



i n v e n t

Programming the HP series

An extract from a larger text...

by Colin Croft
www.hphomeview.com

Note: This text is extracted from the book "Mastering the hp 39g+: A guide for students and teachers", which can be found on the Help page at www.hphomeview.com
Copyright resides with the author and with Hewlett Packard.

PROGRAMMING THE HP 39G+

The design process

An overview

Although you can choose to simply create programs which are self sufficient the whole point of working on the HP is to use aplets. Hence this chapter will concentrate on the process of creating aplets with enhanced powers provided by attached 'helper' programs.

The key to the entire process of creating completely new aplets is the **IEWS** menu and its controlling command function **SETIEWS**. This function allows you to override the normal behavior of an aplet and superimpose new properties by linking in a set of programs written by you.

It is mildly deceptive to call these aplets "new", as they derive from one of the standard ones, but the modification of the **IEWS** menu means that their final appearance and behavior can be very different to the aplet they derive from.

Essentially the process involves the following stages...

- Choose the parent aplet;
- Analyze the expected behavior and design the **IEWS** menu;
- Write the 'helper' programs and attach them to the aplet using the **SETIEWS** function;
- Add supporting documentation.

Choosing the parent aplet

The first stage in the creation process is to decide which of the standard aplets you wish to make the "parent" of your new child aplet. For some aplets this may not matter, but for others this can be a very important choice. All the abilities of the parent are inherited by the child so the parent choice is crucial if your aplet requires particular abilities. The most commonly used parent aplets are the Function and Statistics aplets, whereas the Quadratic and Trig Explorers would probably not make good parent aplets, since they are specialized teaching aplets without the flexibility of the others.

If your new applet is going to be concerned with analyzing data then your best choice for a parent would probably be the Statistics applet. On the other hand if you were planning to write an applet to teach the behavior of graphs then the Function or Parametric applets would obviously be best. All the tools of the parent are available to the child, so consider carefully what tools you require.



Calculator Tip

When designing applets you should consider using the ADK as it makes the process *far* easier. To use the ADK you must have the Connectivity Kit and for models before the HP this means buying a cable. We will begin by assuming that you have only the calculator and create our first two applets entirely on the HP. We will then look at two more examples using the ADK.

Naming conventions

The process starts by making a copy of the parent applet and giving it whatever name you want to use for your new applet. This copy will form the core of your new applet. Decide also what prefix to use for the programs you will associate with your new applet. The prefix needs to be recognizably linked to the name of the applet, so that the user can know which programs to delete when they want to clear the programs out after deleting the applet from the **APLET** view after use.

For example, an applet called “Linear Equations” might have a list of programs:

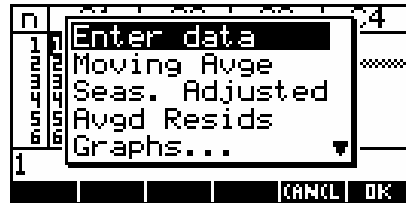
.LINEQ.SV	.LINEQ.S
.LINEQ.EN	.LINEQ.DIS

The next stage is to plan your **VIEWS** menu. The **VIEWS** menu is the controller of your applet. It pops up when the user presses the **VIEWS** key or at a programming command, and offers a choice of options to the user. Most of the options in your **VIEWS** menu will be triggers for ‘helper’ programs you will write, and when the user chooses an option and presses **ENTER**, the appropriate ‘helper’ program will be run by the HP. When the ‘helper’ program terminates the calculator drops into whatever view you as the designer choose. For example, a ‘helper’ program might set up axes based on the data entered and then drop the user into the **PLOT** view.

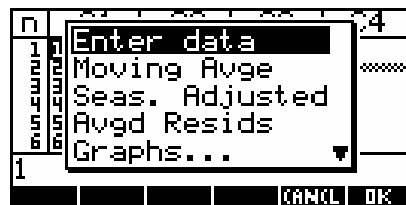
Planning the VIEWS menu

It is very important to the usefulness of your applet that you carefully plan the **VIEWS** menu to be clear, concise and user-friendly. It is possible to have sub-menus in the **VIEWS** menu by having your option call a program which then pops up another menu of options. This is usually denoted by an ellipsis (...) following the **VIEWS** option, such as the one below.

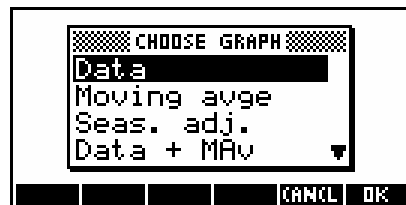
An example of the **VIEWS** menu from an applet is shown right. The applet is called "Time Series" and is designed to analyze time series data.



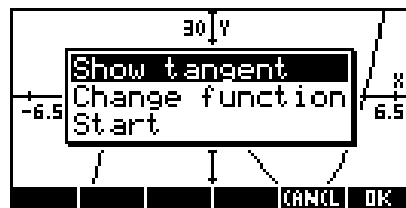
The parent applet for "Time Series" was the Statistics applet. This parent was chosen because of the need for the statistical tools it contains. For this particular applet most of the choices on the **VIEWS** menu trigger a 'helper' program to analyze the data in some way and then drop the user back into the **NUM** view showing the result. Some of the choices drop back into the **PLOT** view to see the data displayed.



Sometimes choices trigger further menus. For example the last option of 'Graphs...' runs a program which pops up another menu, shown right. The reason for this method is generally simply to avoid overcrowding the main menu.



Another example of an applet is shown right. It is called "Tangent Lines" and it draws a tangent line onto a graph and then lets you move it around, displaying the gradient as it does so. This applet has the Function applet as its parent and displays the **VIEWS** menu shown right. In this case the menu has far fewer options.



The **SETVIEWS** programming command which creates the **VIEWS** menu follows a repetitive pattern of listing a menu option, followed by the name of the program the calculator should run if the user chooses that option, followed by a code number which tells the calculator in which view to leave the user once the program finishes.

You should therefore also think about what you want the user to be looking at once the program they have triggered stops running. Do you want them to be looking at the **PLOT** view - perhaps the option they chose was to draw a graph, with the program being there to set appropriate axes; or the **NUM** view - perhaps we are analysing data - or should they be looking at the **VIEWS** menu again so that they can immediately make another choice?

The syntax for **SETVIEWS** is as follows...

```
SETVIEWS "Menu line1"; "Program name"; View_No;  
          "Menu line2"; "Program name"; View_No;  
          "Menu line3"; "Program name"; View_No: (colon on final entry)
```

where View_No is:

- | | |
|----------------------|---|
| 0. Home view | 11. List Catalog |
| 1. Plot view | 12. Matrix Catalog |
| 2. Symbolic view | 13. Notepad Catalog |
| 3. Numeric view | 14. Program Catalog |
| 4. Plot Setup | 15. Views menu item 1 (Plot-Detail in Func.) |
| 5. Symbolic Setup | 16. Views menu item 2 (Plot-Table in Func.) |
| 6. Numeric Setup | 17. Views menu item 3 (Overlay Plot in Func.) |
| 7. Views menu | 18. Views menu item 4 (Auto Scale in Func.) |
| 8. Aplet Note view | 19. Views menu item 5 (Decimal in Func.) |
| 9. Aplet Sketch view | 20. Views menu item 6 (Integer in Func.) |
| 10. Aplet Catalog | 21. Views menu item 7 (Trig in Func.) etc. |

The syntax for **SETVIEWS** allows any number of these triples.

The convention for the **SETVIEWS** command is to place it in a program with a name of **.NAME.SV** where **NAME** is whatever name you chose at design stage. When you run this program it severs the aplet's link to the normal **VIEWS** menu inherited from its parent and replaces it with the new options.



Calculator Tip

If an aplet is created using the ADK then it may not have this **.NAME.SV** program. The ADK creates the **VIEWS** menu in a different way that doesn't require it.

The linking process performed by the **SETVIEWS** command (or by the ADK39) is also important in that it tells the calculator which programs are to be transmitted with the applet when it is copied via cable or infra-red link. Only those programs named in the **SETVIEWS** command (or linked by the ADK39) will be transmitted.

In addition to the lines which form the menu for your applet, there are some special entries which are treated differently.

- i. If you include entries called "Start" or "Reset", then the 'helper' programs associated with those entries will be run when the user presses **START** or **RESET** in the **APLET** view.
- ii. If you include a menu line entry which consists of a single space character in double quotes, then the entry will not appear in the **VIEW**s menu, but the program named in the line *will* be transmitted with the applet. This can be handy if you have a program which is a subroutine. ie one which is not directly called in the menu but which is called by another program which is in the menu.
Another example of this is the **.NAME.SV** program itself. It needs to be included in the list in this fashion, since we don't want it to appear on the **VIEW**s menu but it is usually kept and transmitted with the applet. Strictly this is not necessary since, once it has done its job, it would normally never need to be run again.
- iii. If you include an entry which consists of empty double quotes, then you can access the commands which appear on the parent applet's normal **VIEW**s menu which has been replaced by yours. The standard menu options of *Auto Scale*, *Plot-Detail* etc. can be included in this way. View numbers 15 onwards are reserved for this purpose. For example, if your parent applet was the Statistics applet in **EWAR** mode and you wanted to include its *Auto Scale* command then you would use a view number of 18 since *Auto Scale* is entry number 4 on the normal **VIEW**s menu for the Statistics applet in **EWAR** mode. You need to be quite careful when using this option since the commands like *Auto Scale* appear in different positions for different parents.

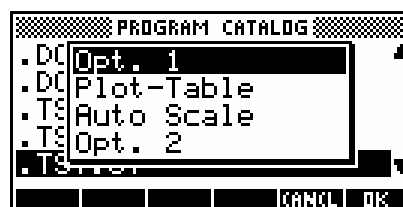
Shown below is a **SETVIEWS** program which illustrates this...

```

.TST.SV PROGRAM
SETVIEWS
"Opt. 1";".TST.A";0;
"";";16;
"";";18;
"Opt. 2";".TST.B";1;
STOP|SPACE| | |Am2|BKSP

```

producing a menu of...



It is a very good idea to include a "Start" entry, since it will be automatically run when the user presses **START** and it thus allows you to enter pre-set values in variables, or to pre-set axes, so that the aplet runs smoothly. Additionally, if you terminate the "Start" entry with a view number of 7 then as soon as the user runs the aplet the **VIEWS** menu will be displayed (it is view number 7). This makes the aplet more friendly since the controlling menu is the first thing the user sees. Some aplets tend to opt for first displaying the Note view because they include instructions there. I usually opt for the **VIEWS** menu and include instructions in a separate file.

Example aplet #1

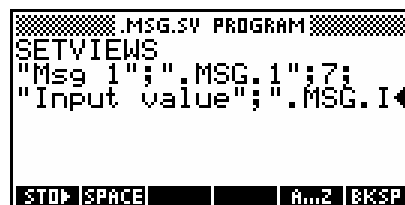
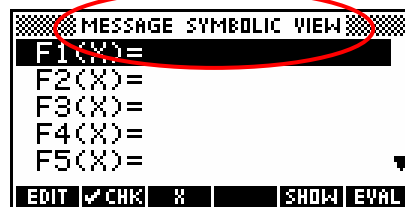
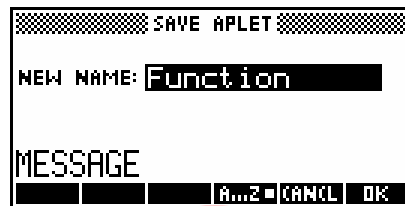
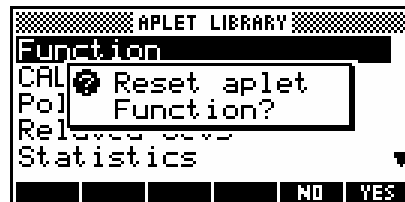
This example will use the **SETVIEWS** command to design a very simple (and totally useless) aplet, which will illustrate a few of the concepts useful in programming the HP. We'll call it the 'Message' aplet and create it as a descendant of the Function aplet.

Change into the **APLET** view, move the highlight to the Function aplet and **RESET** it. Now save it under the new name of 'Message' and then **START** this new aplet.

You will find that you are looking at the normal **SYMB** view but for the Message aplet instead of the Function aplet.

Press **SHIFT PROGRAM** to view the **Program Catalog**. Press **NEW** to create a new program and call it **.MSG.SV** (see right, with part of the new program typed in)

Into this empty program, type the following code, obtaining the quotes from the **CHARS** view. When you finish typing, just press **SHIFT PROGRAM** again to exit back to the **Program Catalog**. There is no need to save as this is done continuously as you type.



Spend a moment to go through the code and ensure that you are clear in your own mind the menu it will create, the programs it will run, and the views it will enter after the running of each program. You will be told at a later stage in this example when to run this program and create the menu.

We'll now create the associated 'helper' programs (shown below). Their names/titles are supplied above the code for each one.

.MSG.1	.MSG.IN	.MSG.2
<pre> .MSG.2 PROGRAM ERASE: DISP 4;"You entered "N: DISP 5;" when prompted.": FREEZE: STD SPACE A...Z BKSP </pre> <p>ERASE clears the screen, ready to DISP a message on lines 4 and 5 of the screen. The calculator then FREEZE's waiting for a key to be pressed.</p>	<pre> .MSG.1 PROGRAM MSGBOX "Hello world! 3+4 = "3+4: STD SPACE A...Z BKSP </pre> <p>The MSGBOX command is used to display the traditional first message for programmers learning a new language!</p>	<pre> .MSG.IN PROGRAM INPUT N;"MY TITLE";"Please enter N.:"; "Do as you're told.":20: MSGBOX "You entered " N" when prompted.": STD SPACE A...Z BKSP </pre> <p>The INPUT command asks the user for info, displaying a title, prompt and tip and having a default value of 20.</p>
.MSG.FN	.MSG.S	
<pre> .MSG.FN PROGRAM ERASE: "((X+2)^3+4)/(X-2)">F1(X): ->GROB G1;F1(QUOTE(X));0: ->DISPLAY G1: FREEZE: ERASE: ->GROB G1;F1(QUOTE(X));1: ->DISPLAY G1: FREEZE: ERASE: ->GROB G1;F1(QUOTE(X));2: ->DISPLAY G1: FREEZE: ERASE: ->GROB G1;F1(QUOTE(X));3: ->DISPLAY G1: FREEZE: ERASE: LINE XminiYminiXmax;Ymax: BOX 3:3;-2;-2: FREEZE: STD SPACE ▲PAGE A...Z BKSP </pre>	<pre> .MSG.S PROGRAM MSGBOX "Aplet starting now": STD SPACE A...Z BKSP </pre> <p>The command \rightarrowGROB in the program left, stands for "Graphic Object" and creates a GROB from the F1(X) expression stored in the SYMB view, storing it in the graphic memory G1, using the font specified (0, 1, 2 or 3). The reason for doing it this way is to use proper mathematical layout like SHOW does. The \rightarrowDISPLAY command then shows it on screen.</p>	<pre> .MSG.SV PROGRAM SETVIEWS "Message 1"; ".MSG.1";7; "Input value"; ".MSG.IN";7; "Message 2"; ".MSG.2";0; "Show func."; ".MSG.FN";7; "Start";:".MSG.S";7; "Quit";:".MSG.SV";0; " ";:".MSG.SV";0: STD SPACE ▲PAGE A...Z BKSP </pre> <p>The SETVIEWS command is discussed in detail on the previous pages.</p>

Having created all of the programs that make up the aplet 'Message', we can now run the program .MSG.SV, severing the aplet's link to its current VIEWS menu which was inherited from its parent the Function aplet, and substituting this new menu. *Before you do this*, check that you are still in the correct aplet. Press the SYMB key and check that the title at the top still says "MESSAGE SYMBOLIC VIEW". If it doesn't show this, then **START** the aplet again to ensure that it is the active one and so the one whose VIEWS menu will be changed.

Swap back to the **Program Catalog**, position the highlight on the program **.MSG.SV** and **RUN** the program. Apart from the screen going blank for a moment nothing will appear to happen, but in fact the link to the normal **IEWS** menu which 'Message' inherited from its parent applet Function has been severed and a link to the new menu you built in **.MSG.SV** has been substituted. Press **IEWS** to check.

Providing that you have done everything correctly, this is now the end of the process - the applet is now ready to be run. In the **APLET** view, make sure the highlight is still on the applet and press **START** or **ENTER** to run it. If you get an error message at any time then you may have to **CANCEL** and **EDIT** the program.

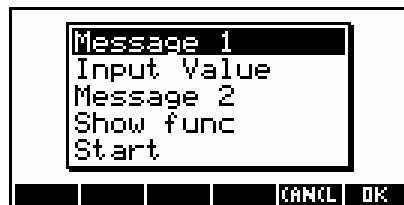
When you do this, the applet will run the program **.MSG.S** which will display a **MSGBOX**.



The line in the **SETIEWS** command controlling this was:

```
"start";".MSG.S";7;
```

Since the triple ends with a view number of 7, this means that after the program terminates (when you press **OK**), the **IEWS** menu will display again.



If you choose the option 'Message 1', then this will cause the program **.MSG.1** to be run, displaying the screen on the right. This line in the **SETIEWS** command also terminated with a view number of 7 so when you press **OK** the **IEWS** menu will display again.



The program line for this was:

```
MSGBOX "Hello world! 3+4 = "3+4:
```

Items in quotes are displayed as they appear, while expressions outside them are evaluated before being displayed. Expressions can include variables and calls to functions.

The next option in the menu is 'Input value'. Choosing this option will create an input screen. The statement controlling this was:



```
INPUT N;"MY TITLE";"Please enter N..";"Do as you're told.";20:
```

Examine the snapshot on the right and notice the connection between the various parts of the **INPUT** statement and their effect. Note the suggested value of 20, and note also that the prompt of "Please enter N.." was too long to be displayed. See the **PROMPT** command for an alternative that is simpler but less flexible.



When you enter a number into the input screen and press **ENTER**, the next line in **.MSG.IN** will display this value in a **MSGBOX**. When you then press **OK**, the view number of 7 specified in the relevant line of **.MSG.SV** will cause the **IEWS** menu to be displayed again.

Notice that the input window is still displaying in the background. To stop this happening, you could have included in **.MSG.IN** a line of **ERASE: ,** which is a command to erase the display screen. Try editing the program, inserting this line before the **MSGBOX** line, and running it again.

The option of 'Message 2' displays the same message as we saw before, but presented in a different way. The **DISP** command divides the display screen up into 7 lines (1 - 7) on which you can display data.



For example, suppose memory A contained 3.56, then the command:

```
DISP 3;"The value of A is: "A:
```

would display the message **The value of A is: 3.56** on line 3 of the display screen.

Notice also that this time when you press **ENTER**, you end up in the **HOME** view rather than in the **IEWS** menu again. This is not an error. If you look at the line in **.MSG.SV** controlling this option of the menu you will see that its post execution view number was 0 (**HOME**) rather than 7 (**IEWS** menu) like most of the others. To see the **IEWS** menu again, press **IEWS**.

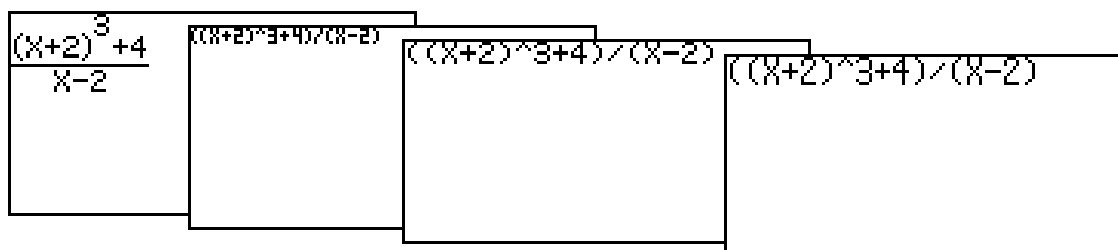
The final option is 'Show function'. The program this runs is a little more complex than the ones shown so far and illustrates a useful technique.

The line: `'((X+2)^3+4)/(X-2)' ►F1(X) :`

stores the expression $\frac{(x+2)^3+4}{(x-2)}$ into the function F1(X).

Notice the way the function is in single quotes so that the algebraic expression itself is used rather than its value when evaluated using the current contents of memory X. If you wanted to graph this function by setting the post execution view number to 1 (the **PLOT** view), then you would need to include the command `CHECK 1:` in order to **CHK** it or it would not graph. You may wish to edit the `.MSG.SV` and `.MSG.FN` program to try this.

The next lines display the expression using the four options available.



The line: `→GROB G1;F1(QUOTE(X));0:` converts the expression F1(X) into a Graphic Object (GROB). The number at the end, which changes with each repetition, controls the font used to display the function.

The line: `→DISPLAY G1:` displays this GROB on the screen, and the `FREEZE` command freezes the display until a key is pressed.

Finally the **LINE** and **BOX** commands are used to draw an oblique line across the screen and a box near the center. Notice the use of Xmin, Xmax, Ymin and Ymax in the **LINE** command. This means that the line would appear the same even if the screen were to be re-sized in the **PLOT SETUP** view. The **FREEZE** command is needed to ensure that your screen is visible to the user and not immediately replaced by the next view.

Note that **SETVIEWS** has a "Start" option and also a final option consisting of a single space in quotes which is simply to link in the `.MSG.SV` program so that it is transmitted with the applet. It does not appear on the menu.

Example applet #2

In the text from which these pages come, there is an explanation of how to create a copy of the Parametric applet to explore geometric transformations using matrices. We will now look at using programming to enhance this applet by automating the process. You don't need to have read the main text to follow this but it would certainly help.

Start by highlighting the Parametric applet and pressing **RESET**. Now **SAVE** the applet under the new name 'Transformer'. Press **SHIFT NOTE** (not **NOTEPAD**) and enter some explanatory text into the applet's Note view. You can use the text shown right.

```

TRANSFORMER NOTE
This applet will let
you investigate geo-
metric trans-
formations using a
2x2 matrix.
Press VIEWS.
SPACE | A...2 | BKSP
    
```

The next step is to create the 'helper' programs for the applet, including the one containing the **SETVIEWS** command used to create a new **VIEWS** menu for the applet. These programs are shown on the next page. When you have typed them all in then **RUN** the program **.TRANSF.SV** to create the **VIEWS** menu.

```

Change matrix
Plot transf.
Change shape
Change axes
Start
CANCL | OK
    
```

Programs for the applet 'Transformer' are given below.

.TRANSF.SV	.TRANSF.S	.TRANSF.PLOT
<pre> .TRANSF.SV PROGRAM SETVIEWS "Change matrix"; ". TRANSF.MAT";7; "Plot transf."; ". TRANSF.PLOT";1; "Change shape"; ". TRANSF.SHAPE";7; "Change axes";";4; "Start"; ". TRANSF.S";7; " ";". TRANSF.SV";0; STO SPACE PAGE A...2 BKSP </pre>	<pre> .TRANSF.S PROGRAM -10Xmin:10Xmax: -6Ymin:6Ymax: 1Tmin:4Tmax: 1Tstep:1Connect: 1Grid:0Simult: 'M2(1,T)'X1(T): 'M2(2,T)'Y1(T): CHECK 1: 'M3(1,T)'X2(T): 'M3(2,T)'Y2(T): CHECK 2: [[1,0],[0,-1]]M1: [[1,2,1,1], [1,1,3,1]]M2: M1*M2M3: STO SPACE PAGE A...2 BKSP </pre>	<pre> .TRANSF.PLOT PROGRAM: XminZ:3Xmin:ZXmin: M1*M2M3: STO SPACE A...2 BKSP </pre>
<p>This program sets up the VIEWS menu to call each of the other programs. It need only be run once at the creation of the applet, but is attached via the final line so that it will be sent with all the others if the applet is transmitted. The new user does not have to re-run it: it will never normally be run again.</p>	<p>This program sets up the required axes using variables from the PLOT SETUP view. It then loads the equations and ensures they are CHKed and ready for use. Finally it loads the initial values into the matrices.</p>	<p>This program changes the value of Xmin and then changes it back. In the original version the user had to press PLOT to force a re-draw. This technique fools the hp 39g+ into thinking that the PLOT view has changed and therefore forces a re-draw without the need to press a key. It also re-multiplies the matrices in case the user has changed one by hand instead of going through the VIEWS menu.</p>

.TRANSF.SHAPE		.TRANSF.MAT
<pre> .TRANSF.SHAPE PROGRAM 1▶C: CHOOSE C;"SHAPE"; "Right triangle"; "Pointer"; "User defined"; IF C==1 THEN [[1,2,1,1], [1,1,3,1]]▶M2: ELSE IF C==2 THEN [[1,2,2,1.5,1,1], [1,1,3,4,3,1]]▶M2: ELSE DO EDITMAT M2: SIZE(M2)▶L0: IF L0(1)≠2 THEN MSGBOX"Matrix must be 2xN": END: UNTIL L0(1)==2 END: END: SIZE(M2)▶L0: L0(2)▶Tmax: M1*M2▶M3: STO▶[SPACE]▲PAGE A...Z BKSP </pre>	<p>This program (left) uses the CHOOSE command to offer a list of options. Note the need to pre-load a value into C. This value determines which option is highlighted when the menu appears. If a list has only three options but the highlight is set to some other value than those three then it can crash the program. Options 1 and 2 load preset matrices while option 3 allows the user to edit their own. Note the check to ensure the matrix they entered has a valid size. The number of columns is then extracted and used to reset the value of Tmax. The new image matrix is also recalculated.</p> <p>The indenting used is not required and is there simply to make the program easier to read.</p>	<pre> .TRANSF.MAT PROGRAM MSGBOX"Enter the 2x2 matrix. Press OK when finished.": DO EDITMAT M1: IF SIZE(M1)≠(2,2) THEN MSGBOX"Matrix must be 2x2 only.": END: UNTIL SIZE(M1)==(2,2) END: M1*M2▶M3: STO▶[SPACE]▲PAGE A...Z BKSP </pre> <p>This program puts up a message instructing the user and then allows them to edit the transformation matrix in M1. The size of the matrix is checked to ensure it is 2x2, with the DO...UNTIL loop ensuring that the user cannot exit without a valid matrix entered.</p>

Assuming that you have **RUN** the **.TRANSF.SV** program to create the new altered **VIEWS** menu then you can now test the aplet. Its operation should be familiar to you if you have read the original explanation in the larger text.

In the next example we will use the Aplet Development Kit (ADK39) to re-create the same 'Transformer' aplet used in example 2. This will allow us to concentrate on how to use the ADK39 rather than the aplet. The ADK39 runs only on Windows computers and was originally written for Windows 3.1. Because of this it does not understand long filenames or the Desktop and this makes it difficult to use at times. It may be that when you read this text new software will have been released by HP to supercede the ADK39. The behavior of any successor is likely to be quite similar to that shown below since the basic design process is fixed by the calculator.

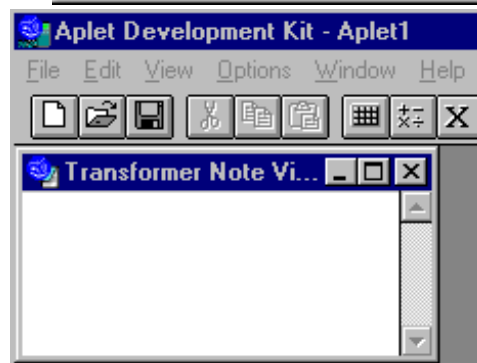
There are two versions of the ADK - one for the hp 38g and one for the later models. Aplets created by one version are not compatible with the other version's calculator model(s). Look for the ADK on the Utilities page of *The HP HOME view* (at <http://www.hphomeview.com>). Images shown here are from the older version of the ADK but the only real difference is that the title bar is red in the newer version

!Example aplet #3

Run the Aplet Development Kit and use the *File - New* command to see the box shown right. Enter 'Transformer' and nominate the parent aplet to be the Parametric aplet in the box provided.

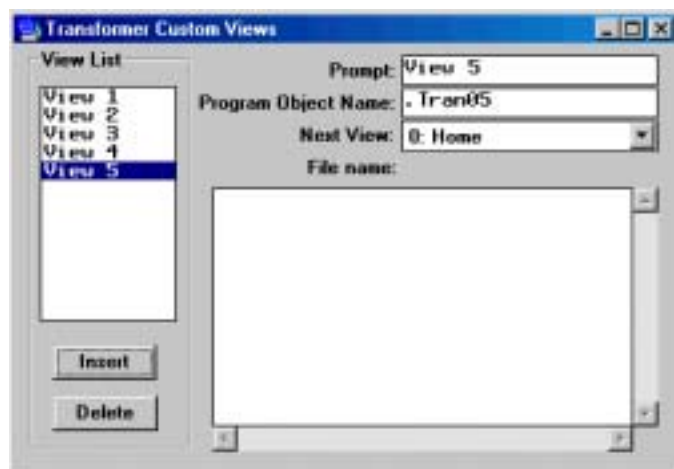


When you press the 'OK' button the aplet will be created and its Note view will be displayed. Enter the text below as a hint to the user on how to use the aplet in case they don't have the documentation.

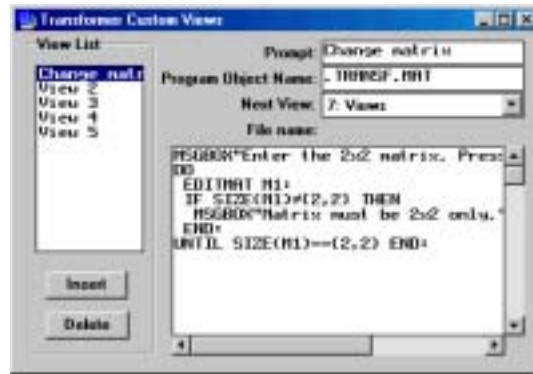




"This aplet will let you investigate geometric transformations using a 2x2 matrix. Press VIEWS to see the menu."

The next stage is to create the **VIEWS** menu. From the *View* menu, select *Special views...* and you will see the **VIEWS** menu creation screen. Press the *Insert* button five times to create the five entries we require for our menu. You can also create them one by one as required.



Click on “View 1” in the View List window. Change the prompt to “Change matrix”, the Object name to “.TRANSF.MAT” and the Next View to “7: Views”.

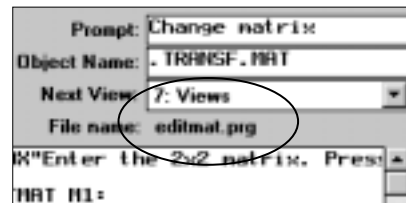


In the main window, enter the code for this program (see Example 2). Special characters such as  can be obtained from the  button.

We now must save the code. Click on the “View 2” entry in the View List window and the ADK will ask if you wish to save the code you entered for the first view. Tell it ‘Yes’ and a save dialog box will appear. At this point I usually realize I have forgotten to create a directory to hold my aplet and you may have too. If so, click on ‘Cancel’, and create an empty folder to hold the aplet. Remember that the ADK is a very old program and does not recognize long filenames. It only accepts names with up to eight characters and no spaces. This applies to directories also, so if you use ‘Transformer’ for your directory then you may find it appears in the ADK’s directory view as TRANSF~1, which is the Windows ‘short’ version.

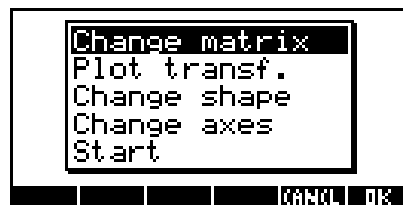
Save the program under any name. I used EDITMAT.PRG. Make sure you are in the directory you created before you click on ‘Save’.

If you click again on the first view in the View List window you will find that it now records the filename you used.

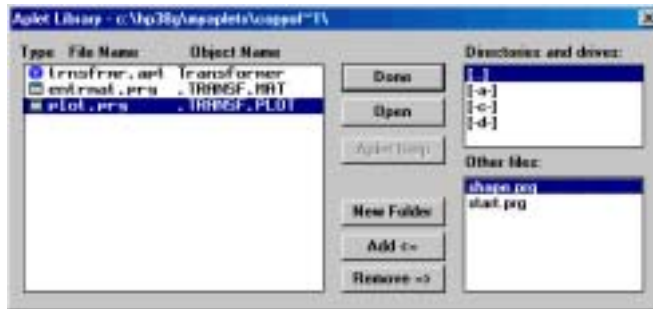


At this point you should also save the aplet itself, perhaps as TRANSFRM.APT.

Using similar methods, enter the code for the other programs, with two exceptions. Firstly, the *Change Axes* option has no program attached and you should leave the Object Name blank for that entry, simply specifying the Next View entry to be the **PLOT SETUP** view. Secondly, the program we used before to contain the SETVIEWS command is not required for aplets created by the ADK. When you finish, close the views planner and use the *File* menu to save the aplet also.



The final stage is to use the ADK to create the two special files HP39DIR.CUR and HP39DIR.000.



In the *File* menu, choose *Aplet Library*. You will see a list of the programs and the aplet in the window labeled “Other files”.

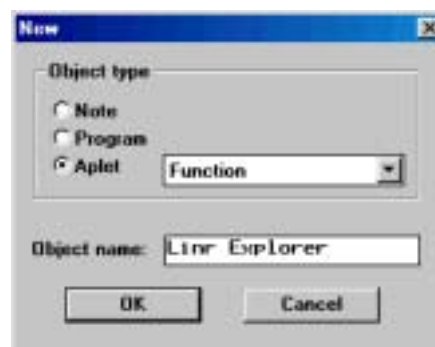
Click on each file in the right hand bottom window in turn to highlight it and then on the “Add <=>” button to add it to the library files. You can only add files which are legitimate calculator files. I usually add the aplet file first and then each of the programs in the order that they appear on the **VIEW**s menu, but this has absolutely no effect on the running of the aplet.

When you finish, click on the “Done” button and the files HP39DIR.CUR and HP39DIR.000 will be created. Exit from the ADK and the aplet is finished and ready to be transferred.

Example aplet #4

The final example is a very useful aplet called “Linr Explorer”. The name would be better as “Linear Explorer” but names of more than 14 characters will not display properly in the **APLET** view. This will be somewhat similar to the Quad and Trig Explorer aplets, except that it will explore linear equations. Its parent is the Function aplet.

Create a directory to hold the aplet and then run the ADK. Use the *File - New* command to create an aplet called “Linr Explorer” with a parent of the Function aplet.

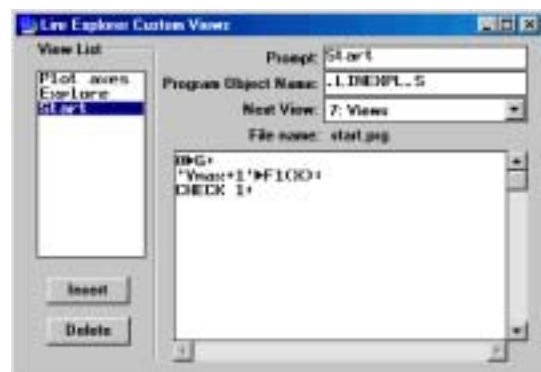
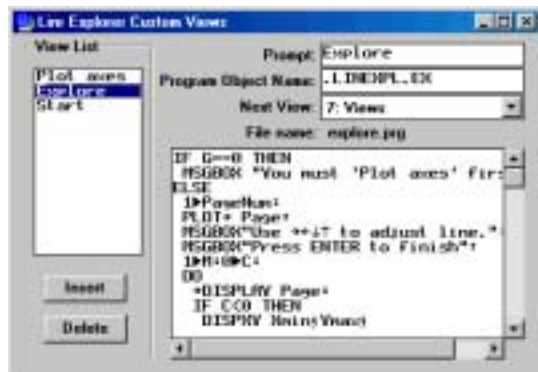
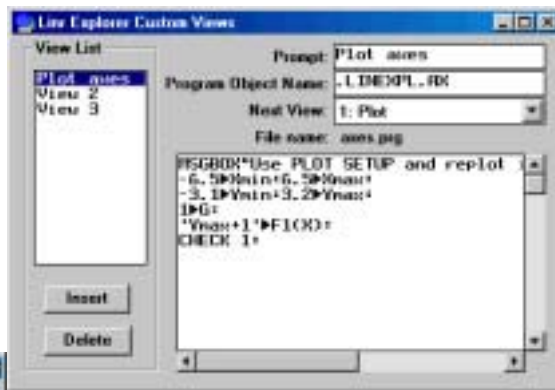


In the Note view, enter the text shown right.

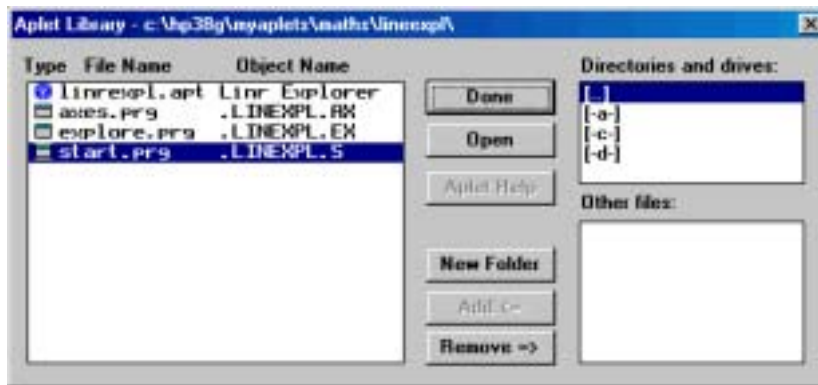
Our **VIEWS** menu will only have three entries, so use the *View - Custom views...* command to display the menu planner and press 'Insert' three times. Our **VIEWS** menu will be as shown below.



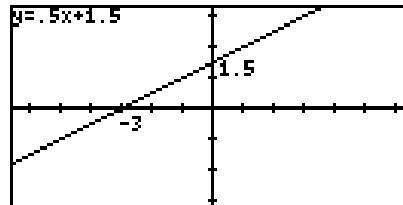
The first entry should have a Prompt of 'Plot axes', an Object Name of '.LINEXPL.AX' and a Next View of 1 (Plot view). The full code for each of the programs is given on the next page, and other settings are shown below and right.



Save the applet and use the *File - Applet library* facility to create the two special files for the directory which allow the calculator to download it. Finally, download it to the calculator and test it.



Choose the first option on the **VIEWS** menu to plot the axes and then the 'Explore' option to explore the equation of a line. On the pages following we will examine the code in detail, as it illustrates many highly important techniques.

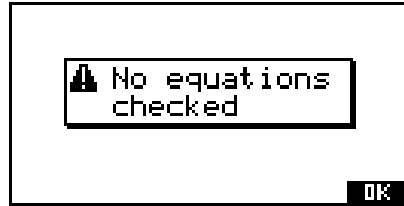


Since it is the first program run we will look first at the program .LINEXPL.S.

```
.LINEXPL.S PROGRAM
G:
'Ymax+1' F1(X):
CHECK 1:
-6.5 Xmin:6.5 Xmax:
-3.1 Ymin:3.2 Ymax:
STOP |SPACE| | A...Z |BKSP
```

The variable **G** is being used here as a flag. Before the 'Explore' option can be used we must be sure that the axes have been plotted, and this will be done by checking the value of **G**. Zero will mean 'unready' and one will mean 'ready'. When the 'Plot axes' option is run the value of **G** will be set to 1 and this will be checked before allowing the user to run the 'Explore' option. By setting it to zero in the program which is run when the user presses **START**, we ensure that the correct value is going to be in **G** initially. If this check is not done then the code used in .LINEXPL.EX will cause the program to crash, which is obviously not something we want. The axes are also set back to the default settings in case the user has changed them.

The 2nd and 3rd lines are there to insert a function. We need a function when the axes are plotted or the normal error message will be displayed (see right), which is undesirable as it confuses the user. On the other hand we need *blank* axes, so we use a function 'Ymax+1' which is guaranteed to be off-screen for the entire x axis range no matter what axes are used. Clever, eh?



The next program code we will look at belongs to the 1st option on the **VIEWS** menu of 'Plot axes'.

```
.LINUXPL.AX PROGRAM
MSGBOX"Use PLOT SETUP
and replot if
different axes are
wanted":
1▶G:
'Ymax+1'▶F1(X):
CHECK 1:
STOP SPACE ▲ PAGE | A...2 | BKSP
```

A message is first given to the user of how to proceed if they want to use different axes. The flag value of **G** is then set to 1 so that the next program can tell that the axes are ready to use. The function is also re-entered in case the user has changed the **SYMB** view. Users have a habit of changing things so try to allow for this in your programs.

The next program below illustrates a very important technique where a copy of the **PLOT** view is stored in the aplet's sketch view and then retrieved and modified using the various graphics commands. The program is broken into parts for discussion purposes.

```
.LINUXPL.EX PROGRAM
IF G==0 THEN
MSGBOX "You must'Plot axes' first":
ELSE
1▶PageNum:
PLOT▶ Page:
MSGBOX"Use →←↓↑ to adjust line.":
MSGBOX"Press ENTER to finish":
```

The reason for the "IF G==0 THEN" is to check that the blank axes have been plotted and are available for use. If not then the user receives a message to tell them what to do and the remainder of the program is bypassed. Trying to capture a **PLOT** view that doesn't exist is a major error.

Still referring to the code on the previous page, you will see that it refers to **PageNum**. The sketches in the HP's **SKETCH** view are numbered 1, 2, 3...etc. Sketch number 1 is always present but after that only sketches that have been created are available and the program will crash if you try to access one that does not exist. The aplet variable **PageNum** is the pointer to the sketch you want and the actual sketch is called **Page**. Thus the two lines after **ELSE** tell the program to store the **PLOT** view into the first page of the **SKETCH** view. The **PLOT** view *must* exist before this can be done or the program will crash. If you run the program and then later change to the **SKETCH** view you will be able to see this stored image. Finally, the user is presented with two messages which tell them what to do.

The next section begins the code which performs the work in the aplet.

```

1▶M:0▶C:
DO
  →DISPLAY Page:
  IF C<0 THEN
    DISPHY Xmin;Ymax;1;"y="M"x"C:
  ELSE
    DISPHY Xmin;Ymax;1;"y="M"x+"C:
  END:
  DISPHY .3;C;1;C:
  IF M≠0 THEN
    DISPHY -C/M;-.3;1;ROUND(-C/M,2):
  END:
  LINE Xmin:M*Xmin+C;Xmax:M*Xmax+C:

```

The first line assigns initial values to the variables **M** (gradient) and **C** (y-intercept). The **DO...UNTIL** loop which follows (partly in the next section of code) loops through the code within it until the **ENTER** key is pressed.

Within the loop, the previously stored **SKETCH** view is transferred from storage to the display using the **→DISPLAY** command. The equation of the current line is then displayed in the top left corner using the **DISPHY** command. Two versions are needed to avoid an expression like "y=2x+ -1".

The **DISPHY** command appears in the *Prompt* section of the **MATH** menu.

The **DISPXY** command allows you to place a string of text at any position on the screen using two different fonts. Until this command was added to the language (after the first HP39G) the only way to do this was to:

- save the current screen into a graphics variable.
- create a special GROB which contained the text.
- superimpose the GROB onto the stored image.
- redisplay the modified image onto the screen.

Quite apart from the fact that this was incredibly involved, it was very slow and the command to create the GROB was very limited in what it would allow.

The command has the syntax:

DISPXY <x-pos>;<y-pos>;<font#>;<object> :

Suppose $M=2$ & $C=3$. Then the command `DISPXY Xmin;Ymax;1;"y="M"x+"C:` will display the text "y=2x+3" using font #1 (small) at the top (Ymax) left (Xmin) of the screen.

The next line places a label on the y axis (offset slightly) to mark the y-intercept. A check is then done to see if an x-intercept exists and, if it does, a label is placed to mark it. Any labels off the edge of the screen will be ignored.

Finally the line itself is drawn. Even though part of the line extends off the screen there is no problem - the excess is clipped.

The next section of code below waits until the user presses a key (**GETKEY**) and stores the key's code into the variable **K**.

```
GETKEY K:
CASE
  IF(K==25.1)THEN C+.5▶C:END
  IF(K==35.1)THEN C-.5▶C:END
  IF(K==34.1)THEN M+.5▶M:END
  IF(K==36.1)THEN M-.5▶M:END
END:
UNTIL K==105.1 END:
END:
ERASE:
STOP|SPACE|▲PAGE|A...Z|BKSP
```

A **CASE** statement is then used to check for the use of the arrow keys. Notice the lack of colons (:) after each **END** in the **CASE** statement.

If the left or right arrows have been pressed (keys 34.1 or 36.1) then the line is 'twisted' by changing the value of **M**. If the up or down arrows have been pressed (keys 25.1 or 35.1) then the line is raised or lowered by changing the value of **C**. See the manual for more information on the **GETKEY** and **CASE** commands and on key values.

The final check in the line **UNTIL K==105.1 END:** is to see if the user has pressed the **ENTER** key. If so then the loop will terminate and the screen will erase. If not then the loop begins again with the new line being displayed. On termination of the program the **VIEWS** menu will display again, because we chose this when designing the applet in the ADK.

This applet illustrates most of the commonly used programming techniques. If you would like to gain experience then I suggest that you do as I did - download applets and pull them apart to see how they work.

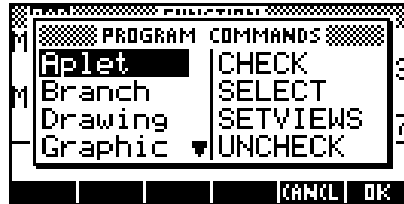
If you would like to further enhance this applet then try the following:

- change the order of the code so that the labels are drawn after the line, thus ensuring that the text is never obscured by the line.
- add a new variable **D** to allow the size of the increment to change. Set an initial value of **D** of 0.5 at the same point as the values of **M** and **C**. Then change the lines above so that **D** is added/subtracted instead of 0.5. Add two more **IFs** to the **CASE** statement so that if they press '+' the increment doubles, and if '-' then it halves. You will also need to add another message box line telling them about this. Finally, add a line to display the current increment size at the top right of the screen using the **DISPXY** command.

The explanation so far should help you in understanding the programming process on the HP. The applet structure is well designed and, if you take advantage of the **VIEWS** menu structure, offers easy creation of complex and very powerful teaching applets. Certainly what has been discussed here is enough that a programmer will be able to make a start without some of the errors that I made.

PROGRAMMING COMMANDS

All programming commands can be typed in by hand but, as with the **MATH** commands, can also be obtained from a menu. Press **SHIFT CMDS** to display this.



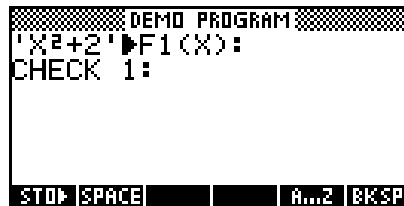
In this section I will only be covering those commands which I have used regularly and so regard as important. These may not be the same as the ones you regard as important. If so, consult the manual.

The Aplet commands

These control aspects of the aplet.

CHECK *n*, UNCHECK *n*

These commands put or remove a check next to the equation whose number is given by *n*. An interesting bug is actually quite useful: if you **UNCHECK 0** then all equations are unchecked instead of only equation 0. Unfortunately the same is not true for the **CHECK** command. As they say in the trade: "It's not a bug, it's a feature!"



SELECT <name>

This is used to set the active aplet if necessary. If the name has spaces in it then it must be enclosed in quotes. This is not usually required as the program will normally be called by the active aplet anyway. I have only used it with 'stand-alone' programs not attached to an aplet so that they can temporarily 'borrow' abilities belonging to an aplet. However, it could also be used to create an aplet that had two 'parents' if you required it to inherit abilities from both. You could then swap from one parent to the other using this command. This could be quite cumbersome but might add some powerful features.



SETVIEWS <prompt>;<program>;<view number>

This absolutely critical command is covered in great detail on page 4.

The Branch commands

IF <test> THEN <>true clause> [ELSE <>false clause>] END

Note the need for a double = sign when comparing equalities. Any number of statements can be placed in the true and false sections. Enclosing brackets are not required.

```
DEMO PROGRAM
IF A==5 THEN
  6>B:
ELSE
  7>B:
END:
STOP SPACE | A...Z BKSP
```

CASE <if clauses> ...END:

This command removes the need for nested IF commands but is only worth it if you have more than two or three nested IFs. Note that colons are not required for the ENDS which terminate the internal IF clauses.

```
DEMO PROGRAM
CASE
IF A<=0 THEN 5>B: END
IF A>0 THEN 2>B: END
END:
STOP SPACE | A...Z BKSP
```

IFFERR <statements> THEN <statements> [ELSE <statements>] END

This can be used to error trap programs where there is a possibility of something going wrong which would normally crash the program, such as evaluating a function at a point for which it is undefined. By trapping the suspect code you can supply an alternative which will perform some other action. This will tend to make your programs more user friendly and is a very good idea!

```
DEMO PROGRAM
IFFERR F1(X)>A: THEN
  MSGBOX"Undef. ":
ELSE
  ::code...
END:
STOP SPACE | A...Z BKSP
```


RUN <program name>

This command runs the program named, with execution resuming in the calling program afterwards. If a particular piece of code is used repeatedly then this can be used to reduce memory use by placing the code in a separate program and calling it from different locations.

See the SETVIEWS command for information on how to link a program to an applet when it does not appear on the primary menu. Note that if the name has spaces in it then it must be enclosed in quotes.

```
DEMO PROGRAM
RUN Small:
RUN "Input value":
STOP SPACE | A...2 BKSP
```

STOP

This command can be used to abort execution of a program. Control resumes in the HOME view.

```
DEMO PROGRAM
IF A==0 THEN
STOP:
ELSE
... Code...
END:
STOP SPACE | A...2 BKSP
```

The Drawing commands

ARC <x-center>;<y-center>;<radius>;<start angle>;<end angle>

This command draws an arc on the screen. It uses the current values in the PLOT SETUP view as the screen coordinates and the settings in the the MODES view for angle format. This command is quite slow.

```
DEMO PROGRAM
ARC 0;0;3;0;π/2:
STOP SPACE
```

BOX <x1>;<y1>;<x2>;<y2>

This draws a rectangular box on the screen using (x1,y1) and (x2,y2) as the corners. The coordinates are relative to the settings in the PLOT SETUP view.

```
DEMO PROGRAM
BOX -3;2;1;-1:
STOP SPACE
```

ERASE

This command erases the current display screen.

FREEZE

This command halts execution until the user presses any key.

LINE <x1>;<y1>;<x2>;<y2>

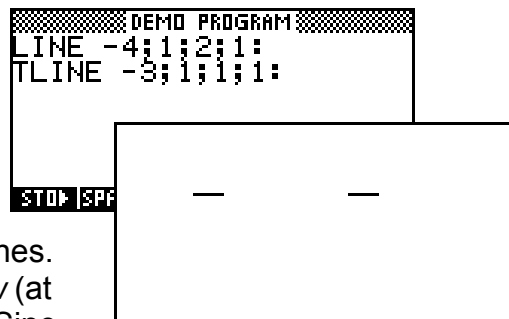
This draws a line on the screen using (x1,y1) and (x2,y2) as the ends. The coordinates are relative to the current settings in the **PLOT SETUP** view.

PIXON <x>;<y> and PIXOFF <x>;<y>

This command turns a pixel point on or off at the specified point. The coordinates are relative to the current settings in the **PLOT SETUP** view.

TLINE <x1>;<y1>;<x2>;<y2>

This command is the same as **LINE** except that the line drawn reverses the current set/unset value of all pixels. It can be used to erase previously drawn lines. One of the applets on *The HP HOME view* (at <http://www.hphomeview.com>) is called "Sine Define" and contains extensive use of this command. It is an incredibly useful command if you're doing any sort of animation of diagrams because you can draw new lines to the diagram and then erase them without disturbing the diagram underlying.



The Graphics commands

See page 26 for examples illustrating some of the graphics commands used regularly.

The Loop commands

**FOR <variable> = <start value> TO
<end value> [STEP <increment>] <statements> END**

This is a standard FOR...NEXT command. The STEP value is optional and is assumed to be 1 if not stated. Whatever you do, don't use NEXT to terminate the loop! It doesn't register as an error but all sort of strange things happen!

```
FORNEXT PROGRAM
FOR I=10 TO 50 STEP 5;
  DISP 3;I:
END:
FOR I=10 TO 40;
  DISP 3;I:
END:
STOP SPACE | A...Z | BKSP
```

DO <statements> UNTIL <test clause> END

This loop executes the statements within it until the test clause evaluates as true. It must execute at least once. The example right checks for a positive integer from the INPUT statement. To be even more user friendly you could let the user know what they had done wrong by adding another few lines of code within the DO loop of..

```
IF INT(N)≠N OR N≤0 THEN
  MSGBOX "Enter a positive integer only":
END:
```

```
DOUNTIL PROGRAM
DO
  INPUT N;"ITERATE";
  "N:";"Runs?";100:
UNTIL INT(N)==N AND
N>0 END:
STOP SPACE | A...Z | BKSP
```

WHILE <test clause> REPEAT <statements> END

This is similar to the DO...UNTIL loop except that the test clause is evaluated before starting so that the loop may not be executed at all.

BREAK

This command will exit from the current loop, resuming execution after the end of it. There is no GOTO <label> command in the language.

The Matrix commands

EDITMAT <matrix var>

This command pops up a window in which the user can edit or input a matrix with an **OK** key at the bottom. When the user presses **OK**, execution resumes after the EDITMAT statement.

REDIM <matrix var>;<size>

This command is very useful if the size of a matrix is not known in advance. The user might be prompted to input the size and then these values used to resize it. Note that the dimensions must be supplied as a list variable. The SIZE command can also be used in this context as it returns a list variable when used with a matrix.



The Print commands

These commands are supplied for use with the battery operated HP infra-red thermal printer that was designed for use with the hp 38g.

As far as I am aware you can no longer buy it so they are generally a bit useless now!

PRDISPLAY

If you place this command in a program then the current display will be sent to the infra-red printer.

PRHISTORY

This command, whether issued in the **HOME** view or in a program, will send the entire contents of the History to the infra-red printer.

PRVAR <variable>

This command, whether issued in the **HOME** view or in a program, will send the value of the variable to the infra-red printer. This can be used to capture and send images of graphs without need for programming as follows:

- set up the graph or image as required
- press **ON+PLOT** to capture the image and store it in grob **G0**
- in the **HOME** view, issue the command **PRVAR G0**.

The Prompt commands

BEEP <frequency>;<duration>

This will use the piezo crystal in the calculator to create a sound of the specified frequency for the specified duration (in seconds). The resulting frequency is not terribly accurate, varying by up to 5% from one calculator to the next and depending also on the temperature. In later models the volume of this sound was lowered because of complaints from examiners and teachers.

The frequencies of the twelve semi-tone jumps in the harmonic scale form a geometric sequence, and since the ratio from C^1 to C^2 is 2, the ratio for each semi-tone must be $^{12} \sqrt{2}$. The standard frequency used in tuning instruments is usually 440 cycle/sec for the note A. Since much of the simple music used by students is written using the scale of C, I use $440 / (^{12} \sqrt{2})^9$ to find the frequency of C as 261.6 cycles/sec.

We can use this to form a standard 'header' for any program we want to use to play music. The header shown right in the rounded box sets up the scale of C major. The code which then follows plays the first two bars of the tune "Strangers in the Night".

```
.5▶T:
261.6▶C:
12 NTHROOT 2▶R:
R²C▶D:
R²D▶E:
R²E▶F:
R²F▶G:
R²G▶A:
R²A▶B:
```

```
BEEP F; T/2:
BEEP G; T/2:
WAIT .01T:
BEEP G; T/2:
BEEP F; T/2:
BEEP G; 2.5T:
BEEP F; T/2:
BEEP G; T/2:
BEEP A; T/2:
BEEP G; T/2:
BEEP F; 1.5T:
```

In this header, the duration of a note (T) is set to 0.5 seconds. It is easy to change the tempo of the music by adjusting this. In this case you may find that the music sounds a little better with T set to 0.55 or 0.6 seconds. T is a crotchet, T/2 a quaver etc.

CHOOSE <variable>;<title>;<menu option1>;...

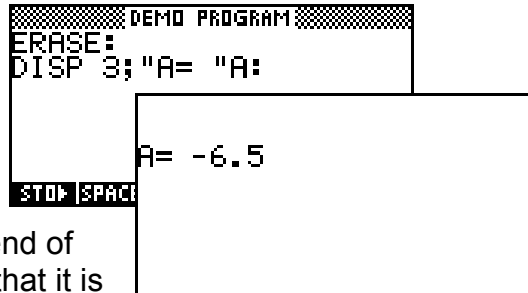
This command pops up a menu with the title specified and with however many options follow. The number of the menu option **OK** highlighted when the user presses **OK** is returned in the variable. The initial value of the variable before the CHOOSE statement determines which option is initially highlighted.



This value must be a valid one. Assigning an initial value outside the range of the menu may crash the program. If the user presses CANCL then a value of zero is returned but the program will still continue to execute from that point unless you include code to terminate it.

DISP <line number>;<expression>

This command breaks the display up into 7 lines and allows output to them.



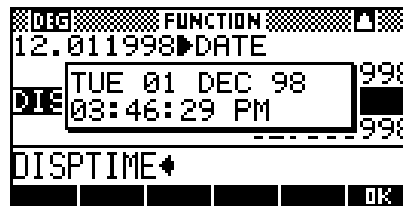
Using the DISP command on a line wipes that entire line to the right hand end of the screen before display. This means that it is not possible to use DISP to 'edit' material already present. The DISPXY command should be used for this purpose.

DISPXY <xpos>;<ypos>;;<expression>

This command displays the text/object/result contained in <expression> at the screen position specified using the font specified. An extensive example can be found on page 16.

DISPTIME

This command pops up a box displaying the calculator's internal time and date. These can be set by storing values to the variables Time and Date. Suppose the current time is 3:46:29 pm on the 1st of December, 1998 then you would store 15.4629 to Time and 12.011998 to Date.

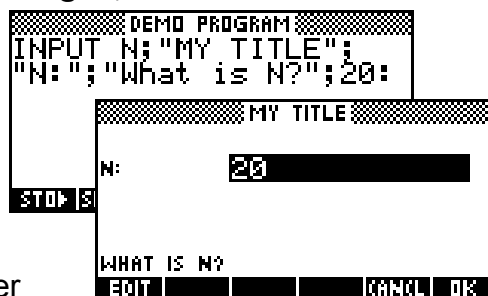


GETKEY <variable>

This pauses the execution until the user presses a key and stores the code for the key into the variable for later use. See the manual for how the value of the key changes according to whether it is pressed with or without the SHIFT or ALPHA keys. A key code of 53.1 would mean row 5, column 3 and unshifted. It can't readily be used for games because execution pauses instead of continuing in the background.

INPUT <variable>;<title>;<prompt>;<message>;<default value>

This command puts up an input view which can be used to obtain responses from the user. The degree of control over appearance is quite high as can be seen in the example.



If you want the default value to be whatever the user last input then use INPUT N;.....;N instead. If you do this then you will need to store and initial reasonable value into N before the first use of the INPUT command.

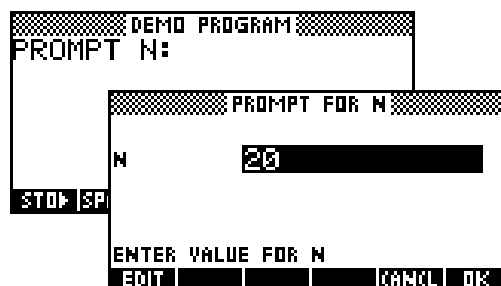
MSGBOX <expression>

This puts up a box with the text/expression you specify. If you want a newline character then just enclose a pressing of the ENTER key within the quotes.



PROMPT <variable>

This is a short form of the INPUT statement for those that don't require such precision of control over appearance. The default value is the current value of the variable.



WAIT <duration>

This command pauses execution for the specified number of seconds.



Calculator Tip

This list does not cover anywhere near the full range of commands, but it does cover enough that a competent programmer will be able to work out the rest independently. It is also enough that an enthusiastic amateur will be able to accomplish the more common tasks required in programming the HP.