# Advanced Programming for the HP-39/40G

There is basically no information on programming these calculators availible. This is partially why there are so few programs availible. Would be programmers try and try to figure it out, but with such a lack of documentation, it is almost impossible to figure out.

This is the method you need to go about programming the calculators in sysRPL (and assembly but I won't cover that here because if you learn how to program in sysRPL for the 39/40, assembly isn't much different in terms of how you compile it into an aplet)

First thing you need to do is to learn sysRPL. This is best accomplished by learning userRPL first, even if that isn't run on the 40G. Just reading though the document, working some of the problems, etc. will help you understand how RPL works.

http://www.hpcalc.org/hp48/docs/programming/usrtutpd.zip

After you feel fairly confident with userRPL, sysRPL will not be very hard. Learning how to "think backwards" is what is the hardest thing for most people. It shouldn't be too hard after learning userRPL.

http://www.hpcalc.org/hp48/docs/programming/stut_pdf.zip

You will also need some programs for programming the calculators. Below I will give a list of all the things you'll need.

- http://www.hpcalc.org/hp39/pc/

  On this page you'll find most of the things you'll need. You need to get Hptools for your respective system, Entries 39 (I don't think there is an 40 version, but you porbably want the program you write to run on all calculators so it's best not to use an 40G only commands), HPGcomm 2.0 (or some other transferring program), and the Aplet Frame found on the same page.

- http://www.epita.fr/~avenar_j/hp/hp39/FuncAplet.zip

  Function Aplet: This aplet shows how to compile a function MYFUNCTION() so it is built into the system. Also at hpcalc.org.

- http://sources.redhat.com/cygwin/

  This is Cygnus tools for Windows. If you are running Unix or Linux, you don't need this as everything should work fine for you. As I have never used them, I can't be sure. Basically this is a Unix "emulation" that runs on windows. It allows you to compile aplets easily. Doing it manually takes almost 30 minutes. This way you just type "./create" and it is done! While this isn't completely neccecary, unless you plan on sitting around for hours compiling aplets, this is the way to go. I won't be talking about how to manually compile things, so I recommend you get this.

- http://www.epita.fr/~avenar_j/hp/39.html

  This is the page of the man in charge of making the 39/40. All the stuff is here, including a SIMULATIONS aplet that you should get. It has basically everything you can do on the 39/40, so it will be a good resource to look at later.

Ok, now that we have our tools ready, let's start programming!

# Your First Aplet!

Before we start this, it is a good idea to explain what an aplet is. An aplet is basically a complete environment in the calculator. It has all the setting and variables contained in it. An important thing to remember is that you CAN'T create new objects in an aplet. This means that if you have something that needs to be saved, you must create it first before it gets put into the calculator.

There are two major kinds of aplets. One is a Function aplet that simply defines a function, and the other is a code embedded aplet. Aplets have a main part that you can see in the APLET menu, and a library attached to them in the SHIFT-MEMORY…Library area. While libraries installed in the calculator will run fine by themselves, they can't be deleted or send to others unless they have an aplet directory attached to them.

Unzip the function aplet into a folder (I suggest something like c:\funct\ ; make it as short as possible so you have less to type later). Inside you'll see several things. Suprom39.a is the file containg the supported entries. Create is the script used by Cygnus tools (and unix/linux I think as well) to easily compile the aplet. =Common.h has sevral common definitions of things. =Entry39.h has another copy of supported entry points. =Funcdir.h has some fixed length variables in it. =Head39.h is the header file for our aplets. =Sharedvar.h has several shared fixed length variables.

Oh! What have we here! Function.s and lib.s! These are the two main files we'll be dealing with. Function.s is the source for the aplet part of the program. Lib.s is the library part. Open them up and look at them. Quite confusing, isn't it? It really isn't though. Once you've finished this most of it should start making sense to you.

Now we're going to start compiling it. You'll need sasm, rplcpl, sload, and makerom in the same folder as the function aplet. Now open Cygnus tools. First thing you'll have to do is get into the folder you have the aplet in. If you have used Unix before, this will be no problem. For those who haven't, you need to type "cd c:/funct/" or whereve you have your aplet (see why we want it to be short? =). CD means change directory. Now type "pwd" (print working directory) to show the directory. Make sure it's the right one.

Now you'll have to set the location our commands will export their results to. So type "export PATH=$PATH:$pwd". This will set the end point of our commands in the "create" script to this directory. If you run create and get back "sasm: not found" . . . all the way to the end, it means that the export isn't in the right directory.

Ok, now it is time to finally start compiling the aplet. Type ./create (./means execute a command). It will run for a little bit and then say Library built. Now open your communications program and send the aplet to the calculator. Once it is installed, go to the home screen and type MYFUNCTION. Push enter. It will say "invalid syntax". Now put the HELPWITH in front of MYFUNCTION. It pops up a box that says <xpr>. This means your function requires one argument. Now put MYFUNCTION(2). It returns 4. Now try MYFUNCTION(4). It returns 8. Any guesses on what this function is doing? =) Now try MYFUNCTION( {1,2,3}). It reutrns "Bad Argument Type". This is because it is only set up to allow real numbers as arguments.

Now go to the folder and open lib.s. On the first page, you see xNAME MYFUNCTION. This is where you put the name of the function. Right below it you see

```
::
  CK1&Dispatch
  ONE :: %2 %* ;
;
```

If you have learned sysRPL (which you SHOULD HAVE BY NOW, at least the basics anyway=), you'll understand this. Basically it means:

1. Check for one argument, get the type and run different programs depending on what it is
2. If it is a real (type ONE), then run the next program.

3. Put the real number 2 on the stack and multiply them together

Simple huh? Now what is that crap above it?

```
ASSEMBLE
        CON(3)  0
RPL
```

This is what defines a function. ASSEMBLE means switch to assembly mode, RPL means return to RPL mode. For a command, you'd replace CON(3) 0 with CON(1) 8. The difference is that functions can be used inside a symbolic object, commands can't. Here is another breakdown for you to see easier

```
ASSEMBLE
        CON(3) 0
RPL
xNAME thats_a_function

ASSEMBLE
        CON(1) 8
RPL
xNAME thats_a_command
```

See?

Now lets go edit the function. Instead of doing :: %2 %* ; , lets add a %1+ after the %*. Add another semi-colon (;) after your %1+ as well. Now you have:

```
::
 CK1&Dispatch
 ONE :: %2 %* %1+ ; ;
;
```

Save it making sure you save it as lib.s. Now return to Cygnus tools and type ./create. This time you get back "unmatched control word ;" and some error messages. This means you have an extra ; in your code somewhere. That's because we added an extra one. We know where that error is.

Now we'll check to see if there are any more errors. Go look at a file you've created while compiling called lib1540l.txt. Open it. Inside you'll see some info that looks like this:

```
Unresolved References
%1+                    ( 1)
```

That means the compiler didn't find that word, %1+, in the supported entry file. The number in the parenthesis means it found one of them. So now go fix the incorrect references back to the original.

## Renaming and changing the library number

Now it is important to change the library number of your library. If everyone used the same library number, they would conflict and only one of the programs with that library number could be installed at once.

You'll see this at the top of the function source file

```
ASSEMBLE
        Dir_Head 16,Function Example,604
RPL
```

We need to change the name. This is the part that is shown in the aplet screen. Let's rename it to My Aplet. Change the "Function Example" to "My Aplet". Now count the number of letters and spaces between the commas. In this case it is 8. Change the "Dir Head 16" to "Dir Head 8". Now comes the harder part. The 604 is the numer of the library. You probably noticed when compiling the library, that it came out as 1540. That's because the 604 is in hexadecimal (base 16). So to create a library with a number like 459, we'd change this to 1CB.

Now you must go replace every 604 with a 1CB. Open the library source file. You'll find two 604's, one at the very beginning, and one at the end. Change them. While you're here, change the TITLE in the beginning to something else (right now it is "Little Function example"). Let's replace it with "My Aplet".

Now open the create file. The first line you'll see says x="FUNCTION". Change this to something else 8 characters in length. This is the name of the directory file you create. Let's change it to "myaplet0". This will create a file called "myaplet0.000" when you're all done. Now go down to the middle. You'll see a lot of "echo" commands. One in the middle is called "echo NAME Funct". Change the Funct to something else. This is the name of the library as you can see in the MEMORY viewer on the calc in the Libs section. Let's change it to "My Library". Length doesn't really matter, but the shorter it is the less memory the name takes up. Now looking down a little further, you'll see a line "echo CK LIB604 SYSEND604". You know what to do. Replace the 604's with 1CB. A little further down is "echo HP39AscA B 12 "$y"8 "$x"H 12 LIB1540L.0008 LIB1540L". Replace the 1540 with another 4 characters, in this case let's use 0459.

Now your aplet is customized with a different name and library number. This will prevent it from conflicting with other people's aplets.

## Hash Table and Saving variables

One of the more confusing parts about aplet programming is the directory. This is made up of a hash table that allows access to all the variables. In the function file, you'll see:

```
      Vfield  L34,IntTable
      CON(5)      =DOHSTR
      REL(5)  endLink
      Link  _Main
      Link  _Entry
      Link  _Exit
      Link  _Action
      Link  _Reset
*>>>  Add links to more routines here, they must
*     be paired w/ the defines just below

endLink
RPL

DEFINE Main       INT_00
DEFINE Entry            INT_01
DEFINE Exit       INT_02
DEFINE Action           INT_03
DEFINE Reset            INT_04
```

These "links" link the program to a certain position in the table. When running internally, instead of calling Main, it calls "the thing at the first spot in the table". This makes it quicker, but it also makes it so nothing can be created or eliminated. This is explained by JYA (the guy who led the team that designed your calculator) at the link below.

http://groups.google.com/groups?q=aplet+programming+39&hl=en&group=comp.sys.hp48&rnum=1&selm=9m2fob%24quu%241%40web1.cup.hp.com

### Other Noteworthy things

The 39/40 can only return one argument to the history. This means commands can only end up with ONE result. If you return more, a lot of strange character appear on the history and it messes the calculator up. If you'd like to return more, package the results in a list or something like that.

To use many commands, you must define them as EXTERNAL commands at the beginning of the source file. I'd put them right after the INCLUDE section. Things that must be defined as EXTERNAL commands include user commands (those in the Suprom39 file that start with x), and any command that has a tilde (~) in front of it. Most, if not all of these should have a seven length address, such as #003B0F1, instead of the usual 5 length one.

http://groups.google.com/groups?q=aplet+programming+39&hl=en&btnG=Google+Search&meta=group%3Dcomp.sys.hp48

There is still much to be added to this document, but it is a start (and I ran out of time), should you choose to add more information, feel free to do so. Just acknowledge me as the author of the stuff at the beginning.

For the best information on advanced programming, read the source files for JYA's simulations aplet and the empty sysRPL aplet. Good luck!

## How do I contact you?

Well it will be pretty hard until December of 2003. I will be in Honduras until then serving a religious mission for my church (www.lds.org), and will not be doing anything on the computer until I get back. So if you send an e-mail to me at timwessman@yahoo.com, I won't be able to answer it until December 2003, and probably won't even get it because my mailbox will be full.

If you'd like to write me a snail mail letter however (talking about things other then calculators), you can send an email with the subject of "Tim's mail address in honduras" or something like that. My parents will check occasionally for these types of messages. Then they'll send it to you.