COPIA

DF1A MOVER

DF1B RCL

DF1C EVALU

DF1D ARBOL

DF1E BORRA

DF1F RENOM

DF20 NUEVO

DF21 ORDEN

DF22 ENVIA

DF23 RECIB

DF24 PARAR

DF25 VER

DF26 EDITB

DF27 CABEC

DF28 LISTA

DF29 CLASI

DF2A XENVI

DF2B CHDIR

DF2C CANCL

DF2D OK

DF2E CHECK

DF2F ATENCION: El formato

borrarÃ¡ la tarjeta SD

DF30 Desea continuar?

DF31 FORMAT

DF32 Espere por favor...

E101 Avogadro's number

E102 Boltzmann

E103 molar volume

E104 universal gas

E105 std temperature

E106 std pressure

E107 Stefan-Boltzmann

E108 speed of light

E109 permittivity

E10A permeability

E10B accel of gravity

E10C gravitation

E10D Planck's

E10E Dirac's

E10F electronic charge

E110 electron mass

E111 q/me ratio

E112 proton mass

E113 mp/me ratio

E114 fine structure

E115 mag flux quantum

E116 Faraday

E117 Rydberg

E118 Bohr radius

E119 Bohr magneton

E11A nuclear magneton

E11B photon wavelength

E11C photon frequency

E11D Compton wavelen

E11E 1 radian

E11F 2π radians

E120 ∠ in trig mode

E121 Wien's

E122 k/q

E123 “0/q

E124 q\*“0

E125 dielectric const

E126 SiO2 dielec cons

E127 ref intensity

E128 CONSTANTS LIBRARY

E129 Undefined Constant

E301 Starting Solver

E302 OF

E303 Keyword Conflict

E304 No Picture Available

E305 NEAR

E306 MINE

E307 MINES

E308 SCORE:

E309 YOU MADE IT!!

E30A YOU BLEW UP!!

E30B Need ROM >= 208

E401 Invalid Mpar

E402 Single Equation

E403 EQ Invalid for MINIT

E404 Too Many Unknowns

E405 All Variables Known

E406 Illegal During MROOT

E407 Solving for

E408 Searching

E501 Bad Molecular Formula

E502 Undefined Element

E503 Undefined Property

E601 No Solution

E602 Many or No Solutions

E603 I%YR/PYR ‰ -100

E604 Invalid N

E605 Invalid PYR

E606 Invalid #Periods

E607 Undefined TVM Variable

E608 END mode

E609 BEGIN mode

E60A payments/year

E60B Principal

E60C Interest

E60D Balance

FF01 StreamSmart Setup

FF02 Plot Setup

FF03 Sensor Setup

FF04 Unit Setup

FF05 Export Setup

FF06 Event Setup

FF07 Calibrate

FF08 Experiment

FF09 Overlay

FF0A Stack

FF0B Average

FF0C Single Value

FF0D Minimum

FF0E Maximum

FF0F Auto

FF10 WYSIWYG

FF11 StreamSmart Setup

FF12 Plot Setup

FF13 Sensor Setup

FF14 Unit Setup

FF15 Export Setup

FF16 Event Setup

FF17 Calibrate

FF18 Experiment

FF19 Waiting to read Sensor¬

FF1A A sensor was not detected. Connect a sensor and try again.

10001 Invalid $ROMID

10002 Invalid $TITLE

10003 Invalid $MESSAGE

10004 Invalid $VISIBLE

10005 Invalid $HIDDEN

10006 Invalid $EXTPRG

10101 Invalid File

10102 Too Many

10103 Unknown Instruction

10104 Invalid Field

10105 Val betw 0-15 expected

10106 Val betw 1-16 expected

10107 Label Expected

10108 Hexa Expected

10109 Decimal Expected

1010A Can't Find

1010B Label already defined

1010C { expected

1010D } expected

1010E ( expected

1010F Forbidden

10110 Bad Expression

10111 Jump too Long

10112 Val betw 1-8 expected

10113 Insuffisant Memory

10114 Matrix Error

10115 Define Error

10116 [ or ] expected

10117 ARM register expected

10118 ARM invalid imediate

31401 No Message here