HP48 Universal Font Library (UFL)

Andre Schoorl <aschoorl@engr.uvic.ca>

UFL v1.0.2

Contents

1 Introduction

1.1 Terms of Distribution

1.2 Scope

An increasing number of HP48 programs are being developed that have their own fonts rather than using the fonts built into the HP48. This is due to a number of reasons:

- Accessing the built-in fonts is slow.
- The built-in fonts do not support all the characters and cannot be modified.
- There is no way to distinguish between lower and upper case characters with the built-in small font.

A number of popular HP48 programs overcome these problems by providing their own copies of the fonts. Almost all of these are based on FNT1 and FNT2 from the Jazz library. Before the UFL, the authors of such programs were forced to either:

- Keep their own copy of one or more fonts in their program so that it would still run if the user does not have Jazz.
- Rely on the existence of Jazz in order for the program to run.

Both options are not ideal, because in the first case with just a few such programs the user ends up having multiple copies of the same font, which is a waste of room. The second case relies on the presence of Jazz, which most people don't have room for. The Universal Font Library (UFL) is a standard designed to alleviate both of the above problems.

The basic idea is to have a separate, standardized library containing FNT1 and/or FNT2. Various levels of the UFL are provided that contain the various font combinations, so users don't need to have fonts around that none of their programs require.

Also, the UFL allows for the separation of fonts and utilities to maintain them. We have developed a library for the UFL known as the Font Maintenance Library (FMnt) that can be downloaded separately from the UFL. It allows the user to check the status of their UFL library as well as recall and edit fonts. Once they are satisfied with their setup, they can purge the Font Maintenance Library to save room. Furthermore, the FMnt library is distributed with several alternate fonts.

The benefits to such a system are immediately evident. Most importantly, if the user has even more than one program using the same font, they start saving valuable room on their calculators. In addition, if the font in the UFL is edited, all of their favorite programs using the UFL immediately start using the updated font.

Furthermore, the UFL opens the doors for programmers who were previously unwilling to add an extra 1 or 2K to their library to use the fast FNT1 and/or FNT2 routines. With the UFL, they no longer need to worry about keeping their own copy of the font.

2 Using the UFL and Font Maintenance Libraries

2.1 Library Numbering

Each UFL library is assigned a "level" that indicates which fonts it contains. If we assign one bit for each font, we can come up with a binary representation as follows:

```
MSB   8 4 2 1  LSB
      . . x y
          ^ ^
       FNT2 FNT1
```

Using this scheme, we see that a library with only FNT1 would be assigned level # 01b (1), a library with only FNT2 would be assigned level # 10b (2), and a library with both FNT1 and FNT2 would be assigned level # 11b (3). Notice that with this numbering convention, if an addition FNT3 is ever added, the numbering of the existing libraries would remain unchanged.

2.2 Libraries in this Package

Using the decimal levels for FNT1 and FNT2 we see there are currently three UFL libraries, as follows:

- ufl1.lib - Contains only FNT1 (Font level 1)
- ufl2.lib - Contains only FNT2 (Font level 2)
- ufl3.lib - Contains both FNT1 and FNT2 (Font level 3)

All UFL libraries have the same library number of 257 (# 101h).

You only need to load one of these libraries on your calculator. Which particular one you need will depend on what applications you use that require the UFL. See the documentation that came with the programs you are using to determine whether you need FNT1, FNT2, or both. If you are getting lots of "Font not Found" errors, most likely you do not have the correct UFL level loaded on your calculator.

There is only one Font Maintenance Library (FMnt) and it is in the file 'fmnt.lib'. It has library number 258 (# 102h).

You only need to load the FMnt library if you wish to check the status of your UFL or if you want to edit or recall the fonts in it. If you are satisfied with the fonts that come with the UFL, you do not need the FMnt library. However, if you decide to change the fonts, you are free to purge the FMnt library once you have edited the fonts to your liking.

2.3 To Install a Library

1. Download it into a variable of the calc.
2. Place the library on the stack.
3. Type the number of the port in which you want to store the library in. 0 is appropriate if you do not have any expansion cards.

4. Press STO to store the library in that port.
5. Turn the calculator off and then on, or press ON-C, to finish installation of the library.

Sometimes you have to purge the variable containing the library before storing it in a port, if you are low on memory.

2.4 To Delete a Library

Say you have library 257 in port 0 that you want to delete.

1. Go to the directory to which the library is attached. In most cases this will be HOME.
2. Enter :0: 257
3. Press [ENTER] or type DUP
4. Type DETACH
5. Type PURGE

2.5 UFL Commands

2.5.1 FNT

Usage: (size -> fnt)

FNT is the user interface to the UFL's fonts. It takes the size of desired font and recalls the font as a library data object if it exists. The size should be a real number or system binary. Indexing is one relative so 1 corresponds to FNT1 and 2 corresponds to FNT2.

Will error with "Font not Found" if the level of UFL you are using does not contain the requested font.

2.5.2 Remaining Access Routines

The remaining access routines in the UFL are hidden from the library menu as a means of encapsulation, and to protect the general user from routines that do not perform argument checking. The interface to these is given in detail for System RPL programmers in the "System RPL Specifications" section of this document.

2.6 FMnt Commands

2.6.1 FSTAT

FSTAT shows a simple screen that allows you check the status of your UFL library. The information shown includes yes or no answers for the following:

- Whether the UFL is installed or not.
- Whether FNT1 is present.

- Whether FNT2 is present.
- Whether the fonts are editable.

Whether FNT1 or FNT2 is present will depend on what level of UFL you have installed. In order for the fonts to be editable the UFL must be in port 0 or an uncovered RAM card (0/1/2 on SX, 0/1 on GX). If the UFL is on a RAM card it must be unlocked for the fonts to be editable.

2.6.2 sFED

Usage: (grob -> grob)

sFED is a FNT1 editor designed for the FMnt library. It takes a 128 x 48 FNT1 grob, allows you to edit it, and leaves the edited grob on the stack. You can use F1-> to provide your current FNT1 as the initial input to sFED. Once you are done you can update your UFL library by using the output of sFED as input to ->F1 (see below). For a G/GX FNT2 editor, see the README in the 'contrib' directory.

sFED will error under the following conditions:

- The provided grob does not have the correct dimensions.

Usage instructions for sFED:

Main Screen:

Edit or [ENTER]
Edit the currently highlighted character.

Quit or [ON]
Quit sFED. If you made any changes, you will be prompted if you want to save your changes upon exit.

About
About information

Edit Screen:

Dot+
Continuous draw

Dot-
Continous erase

[+/-]
Toggle "pixel"

Abort or [ON]
Abort any changes made to the character

Keep or [ENTER]
Keep all changes made to the character

2.6.3 ->F1

Usage: (grob ->)

Takes a 128 x 48 grob and changes FNT1. Modifies the UFL library with the new FNT1 if it is in an accessible port. Will error under the following conditions:

- The UFL library does not exist (install the UFL first).
- FNT1 is not supported in the level of UFL you are using (install a level of UFL that supports FNT1).
- The UFL is not editable. In order to edit the fonts you must have the UFL in an accessible port.
- The provided grob does not have the correct dimensions.

The original FNT1 is stored in the file 'fnt1.zip' as 'ufl1.fnt'.

2.6.4 F1->

Usage: (-> grob)

Returns the current FNT1 as a 128 x 48 grob. This output can be edited to taste and then used as input to the ->F1 command. The output of F1-> can be used as input to sFED. Will error under the following conditions:

- The UFL library does not exist (install the UFL first).
- FNT1 is not supported in the level of UFL you are using (install a level of UFL that supports FNT1).

2.6.5 ->F2

Usage: (grob ->)

Takes a 192 x 64 or 6,7,8 x 2048 grob and changes FNT2. Modifies the UFL library with the new FNT2 if it is in an accessible port. Will error under the following conditions:

- The UFL library does not exist (install the UFL first).
- FNT2 is not supported in the level of UFL you are using (install a level of UFL that supports FNT2).
- The UFL is not editable. In order to edit the fonts you must have the UFL in an accessible port.
- The provided grob does not have the correct dimensions.

The original FNT2 is stored in the file 'fnt2.zip' as 'ufl2.fnt'.

2.6.6 F2->

Usage: (-> grob)

Returns the current FNT2 as a 192 x 64 grob. This output can be edited to taste and then used as input to the ->F2 command. Will error under the following conditions:

- The UFL library does not exist (install the UFL first).
- FNT2 is not supported in the level of UFL you are using (install a level of UFL that supports FNT2).

2.7 List of Possible Errors

There are several errors that can occur with the above commands due to non-existence of the UFL, the combinations of fonts, and locking of RAM cards. The possible errors are:

UFL not Found
: You have either not installed the UFL library, or not installed a valid UFL library. Install an appropriate level of the UFL as described above for the programs you are using.

Font not Found
: The specific font required by the program you executed was not found in the UFL you have installed. You need to determine what font is missing and install a different level of the UFL that contains it.

Cannot edit UFL
: You tried to modify one of the fonts with the UFL installed in a locked or covered port. In order for the fonts to be editable the UFL must be in port 0 or an uncovered RAM card (0/1/2 on SX, 0/1 on GX). If the UFL is on a RAM card it must be unlocked for the fonts to be editable.

G/GX Only
: You cannot use this binary on an S/SX machine. The only place this should occur is in the contributed FNT2 editor, FED.

3 User RPL Access Routines

3.1 Supplied Directory

The UFL comes with a directory of routines in ‘access.dir’ which allow you to use the UFL’s fonts in your own programs. These perform argument checking and can be used in User RPL programs or as the basis for System RPL programs. Included in this directory are the following access routines.

3.1.1 UFL?

Usage: (-> flag)

Returns 0 or 1 on whether the UFL library exists.

3.1.2 FNT1?

Usage: (-> flag)

Returns 0 or 1 depending on whether the FNT1 exists. Errors if the UFL does not exist, so check UFL? first.

3.1.3 FNT2?

Usage: (-> flag)

Returns 0 or 1 depending on whether FNT2 exists. Errors if the UFL does not exist, so check UFL? first.

3.1.4 U->GROB

Usage: (obj size -> grob)

Converts the specified object to a grob with the requested size font. This access routine is designed to ease User RPL access to the UFL, and is meant to be as compatible as possible with the built in ->GROB command.

U->GROB will not error if the UFL does not exist or if the requested font in the UFL does not exist. Instead, it will fall back on calling the built in ->GROB. This means users of U->GROB do not have to do many of the checks normally associated with use of the UFL and do not have to trap UFL errors.

Only strings will be be converted with the UFL fonts; all other objects are passed to ->GROB. Because there is currently no FNT3 in the UFL, a size of 0 or 3 will always be passed to the built in ->GROB.

3.1.5 F1->G

Usage: (-> grob)

Recalls the current FNT1 as a 6 x 2048 grob. Errors for insufficient memory or if FNT1 does not exist. The output of this command is suitable for some older packages.

3.1.6 SGROB

Usage: (string -> grob)

Takes a string and converts it to a graphics objects using FNT1. Errors for bad argument, insufficient memory, UFL does not exist, or FNT1 does not exist.

3.1.7 SDISP

Usage: (string x y ->)

Takes a string on level 3, a real x coordinate on level 2, and a real y coordinate on level 1, then displays string on the screen with the supplied coordinates using FNT1.

- x is in steps of four pixels, so 0 <= x <= 33.

- y is in steps of one pixel, but will not display in the menu bar, so 0 <= ¡ y <= 51.
- If the string runs beyond the screen boundary, SDISP stops displaying.
- If the given coordinates are out of range, SDISP does nothing.
- SDISP does no special handling for newline characters, and treats them like any other character.

3.1.8 SDISP9

Usage: (string ->)

Clears the stack grob and displays the first 9 lines of the string from stack level 1 with FNT1. String can be any size. Leaves menu area alone.

3.1.9 SSTAT

Usage: (string ->)

Clears the status area and displays the first 2 lines of the string from stack level 1 with FNT1. String can be any size.

3.1.10 F2->G

Usage: (-> grob)

Recalls the current FNT2 as a 8 x 2048 grob. Errors for insufficient memory or if FNT2 does not exist. The output of this command is suitable for the contributed font editors or for StringWriter's 'FONT' variable.

3.1.11 MGROB

Usage: (string -> grob)

Takes a string and converts it to a graphics objects using FNT2. Errors for bad argument, insufficient memory, UFL does not exist, or FNT2 does not exist.

3.1.12 DISPN

Usage: (string line ->)

Takes a string on level 2 and a real number on level 1, then displays the string on the screen using FNT2. The real number should be a line number between 1 and 7, where 1 is the top of the screen and 7 is the bottom. Errors for insufficient or bad arguments, UFL does not exist, FNT2 does not exist, or if the specified level is not between 1 and 7.

3.2 Error Handling Hierarchy

To avoid bogging down the main access routines, each level of error checking assumes you have properly checked the previous level. For example, if the UFL does not exist and you try to use DISPN, you will find you get "Error: Undefined XLIB Name" rather than a nice error message.

4 Replacements Fonts

4.1 Changes to FNT2

FNT2 has been changed slightly from the previous copy in Jazz to be more universably suitable. A total of 7 characters are affected, as follows:

```
ASCII    Previous          Now
==================================================================
0        Ellipsis          Null star from old FNT1
9        Diamond           Tab character from Java
10       Inverse Diamond   Newline character from Java
12       Female Symbol     Formfeed character from Java
13       Music Note        Carriage Return from Java
30       Up Arrow          Scrolled off to left in edit line (...)
31       Down Arrow        Scrolled off to right in edit line (...)
```

The tab, newline, and two scrolling characters are the most important for general use; the other characters were changed to be consistent with Java.

4.2 Zip Files

The UFL comes with two zip files, 'fnt1.zip' and 'fnt2.zip'. These contain several alternate fonts to the ones that come with the UFL. However, not all may be suitable for the applications you are using.

- The original FNT1 is stored in the file 'fnt1.zip' as 'ufl1.fnt'.
- The original FNT2 is stored in the file 'fnt2.zip' as 'ufl2.fnt'.

4.3 Credited Fonts

These fonts deserve special credit since they were designed specifically for the UFL. A description of each follows.

If you come up with some good alternative fonts, please mail them to me so I can include them in future releases of this package.

fnt1/euro1.fnt and fnt1/euro2.fnt

From: Joergen Blomgren

euro1.fnt:

- Character 0-31: Designed on fnt2/ufl2.fnt, except a few characters.
- Character 3-6: Cards are replaced with the first letter of the name.
- Character 29: Single plus sign instead of double plus sign.

euro2.fnt:

- Characters 0-31: ASCII/character-value except:

- Character 0: Star as in UFL
- Character 7: A musical note (bell)
- Character 8: BS (backspace)
- Character 9: >| (tab forward)
- Character 10: LF (line feed)
- Character 12: FF (form feed)
- Character 13: CR (carriage return)
- Character 27: EC (escape)
- Character 30: .. (left continue)
- Character 31: .. (right continue)

Both:

- Characters 32-126: Standard characters, with small modifications.
- Character 101: Is changed in style of fnt1/euro3.fnt
- Character 127: In style of the HP48 font.
- Characters 128-159: Greek and graphic characters in style of the HP48 font.
- Characters 160-255: International and some graphic characters in style of HP48, all characters are on the same baseline, so some characters are a bit squashed, but I think it looks bad when characters are unaligned in the base line. Lowercase e umlauts are changed in the style of fnt1/euro3.fnt.

fnt1/euro3.fnt

From: Rodney Hess

- The baseline for this font is the original FNT1 contained in the UFL release (fnt1/ufl1.fnt).
- The lowercase e (char 101) has been modified to look like Java's. As Java's e is more readable with accents than the original, a consistent look among the lowercase e's is maintained with this change.
- Character 127 has been replaced with a greyscale block along the lines of the one found in the built-in big font.
- Character 161 replaced with a cent symbol.
- The original upside-down question mark (char 191) was facing the wrong direction. This has been fixed.
- Characters 192-214, 217-221: Capital characters. Some of these sit below the baseline to aid legibility. For clarification, the twiddle mark is represented as two pixels right justified, for example, as in character 209.
- Characters 224-246, 249-253, 255: Lowercase characters.

fnt1/euro4.fnt

From: Ivan Rivera Rodriguez

This is an european-suitable small font for use with the UFL. It contains the same characters and ordering as the HP standard font, except for the first 30 characters, which are the same used in the standard UFL FNT1. The different accentuations are nearly always discernible, except for the circumflex, which had to be reduced to a single point in order to properly identify the vowels under it.

5 System RPL Specifications

This section is designed for System RPL programmers who wish to use FNT1 and/or FNT2 in their own applications. If you just need the UFL for an application you want to use, you don't have to read this section.

In order to use System RPL or HP48 machine language you need a compiler/assembler such as Jazz or GNU-Tools. See the HP48 FAQ for more information on how to get started programming in System RPL.

5.1 Programmer's Interface

The programmers' interface to the UFL has been designed to simplify checking existence of fonts as well as allowing for future expansion if necessary.

Due to the requirements of the Font Maintenance Package (FMnt) we had to go with fixed XLIBs for each font. However, we tried to do this in a way that didn't restrict further growth of the UFL, while at the same time minimizing overhead. This is the result we came up with:

```
ROMPTR 101 0    xFNT          ( user-accessible )
ROMPTR 101 1    Font
ROMPTR 101 2    Font?
ROMPTR 101 3    FNT1
ROMPTR 101 4    FNT2

ROMPTR 101 5 and on are for future expansion if necessary.
```

Notice that once the checking for fonts has been done, there is no performance loss imposed by using the UFL because the application using the fonts can call the appropriate XLIBs directly.

Libraries that do not have FNT1 or FNT2 respectively will still have the XLIBs in the above order, however if you try to recall the font that doesn't exist you will get an error. See the next section for more information.

5.2 Behaviour of each XLIB

Since there is more than one UFL library, the behaviour of the XLIBs in each library must be consistently defined. The following standard is defined:

xFNT (ROMPTR 101 0)

This is a user-accessible XLIB that accepts either a real or system binary and recalls the specified font if it exists. Otherwise it issues the error "Font not Found" and leaves the original argument on the stack.

Font (ROMPTR 101 1)

This is a nullnamed equivalent to xFNT that only accepts a system binary. It either recalls the specified font or issues the error "Font not Found" and drops the system binary.

Font? (ROMPTR 101 2)

Accepts a system binary and returns TRUE or FALSE depending on whether the font number specified exists or not.

FNT1 (ROMPTR 101 3)

Either returns FNT1 directly as a Library Data object or issues the error "Font not Found" if you neglected to check if FNT1 actually exists with Font or Font? first.

FNT2 (ROMPTR 101 4)

Either returns FNT2 directly as a Library Data object or issues the error "Font not Found" if you neglected to check if FNT2 actually exists with Font or Font? first.

5.3 Coding Standards

With font standards come coding standards. We recommend you create a file called ufl.h containing the following defines:

```
DEFINE  xFNTSTR         $ "FNT"
DEFINE  xFNT            ROMPTR 101 0
DEFINE  Font            ROMPTR 101 1
DEFINE  Font?           ROMPTR 101 2
DEFINE  FNT1            ROMPTR 101 3
DEFINE  FNT2            ROMPTR 101 4
```

You should then include this at the start of your program with:

```
        INCLUDE ufl.h
```

Furthermore, the check for existence of the UFL has been standardized with the following code, which will return TRUE or FALSE depending on whether the UFL library exists or not:

```
NULLNAME UFL?
::
 xFNTSTR palparse DROP ' xFNT EQUAL
;
```

You should check this at the start of your program. If it returns FALSE you should generate the error "UFL not Found".

The only exception to the above check should be if you need to check for the UFL in a library config routine. In that case the above code will not work because palparse is not valid at config time. If this is the case, you will have to use the following, inferior check:

```
NULLNAME UFL?
::
 ' Font ROMPTR@ NOTcaseFALSE DROPTRUE
;
```

Once again, do not use this alternative method if you do not have to! The only case where there should be a need for this check is in replacement stack programs such as Java.

For the font checks themselves, you do not have to provide the error message yourself. Instead, you can take advantage of the fact that the UFL will generate an error for you. See the Examples section for details.

5.4 Summary of Usage

Named XLIBs:

- UserRPL programs can use xFNT and/or the access routines provided with the UFL.
- SysRPL programmers can use palparse on "FNT" if they don't want to hardcode any XLIBs.

Nullnamed XLIBs:

- SysRPL programmers can pass Font a system binary to get a font if they are willing to do error trapping, or if they don't mind their application exiting (i.e. at the beginning of their program)
- SysRPL programmers should check Font? before accessing any fonts directly. After they have confirmed the font they want exists, they are free to call them directly for speed.

5.5 Examples

Let's say someone wants to write a library that only needs FNT1. The simplest way to use the UFL would be something like:

```
xNAME   UFLTEST
::
 CK0
 FNT1 DROP                      ( Two possible errors - see below )
 ...
 FNT1
 ...
;
```

Notice that FNT1 (or ONE Font) will provide an appropriate error message "Font not Found" for you if the font you want doesn't exist. Another possibility is that the UFL library doesn't exist at all, in which case the above example will cause Error: Undefined XLIB Name.

This error isn't very informative, so if possible you should use the UFL? check given earlier to generate the standard error "UFL not Found". If you are making a library you should add this error to your message table. With these changes the improved code segment becomes something like:

```
ASSEMBLE
=xxMESSAGE
RPL
ARRY
[
 "UFL not Found"
]

DEFINE  UFLErr  #xxxxx

xNAME   MYLIB
::
 CK0
```

```
 UFL? NOTcase :: UFLErr ERROROUT ;
 FNT1 DROP
 ...
 FNT1
 ...
;

NULLNAME UFL?
::
 xFNTSTR palparse DROP ' xFNT EQUAL
;
```

The UFL? check does not necessarily have to be its own nullnamed XLIB, but it is handier that way if you have more than one place to check it in. Alternatively, for the above example we could use the following method that does everything in one place and is slightly smaller:

```
NULLNAME UFL?DoErr
::
 xFNTSTR palparse DROP ' xFNT EQUALNOTcase :: UFLErr ERROROUT ;
 FNT1 DROP
;
```

This checks both for the existence of the UFL and errors if FNT1 does not exist. This way, each user accessible XLIB only needs to call UFL?DoErr once. After it has passed the checks you are free to use FNT1 directly. These methods also have the benefit that they allow error trapping for your routines when called by external programs.

Of course, everything above applies to FNT2 as well, you would just replace all references to ONE with TWO and FNT1 calls with FNT2. If you need both then you would just use an appropriate combination of FNT1, FNT2, Font, and/or Font? calls that check both.

5.6 Conversion of Existing Programs to the UFL

Conversion of existing programs that have their own copies of FNT1 or FNT2 should be relatively straightforward. The two steps are:

- Add code on startup of the program to check for the UFL and if the required font(s) exist. If the UFL is not found, exit with the standard error "UFL not Found".
- Replace all calls to your own copy of the font with corresponding calls to the UFL fonts, using the given DEFINE's for consistency.

5.7 Stack Access and Purging

One gotcha when using the UFL is the fact that if your program allows access to the stack, the user might actually purge the UFL while you are using it! This also applies if you are writing some sort of file browser that allows people to purge files. In Java we deal with this by simply binding a LAM to something small in the UFL to prevent purging. For example:

```
::
 ... Check existence of UFL as above ...
 ' Font? ROMPTR@ DROP { LAM #whatever } BIND
 ...
 ABND
;
```

If the user tries to purge the UFL with the above LAM bound, they will get Error: Object in Use.

6 Credits

The UFL was co-developed by a group of programmers who deserve special credit. Their names in alphabetical order and respective roles are as follows:

- Alfred Arduengo: Design ideas and testing.
- Mika Heiskanen: Design ideas and the original FNT1 and FNT2 from Jazz.
- Nathanael Morriss: Contribution of extra FNT1 access routines.
- Arnold Moy: Design and implementation of FNT1 editor/recaller.
- Jarno Peschier: User RPL access suggestions and testing.
- Jack Levy: Design ideas, testing, and FNT1 access routines.
- Andre Schoorl: Original concept, documentation, maintainer.
- Richard Steventon: Design ideas and the FNT2 recalling, editing, and resizing routines.

Font Designers:

- Joergen Blomgren for fnt1/euro1.fnt and fnt1/euro2.fnt
- Rodney Hess for fnt1/euro3.fnt
- Ivan Rivera Rodriguez for fnt1/euro4.fnt