# HP Prime : A Programmer's View

**Mark Power, mark.power@btinternet.com**

It has been around eighteen months since the HP Prime hit the market, so I thought it really is about time that I wrote about my experiences. Wlodek observed in his 2014 AGM report that whilst there was interest in the HP Prime, it has been nothing like that when the HP-41, HP-71 or HP-48 families were introduced. This is a shame as it is the first real major shift from those architectures and brings us an affordable, state of the art calculator.

Over the years I've progressed through a variety of machines: Commodore P50, TI-57, Casio fx-502p, Casio fx-702p, HP-41CV, HP-48SX, HP-48GX and now the HP Prime. Each of these was purchased to run programs that I've written and each was sold to pay for the next model. The Commodore, TI and Casio calculators primarily ran maths programs that I wrote when I was at secondary school and sixth form. When I could finally afford them I moved over to HP when I went to university to study computer science. The HP-41 allowed me to write software in FOCAL and machine code that otherwise I would have had to get on my bike and cycle several miles to try out on the university minis and mainframes. Best of all, I could carry these programs around with me.

My use of calculators for calculating really peaked at university with differential equations, matrices, integration and differentiation, errors in computer systems and hexadecimal arithmetic. Since leaving university, my use of maths has really regressed back to simple spreadsheets with the odd pie chart or graph. For this reason, I really feel that I'm totally unqualified to comment on the HP Prime's maths capabilities. Indeed I have pressed the CAS button on a couple of occasions, played for a few minutes, thought to myself "That's nice" and went back to doing simple sums and graphs.

So beyond simple maths programs, I've used programmable calculators as a platform for programs that people would use in their professions. On the Casio fx-502p I wrote a program to help a friend's Dad with rates calculations when he worked for the local council. On the HP-41 I wrote machine code routines to simplify entry of longitude and latitude for Colin Crowther so that he didn't have to worry about positive and negative value entry when flying Jumbo Jets. For myself I wrote a terminal emulator on the HP-48, which I recently ported to the HP-50 for a guy in the NHS who still uses the program to talk to medical equipment.

As well as this serious stuff, for every single calculator I've owned I've written simple games to get the hang of the programming language of the machine, determine its speed and generally show off the capability of a device that you can carry in your pocket. Of course, we now have amazing toys like the iPod Touch, iPhones and their clones that make our calculators, up to the HP-50g, look positively antiquated. So when presented with an HP Prime I was really exited to

try out programming a fast, colour touch screen machine with masses of storage and the obvious place to start was with a game.

## Programming on the go

I mentioned the iPod Touch, because my favourite game over the last few years has been a card game that it runs. This made me wonder how good the HP Prime would be at doing the same. I've tried a bit of iOS programming and whilst it is clear that great things can be accomplished you need a full development environment, a lot of time to get the hang of the frameworks and libraries, and a laptop or desktop on which to do the development.

The great thing about the HP Prime is that while not perfect for on the go programming, you can do it and without a massive learning curve. Indeed the Prime has some great features to help you with this: a reasonably straightforward BASIC like language, lots of memory to create copies of programs so that you can play without fear of destroying earlier versions and a very comprehensive built in help system so that you don't need to carry around a massive manual (though you can of course carry the PDF manual on your iDevice should you feel the need).

The other reason why I wanted to try on the go programming is that I don't actually own a PC and there is still, after eighteen months, no officially supported way to run the HP Prime software on a Mac - especially one as old as mine. If you have a Mac with an Intel chipset, you can try using WINE to run the HP Prime emulator, or if you are feeling really brave you can try libhpcalcs, a very basic portable (Windows/MacOS X/Linux) connectivity kit library. This seems like a pretty basic problem for the HP Prime given how common Macs are now and a backwards step when you think that as far back as the HP-48S/SX you could use pretty much any development machine you liked because the calculator supported the Kermit file transfer protocol.

## Playing cards

Back in Datafile V14N1 Joe Horn wrote a very neat card shuffler for the HP-48:

```
DECK, by Joe Horn (BYTES: #2378h 62.5)
<< 1 52 FOR x x DUP RAND * CEIL ROLLD NEXT 52  ->LIST >> INPUT: None OUTPUT:
{ 1 2 3 ... 52 }, randomly ordered.
```

This would be the foundation for my Prime programming attempt, as the Prime supports list of numbers and would also allow me to get an idea as to how fast a calculator based on a modern ARM chip could be when freed from the overhead of emulating a Saturn architecture.

This is the code I settled on:

```
NewDeckN(top)
BEGIN
 RETURN MAKELIST(I,I,1,top,1);
END;
```

```
NewDeck()
BEGIN
 RETURN NewDeckN(52);
END;

Shuffle()
BEGIN
 LOCAL c,l1,l2,l3,n;
 NewDeck()▶l1;
 FOR c FROM 1 TO 52 DO
  RANDINT(1,SIZE(l1)-2)▶n;
  SUB(l1,n+2,SIZE(l1))▶l2;
  SUB(l1,1,n)▶l3;
  CONCAT(l2,l1(n+1),l3)▶l1;
 END;
 RETURN l1;
END;
```

As you can see, the code is significantly longer, but I've made it a bit more flexible than the original. NewDeck() creates a list containing the numbers 1 to 52. NewDeckN(n) in called by NewDeck() and creates an arbitrary list of numbers starting at 1 and going up to n using a single command. Shuffle() uses NewDeck() to create a deck of cards and then shuffles them. This last one is much less elegant than the original HP48 version as chopping up lists is a bit more involved.

If you want to use these functions in programs of your own, you need to add EXPORT before the function name, otherwise they are local to the program file.

Despite the size of these functions, particularly Shuffle(), they serve to show how fast the Prime is. Wrap them in a function like this:

```
EXPORT TimeShuffle(n)
BEGIN
 LOCAL c, ts;
 TICKS▶ts;
 FOR c FROM 1 to n DO
  Shuffle()▶L1;
 END;
 RETURN (TICKS-ts)/n/1000;
END;
```

Run TimeShuffle(100) and you'll see that the Shuffle() program runs in about 0.02s. Joe's original on the HP48GX took 0.76s and my System RPL version in Datafile V15N2[1] took 0.48s. The modern hardware is certainly very quick.

If you look at my Shuffle() program you'll notice a few things:

[1] http://www.hpcc.org/datafile/V15N2/shuffle.html

1. I have used ► to store values in variables. I have been asked on the MoHPC forum how to enter this character. The answer is that it is by pressing Shift then EEX, which is labeled STO► on the keyboard. This style of assignment is shown in the HP Prime User Guide in some places and was the one I picked up on. Other programs in the User Guide use an alternative form of assignment which other people have picked up on as their first choice format:

   **variable:=value;**

   This is easier to type on a PC, arguably more consistent with other languages and easier to pass around in a PC environment or print as it doesn't rely on a special character. The choice is yours as to which to use, just don't try to switch from one to the other without switching around the order of the variable and the value.

2. I've used l1, l2 and l3 as variable names. That's lowercase L followed by 1, 2 or 3. These came about because I was playing with using the global list variables that are L0, L1, L2, L3… L9 and then just changed from upper to lower case. The variables on the Prime are case sensitive: "L0" is not the same as "l0".

3. This led to another observation: you need to be really careful with variable names on the Prime. If you happen to pick a variable name that already exists on the calculator, you can send up with valid syntax but when you run the code it behaves in ways you weren't expecting. For example: "ticks" entered in a program is the same as "TICKS" which is the same as "TICKS()". Maybe this was done to provide flexibility, but it does seem odd and can trip you up if you happen to pick the same name as a reserved word – and there are lots of those.

As my program got more complicated, other things have cropped up. IF (expr1) AND (expr2) AND (expr3) will always evaluate expr3 even if expr1 or expr2 is false. This is important as if you have a list, want to pick a particular element in it and then compare that element to a value. It is no good checking the length of the list in expr1 or expr2 and checking the value of the element in expr3 as it will cause the program to terminate abnormally.

Early firmware versions of the Prime were pretty buggy. Entering the debugger could quite often cause the Prime to crash and reboot. The latest version is much better, so if you have a Prime and haven't updated to 6975 then you really should.

There have been other changes between firmware versions. As well as additional functionality the sigma character used in ∑LIST and other keywords changed between different versions. So programs that worked perfectly on older versions of firmware generate errors on newer versions. This means you need to load the text,

find the syntax error, which fortunately is quite easy using on the Prime and change the keyword to the new version.

Before moving on I should mention one particular annoyance with programming the Prime directly: you rely a lot on the alpha keys. Getting used to the layout is expected with any new calculator. The real problem is the colour of the lettering on the keys. Under low energy bulbs in particular, the orange on white and worse still the orange on gray of the numeric keys is terrible which is a real shame as the rest of the hardware is excellent.

## Drawing cards

The main draw of the Prime over previous HP calculators is the colour screen. It is a world apart from all previous generations and whilst not up to "retina" resolutions, it is still stunning for a calculator. Newer firmware allows you to choose colours for lines on graphs rather than being stuck with a limited set of preset colours.

As already discussed the Prime is very fast, so the drawing primitives don't require rework to make them quick – unlike on the HP48 series. You also benefit from multiple graphics objects (grobs) being available into which you can draw, so you can prepare part of a picture in one grob and then paste or "BLIT" it into another nearly instantaneously.

I use this technique to build up the playing cards and overall board in my game and then BLIT them onto the screen with no flicker or delay.

Playing cards are easy to draw as the Prime has a very large character set built in which includes ♣♦♥♠, chess and dice symbols. These are in addition to Latin, Greek, Coptic, Cyrillic, Chinese, Punctuation, Subscript, Superscript, Currency, Arrows, Maths Operators, Block Drawing, Geometric Shapes and Dingbats.

## Taking input

In addition to the colour ability of the screen, it is touch sensitive and the latest firmware adds more touch gestures to the built in applications. If you are writing your own you can use MOUSE or WAIT to detect touches or gestures on the screen. For keyboard input you can use GETKEY or WAIT.

I chose to use MOUSE and GETKEY in my program as neither stop execution. If starting again with the input routines I would experiment with WAIT as it combines both mouse, gesture and keyboard input all in one. The downside being that it is quite difficult to unpick all the different options that it can return.

I found handling of the MOUSE result quite hard to process as you don't seem to be able to compare it to {{},{}} directly. This indicates that the screen was not touched. To overcome this I found I had to use the STRING function on the result of MOUSE and then compare it to the string "{{},{}}".

The downside of using MOUSE and GETKEY is that they form a busy wait, so my program runs continuously looking for either a screen touch or a key to be pressed. This leads to an unexpected side effect that I've yet to find a solution to.

Playing the complete game typically takes several minutes, and when you finish rather than being presented with the normal stack display the calculator turns itself off. This seems to be a bug in the automatic turn off handling and is present across all versions of firmware up to and including 6975. I suspect that GETKEY and MOUSE are not resetting the automatic turn off timer.

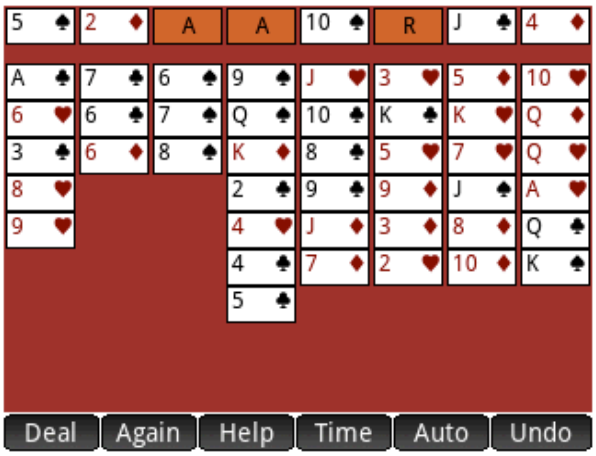
In an early version of my code I used INPUT and MSGBOX to display the final congratulations screen and score, but I found that these do look at the auto turn off state and so rather than showing the score, the calculator turned off. Hopefully this will get fixed in some future firmware version. In the meantime I've had to implement my own version of MSGBOX rather than rely on the built in version.

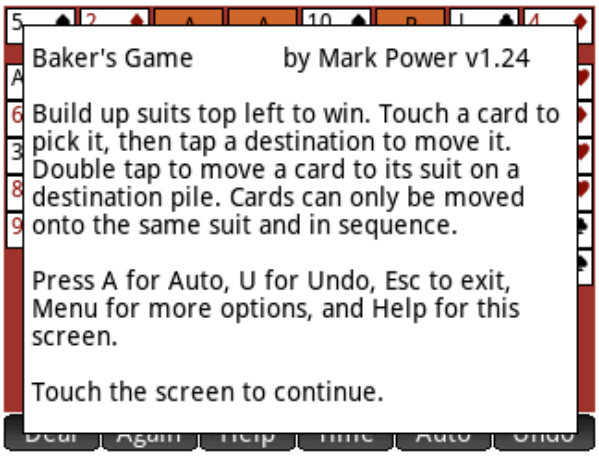
## Putting it all together

I'm not going to reproduce the whole of the game here in Datafile as it runs to many pages. Instead you will need to look at it or download it from:

http://www.hpcc.org/programs/hpprime/sol_bg/SOL_BG.txt - or -

http://www.hpcc.org/programs/hpprime/sol_bg/SOL_BG.hpprgm




## Instructions – Baker's Game

As you can see from the screen shots above I have put in a simple help screen, but feedback from MoHPC suggests that my description of Baker's Game is too brief, so here are expanded notes (with thanks to Jeff O.[2]).

At start up the board has three main areas. In the top left are four holders labelled with "A". These are the destination piles that you put cards onto in suits, starting with Aces. Once a card is here it cannot be moved back unless you use Undo.

Top right are four holders labelled "R" which can each hold one "Reserve" card. The centre of the screen comprises 8 columns of cards. You can only move the bottom card of each column. You can only move a card to another column if the

[2] http://www.hpmuseum.org/forum/thread-379.html

cards are in the same suit and in descending order (Ace low). You can move any single card to an empty Reserve holder. Reserves can move back to the main deck.

You can move cards by touching them to highlight them and then touching a destination location. If the location is not a legal move, the highlight is removed. To speed things up when putting cards onto the destination piles top left you can double tap a card and it will move without needing to move your finger.

The game is over when you have built up the suits on the destination piles top left so that four kings are shown and the rest of the board is empty.

As well as touching the screen to move cards you can use the keyboard. Pressing Help shows the help screen shown above. Pressing the Menu button shows actions that are selected by touching the screen. The Esc button exits the game. Pressing the A (Vars/Chars) button or Auto soft menu button moves as many cards as possible from the Reserve holders and main deck into the destination piles. You may find this useful towards the end of a game when you have built up runs of cards in the main deck.

You can Undo moves either with the soft menu option or by pressing the U (4/Matrix) key. I've limited Undo to the last 10 moves to avoid running out of memory as I hold the whole of the deck. It's not very efficient in memory terms but it is easy to program. If you want another go at the same deal and Undo doesn't give you enough to work with press the Menu button and then touch Again on the screen. If you've had enough and want a new deck press Menu and then Deal.

## Summing Up

Like all your first programs on a new platform, you look back on your code and cringe. I hadn't spotted global definitions until I saw code other people put on the Internet so I've got magic numbers all over the place. I was intrigued about "►" versus ":=" and ended up with both. I wanted to use meaningful names for variables, but the orange keyboard legends just made this a chore. All in all the code could do with some refactoring and tidying up which I'll do when there is Mac connectivity software. It's not yet the model of tutorial code I hoped it would be.

On the positive side I was able to knock this up on the calculator, with some notes jotted on a few pieces of paper, without having to learn a massive framework or use a separate PC or Mac as a development environment. As for playability, this game is as responsive and I think as engaging to play as versions on other platforms.

Machine code games on the HP48 series became very well developed over time and what I think is encouraging about the Prime is that you can get the same responsiveness without resorting to low level code. You also get that fabulous colour screen and touch capability.

Early Prime firmware versions were very frustrating but 6975 seems much more stable. Perhaps not up to HP41/HP48 standards, but getting there. If you've not explored the Prime until now, I'd strongly recommend you give it a go.