Quartus

Handheld Software

www.quartus.net

Quartus Forth 2.0.0

User Guide and Reference Manual

Revision 05/12/05

Table of Contents

Introduction and Frequently-Asked Questions

Quartus Handheld Software is proud to present the latest release of Quartus Forth, the on-board native-code compiler for Palm Powered handhelds.

This manual is a living document; expect revisions. Please send all comments, criticisms and suggestions to support@quartus.net.

By way of introduction to Quartus Forth, here's a collection of Frequently-Asked Questions:

Q. **What is Quartus Forth?**

A. Quartus Forth is a on-board Standard Forth compiler for Palm Powered devices, used to create apps for the Palm OS platform. It generates tight, native-code, optimized stand-alone apps that have full access to the Palm OS API, require no run-time libraries, integrate seamlessly with the Palm GUI, and are compatible with the broadest-possible range of Palm devices.

Q. **What languages are supported by the Quartus Forth compiler?**

A. Quartus Forth supports:

- ISO/ANS Standard Forth (with Palm OS extensions)
- Inline Motorola 68000 assembler (for all Palm Powered devices)
- Inline ARM assembler (for devices running Palm OS 5 and later)

Q. **How does Quartus Forth differ from other Palm development tools?**

A. Quartus Forth:

- Runs entirely on-board your Palm device.
- Generates true native-code stand-alone PRC apps requiring no run-time libraries, with no run-time licensing fees.
- Has an interactive console for development testing and debugging.
- Provides nearly 900 Palm OS functions and over 6200 Palm OS constants and structures integrated directly into the compiler.
- Provides full access to application globals for all Palm launchcodes.

Q. **Why an on-board compiler?**

A. A portable on-board compiler is remarkably handy:

- You can work on your code anywhere the mood strikes -- on a train, in a waiting room, at a coffee shop, on the beach, etc. Time and ideas that might otherwise be lost can be capitalized on wherever you might find yourself.
- You can test your code immediately, without having to synchronize your app with a Palm device each time -- it's already there.
- Feedback is instantaneous, and the edit-compile-test cycle is very fast.

Q. **Why Forth?**

A. For many reasons. Here are a few:

- Forth is an excellent and powerful structured programming language with a long history of suitability for rapid and effective software development, particularly (but not solely) in embedded systems like the Palm.
- Forth is fast, both in compilation speed and in the speed of the resulting executables.
- Forth is concise and brief, so it's well-suited for Palm devices, where screen real estate is limited.
- Forth uses few special characters, so it's a good fit for Graffiti entry.
- Forth is backed by an ISO/ANS Standard, finalized in 1994 and in widespread use by commercial Forth vendors and developers alike.

Q. **I thought Forth was interpreted and therefore slow?**

A. There are many implementation strategies for Forth compilers. Quartus Forth is an optimizing native-code compiler that directly generates executable machine instructions; it is fast. Apps developed with Quartus Forth are virtually indistinguishable in speed from those developed with desktop compilers.

Q. **But isn't Forth an interactive language, like BASIC?**

A. Forth is indeed interactive. There's a console from which code can be entered and tested interactively -- but that's where the similarities to BASIC end. Forth definitions, whether entered at the Quartus Forth console or read from source files, are not interpreted, but rather are compiled directly to fast native-code machine instructions.

Q. **Is all that event-handling scaffolding that I see in Palm C sources necessary in a Quartus Forth app?**

A. No. Quartus Forth takes care of administrative tasks such as loading forms, managing events, and application startup and termination for you; your app need only request events via the EKEY word, and act on them accordingly. This greatly simplifies app development.

By way of example, here's the complete source of a "Hello, World!" app:

```
\ hello

: go
  MainForm
  ." Hello, World!"
  begin  ekey drop  again ;
```

Q. **Can I use native Palm GUI resources and Palm OS functions in my apps?**

A. Yes. Quartus Forth integrates cleanly with the Palm OS, providing sophisticated event-handling, callbacks, and named access to nearly 900 Palm OS functions from Palm OS 1.0 to Palm Garnet (OS 5). More than 6200 Palm OS constants and structures are directly available by name.

Quartus Forth provides simple commands for copying externally-created Palm GUI resources into your stand-alone apps.

Q. **Can I work with floating point values?**

A. Yes. Quartus Forth provides a range of floating point facilities:

- Built-in Motorola Fast Floating Point (FFP) single-precision floats, providing simple floating point support that works all the way back to Palm OS 1.0.
- NewFloatMgr support under Palm OS 2.0 and above, providing IEEE 754 single- and double-precision floating point operations.
- Full MathLib shared library support for extended double-precision floating point operations.
- Full support for the parsing and display of floating point values in both standard and scientific notation.

Q. **What enhanced features does Quartus Forth provide?**

A. Beyond being a full-featured Standard Forth system, Quartus Forth provides a wide range of other facilities:

- Tight integration with the Palm OS
- A simplified event-handling system for managing Palm OS events
- Full access to the Palm API, with over 6200 built-in named constants and structures
- Text error messages for more than 400 Palm API error return codes
- Extensive floating-point support
- Launchcode support (with globals available for all launchcodes)
- Named modules and full multiple-namespace wordlist support
- Deferred words
- Extensive single- and double-precision floating point support
- Inline assemblers for Motorola 68000 and ARM CPUs
- An integrated disassembler for the Motorola 68000
- Library source modules for encryption, graphics, sound, and more

Q. **How do I create and edit source files?**

A. Quartus Forth can read Forth source directly from MemoPad memos, or from standard Palm Doc-format files (compressed or uncompressed). The built-in MemoPad app is the default choice for editing Quartus Forth sources.

Q. **Can I use an alternate editor to write apps?**

A. Yes. Third-party MemoPad editors (such as pedit) or Doc editors (such as QED) work fine.

Q. **Where can I get assistance with Quartus Forth development?**

A. The Quartus Discussion Forum (http://quartus.net/discuss) is an excellent place to start; it is frequented by a group of intelligent and helpful developers. The Quartus Forth staff and development team are also available in the discussion forum, and via email. Quartus Forth is a mature, well-tested and well-supported compiler; many developers have years of experience developing apps with it.

There is also a user-maintained Quartus Wiki, at http://quartus.net/wiki. It's a treasure-trove of information from developers using and extending Quartus Forth. Additionally, the Quartus.net File Area (http://quartus.net/files) has an ever-growing collection of sample sources and other support materials.

Q. **What are the system requirements for the Quartus Forth compiler?**

A. The Quartus Forth compiler runs on Palm OS devices from Palm OS 3.0 and up. With all library sources installed, it occupies less than 400K of Palm memory.

Q. **What platforms do the compiled stand-alone apps run on?**

A. Assuming your apps do not use Palm OS API functions that are only available in specific versions of the Palm OS, apps created using Quartus Forth can run on any Palm device from Palm OS 1.0 all the way to Palm Cobalt (OS 6).

Q. **How do I get my stand-alone apps out of the Palm?**

A. Transfer of your stand-alone apps happens automatically when you perform a HotSync operation; they appear on your hard drive in the Backup directory of the Palm Desktop application. They can also be directly beamed from one Palm to another, transferred via external storage, etc.

Q. **Is there a desktop version of the Quartus Forth compiler?**

A. Quartus Forth can be used on the desktop via the Palm OS Emulator or the Palm OS Simulator. There are no native desktop versions at this time.

Q. **You say there's no run-time library required, but isn't a run-time library effectively crammed into each stand-alone app?**

A. No. Stand-alone apps created using Quartus Forth are optimized for speed and size, and contain only the code required by your application.

Q. **Are there royalties associated with distributing my stand-alone apps?**

A. No royalties! Stand-alone apps generated using Quartus Forth are royalty-free, with no run-time libraries, and hence no run-time licensing fees.

Q. **What's the smallest app I can create?**

A. The very smallest app -- no icon, no forms, and only an alert dialog -- is just over 1K in size.

A minimal "Hello World!" application is less than 4K in size, of which the code segment is less than 2K.

Q. **What's the largest app I can create?**

A. Quartus Forth 2.0.0 currently provides for single-segment applications with up to 64K of code each. With the tight factoring and code-reuse that Forth allows and encourages, appreciably complex apps can be written in this space. Future versions of the compiler will permit multi-segment applications.

Q. **Do I need other tools in addition to Quartus Forth?**

A. If your application will make use of static Palm GUI resources (most do), you will need a Palm GUI resource creator/editor. There are several options for this, among them:

- Quartus RsrcEdit for on-board resource creation/editing ($15 USD if bought separately from Quartus Forth)
- Desktop tools:
    - PilRC for script-based desktop resource creation (free)
    - Palm Resource Editor for GUI-based desktop resource creation/editing (free, a component of the Palm PODS 1.1 desktop development system)

Q. **How much does Quartus Forth cost?**

A. Quartus Forth and Quartus RsrcEdit are available together in a special bundle for $99.95 USD.

Purchased separately, Quartus RsrcEdit is $15 USD.

Quartus Forth registration entitles you to receive free updates in the 2.x.x line.

Q. **Is there an evaluation version of Quartus Forth available?**

A. Yes. The evaluation download is freely available, with all the same library sources and documentation that ship with the registered version.

It differs from the registered version in two key ways:

- It cannot create stand-alone apps.
- It has reduced codespace (25% of that available in the registered version).

Q. **I'm a registered user of Quartus Forth 1.2.5. Can I upgrade to Quartus Forth 2.0.0?**

A. Yes. Visit the upgrade page for details.

Q. **Can I access Palm launchcodes?**

A. Yes. Quartus Forth stand-alone apps have full access to Palm launchcodes and associated parameters. In contrast to other Palm development environments, in Quartus Forth apps application globals are always available for all launchcodes.

Q. **Does Quartus Forth support PACE Native Objects (PNOs)?**

A. Yes. Quartus Forth provides an inline ARM assembler for the creation of optimized PNO subroutines for use under Palm Garnet (OS 5) and later.

Q. **Why is Quartus Forth a 16-bit Forth, rather than 32-bit?**

A. There are two key reasons, both relating to the Palm OS architecture:

- Speed: Approximately half of the arguments required by the Palm API are 16-bit values; in a 32-bit Forth, the arguments for nearly all system calls would need to be copied and down-converted before each call. With a 16-bit stack, both 16-bit and 32-bit parameters can be passed directly, which is faster.
- Size: 16-bit instructions result in smaller stand-alone executables.

Q. **Why does Quartus Forth start clean each time it restarts?**

A. This is a deliberate design decision: rather than maintain the previous state of the dictionary, Quartus Forth starts in a known state each time.

Two reasons: first, this simplifies error determination and support; and second, the Quartus Forth dictionary state is commonly intertwined with the Palm GUI state, and there's no reliable way to save and restore the GUI state. Quartus Forth reads a MemoPad file named "\ startup.quartus" each time it starts, so extensions can be included from there.

Typographical Conventions

EMIT	A Forth word. Standard words are presented in all-caps (though Quartus Forth accepts both upper- and lower-case words).
MakePRC	A Quartus Forth extension word currently under discussion.
\ comment	A text comment.
`nilEvent`	A Palm OS constant name.
(*x1* -- *x2*)	A stack comment. Format: (*input* -- *output*)

Other conventions:

- An *r* indicates a floating-point value (short for 'real').
- A trailing '.' indicates a double-cell value.
- A trailing '*?*' indicates a Boolean flag.
- An item in **“”** is parsed from the command-line.
- A leading '*&*' indicates that the item is an address in memory.
- << indicates a logical shift left (e.g. *n*<<*8*)
- **R:** means the stack comment describes the return stack.
- **F:** means the stack comment describes the float stack.

User Guide

How to Buy Quartus Forth

Pricing

Quartus software products are offered for sale through multiple channels. The link for purchasing is:

http://quartus.net/buy

Special developer's bundle: Quartus Forth 2.0.0 + Quartus RsrcEdit for a single reduced price of $99.95 USD – a savings of $15 USD (note: this bundle must be purchased via either PayPal or RegSoft. Bundle not available via Handango).

Quartus Forth 2.0.0 by itself, via Handango: $99.95 USD

Quartus RsrcEdit by itself: $15 USD

Upgrade to Quartus Forth 2.0.0 for registered users of Quartus Forth 1.2.5r: $30 USD

There is a 30% student/educator discount available for Quartus Forth. Provide proof of your current student status to sales@quartus.net, and we'll set you up with the right purchasing codes.

Thank you for your interest in Quartus Forth!

Quartus Forth License Agreement and Limited Warranty

IMPORTANT: Read Before Using the Accompanying Software.

QUARTUS FORTH SOFTWARE DEVELOPMENT ENVIRONMENT SOFTWARE LICENSE AGREEMENT

CAREFULLY READ THE FOLLOWING CONDITIONS AND TERMS OF THIS AGREEMENT BEFORE USING THE ACCOMPANYING SOFTWARE, THE USE OF WHICH IS LICENSED FOR USE ONLY AS SET FORTH BELOW. IF YOU DO NOT AGREE TO THE CONDITIONS AND TERMS OF THIS AGREEMENT, DO NOT USE THE ACCOMPANYING SOFTWARE. USING ANY PART OF THE ACCOMPANYING SOFTWARE INDICATES THAT YOU ACCEPT THESE CONDITIONS AND TERMS.

Neal Bridges (the "Developer"), 65 Scadding Avenue #809, Toronto, Ontario, Canada, M5A 4L1 and the user of Quartus Forth (the "Licensee"), hereby agree as follows:

EVALUATION: The Developer grants to the Licensee a non-exclusive royalty-free license to use the "evaluation" version of the Quartus Forth software free of charge for an unlimited time for the purpose of evaluation for suitability to purchase a registered copy of the Quartus Forth software. All other terms and restrictions set forth in this license agreement are to remain in effect.

LICENSE: The Developer grants to the Licensee purchasing the registered Quartus Forth software (the "Software") a non-exclusive, royalty-free, nontransferable license to use the Software and its documentation in accordance with the terms and conditions of this License Agreement on any Palm Computing® platform product and/or any personal computer, provided that the Software is used only in connection with the development of products by the Licensee for use with the Palm Computing® platform products. This license provides for the use of the Software by a single individual only. Each individual who desires to use the Software must purchase a registered copy.

Except as explicitly set forth below, (i) no license is granted to any rights to copyrights, patents, trademarks, trade secrets, or any other rights in respect to the Software; (ii) no license is granted to the human-readable code of the Software (source code).

The Developer retains title, all rights, interest, and ownership of, in and to the Software and accompanying documentation. The Software and its documentation are owned by the Developer, are protected by Canadian copyright laws and international treaty provisions, and may also be protected by other laws.

The Software contains certain sample source code in the form of library code, example applications and code fragments (both in the source code files and documentation provided hereunder), and may include tutorial applications (collectively, "Sample Source Code"). The Licensee may use the Sample Source Code internally to develop products for Palm Computing® platform products. The Licensee may distribute any such products built with the Sample Source Code, provided that the following copyright notice is included within the source code and in the location of any such product's copyright notice: "Portions copyright © 1998, 2005 Neal Bridges. All rights reserved."

The Developer grants the Licensee the right to develop and distribute products created using the facilities provided by the Software. Such products may be distributed on a royalty-free basis.

The Developer may furnish the Software to the Licensee electronically or on media in machine-readable object code form.

LICENSE RESTRICTIONS: Modification, decompilation, reverse engineering, reverse compiling, or disassembly of the Software is expressly prohibited. Any information obtained during such unlawful reverse engineering and/or decompilation activities, including but not limited to the logic, organization, algorithms and processes of the Software, shall be deemed confidential and proprietary information of the Developer.

Except as explicitly set forth in this License Agreement, the Licensee is not permitted to rent, lease, network, distribute, sublicense, loan, or create derivative works based upon the Software or its documentation in whole or in part, or to use the Software in a time-sharing arrangement or in any other unauthorized manner, unless with the written permission of the Developer.

The Licensee is not permitted to transfer or assign the Software or the rights under this license, without prior written permission of the Developer.

This License Agreement does not grant the Licensee any right to receive enhancements or updates to the Software or accompanying documentation. Updates and enhancements, if available, may be obtained by the Licensee at the Developer’s then-current standard pricing, terms, and conditions. Title, ownership rights, and intellectual property rights to the content accessed through the Software are the property of the applicable content owner and may be protected by applicable copyright or other law. This License Agreement gives the Licensee no rights to such content.

TECHNICAL SUPPORT: For individual Licensees for a period of sixty (60) days from the date of purchase, ONLY LICENSED PURCHASERS OF THE FULL REGISTERED QUARTUS FORTH SOFTWARE PRODUCT will have access to technical support via electronic mail. Upgrades to the Software and its documentation, if any, are not included and may be sold separately.

LIMITED WARRANTY: If the Developer provides the Software on diskettes, he warrants the Software diskettes for a period of sixty (60) days from the date of original purchase from the Developer or an authorized retailer. To obtain warranty service, proof of date of purchase will be required. Any updates to the Software provided by the Developer (which may be provided at the Developer’s sole discretion) shall be governed by the terms of the License Agreement. The Developer’s entire liability and Licensee’s exclusive remedy will be limited to the replacement of the defective diskettes upon return to the place of purchase within the warranty period. The Developer will not be responsible for replacement of any diskette damaged by accident, abuse, or misapplication.

The Developer makes no warranty or representation that the Software will meet the Licensee’s requirements or that it will work in combination with any hardware or software products provided by third parties, or that the operation of the Software will be uninterrupted or error free, or that all defects in the Software will be corrected.

WARRANTIES EXCLUSIVE: TO THE MAXIMUM EXTENT PERMITTED BY LAW, THE FORGOING WARRANTIES AND REMEDIES ARE EXCLUSIVE AND ARE IN LIEU OF ALL OTHER WARRANTIES, TERMS, OR CONDITIONS, EXPRESS OR IMPLIED, EITHER IN FACT OR BY OPERATION OF LAW, STATUTORY OR OTHERWISE, INCLUDING WARRANTIES, TERMS, OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND SATISFACTORY QUALITY. THE DEVELOPER NEITHER ASSUMES NOR AUTHORIZES ANY OTHER PERSON TO ASSUME ANY OTHER LIABILITY IN CONNECTION WITH THE SALE, INSTALLATION, MAINTENANCE OR USE OF HIS PRODUCTS.

Neal Bridges
Toronto, CANADA
Email: info@quartus.net
http://www.quartus.net/

Quartus Forth Installation

Make a backup!

Installation of Quartus Forth is neither dangerous, nor especially difficult. However, be prudent: before beginning installation: be sure you have HotSync'd your device, and that you have a full backup of all of your data. Especially, make an archive (using the Palm Desktop software) of all of your MemoPad memos.

1. **To evaluate Quartus Forth:** Begin by installing the latest version of Quartus Forth from this online link:

   http://quartus.net/products/forth/install

   This is an automated process that will enable the installation of the English-language trial version of Quartus Forth on your device. This installation process also works over-the-air for connected Palm devices. Note that while the evaluation version is English-only, Quartus Forth is available in multiple international versions: English, German, French, Spanish, Swedish, Portuguese, Norwegian, and Malay.

   In addition to installing the evaluation version of Quartus Forth, the installer will copy additional required files to your desktop under one of these paths, depending on your desktop OS:

   ```
   C:\Program Files\Palm\Installed Packages\Quartus Forth\
   ```

   or in the following folder:

   My Documents
   ↳ Palm OS Desktop
   ↳ Installed Packages
   ↳ **Quartus Forth**

   This directory will be referred to as simply “Quartus Forth”.

   In the “Quartus Forth” directory, the following files will appear:

   docs/manual.pdf: This manual, in Adobe Acrobat format.
   docs/constants.pdf: Details on all 6200+ built in constants and structures.
   docs/events.pdf: A chart showing all Palm OS events and their internal details.
   docs/systraps.pdf: Parameters and return values for all integrated Palm OS systraps.
   docs/isodocs.pdf: Documentation required for compliance with the ISO/ANS Forth Standard.

   library/library.csv:
   The library memos in Palm Desktop .CSV format. Use the File->Import option in the Palm Desktop app to install these. Archive and delete any existing memos with the same names first. These memos all have a category of "Quartus" which you can override during import if you wish.

library/libtxt.zip:
The library memos in individual .txt files, with DOS/Windows-style CR+LF line-endings.

library/libtxtu.zip:
The library memos in individual .txt files, with Unix/Mac-style LF-only line-endings.

library/Qrsrc.PRC:
Additional resources used by some library files. At this point only input.txt requires this to be installed.

s**ample**/: A folder containing sample Quartus Forth application sources and resources.

Notes:

If not installing from Windows, the additional files will be extracted to a directory of your choosing.

If you are installing over-the-air, the files will be downloaded and installed during your next HotSync operation.

2. **Installation of the library memos:**

   It is recommended that you set up a category in your MemoPad named "Quartus", and keep only the Quartus Forth library memos in there. Doing so will make it simple to archive and delete them all, and re-import new memos when updates occur.

   Windows users with the Palm Desktop app can use File->Import to directly import the library.csv file into the MemoPad.

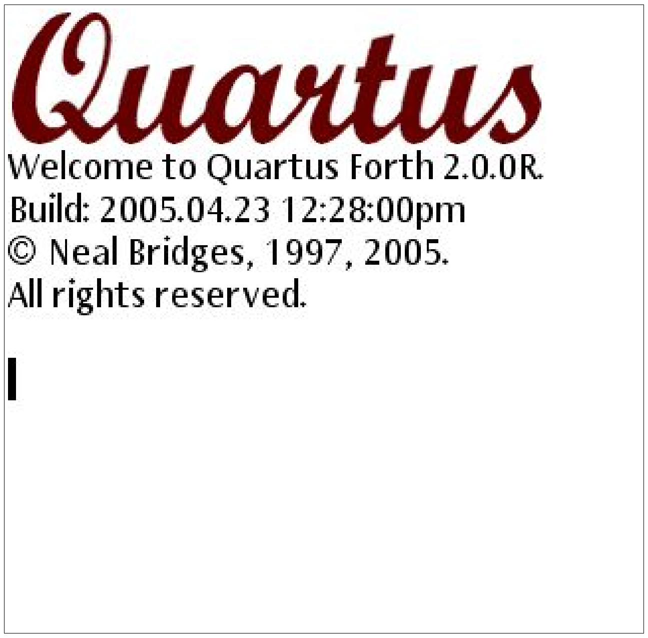
   For users without the Palm Desktop app, the library memos are also provided as individual .txt files in two formats -- with CR+LF line endings (DOS/Windows), and with LF-only endings (Mac/Unix).

   Within the libtxt.zip archives, all files are stored under subdirectories. This is for informational-purposes only; they can all be installed under a single category in the MemoPad on your device. Quartus Forth scans all categories in the MemoPad when searching for filenames.

3. **Registered users**: Once you have purchased the registered version of Quartus Forth, you will install an additional package as specified in the instructions received at registration-time. This package will install the correct international version of Quartus Forth for your handheld device. The registered version of Quartus Forth is provided in English, German, French, Spanish, Swedish, Portuguese, Norwegian, and Malay.

Starting Quartus Forth

Quartus Forth is started by tapping on the Quartus icon in the Palm Launcher. It will show a screen similar to this:



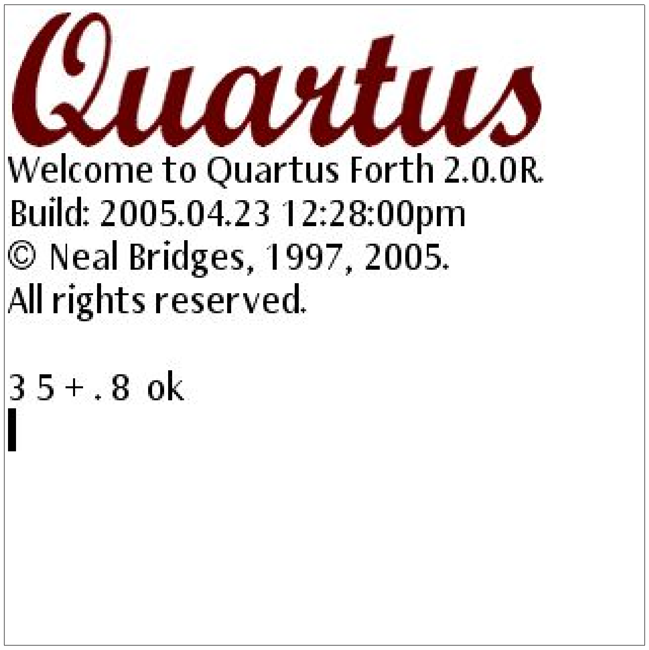
```
Quartus
Welcome to Quartus Forth 2.0.0R.
Build: 2005.04.23 12:28:00pm
© Neal Bridges, 1997, 2005.
All rights reserved.
```

To try out your installation, type this simple sequence:

3 5 + . <*enter*>

Note that <*enter*> is the Graffiti enter stroke. Note also that a space is required between each of the **3**, **5**, **+**, and **.** (called *dot*, which displays the result).

The screen should show this:



```
Quartus
Welcome to Quartus Forth 2.0.0R.
Build: 2005.04.23 12:28:00pm
© Neal Bridges, 1997, 2005.
All rights reserved.

3 5 + . 8  ok
```

As you can see, the console has accepted your input, performed the calculation (adding 3 and 5), and displayed the result of 8.

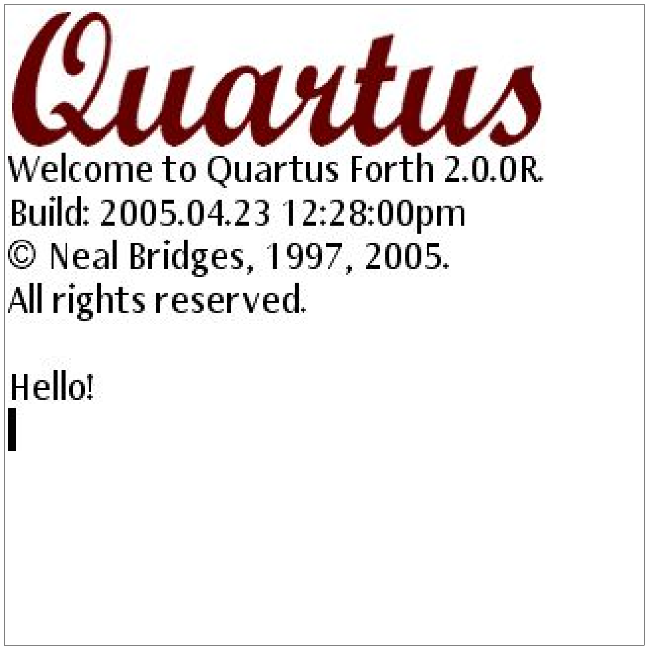
Creating startup.quartus

Each time Quartus Forth starts, it searches for a memo named **startup.quartus** in the built-in MemoPad application. All commands in this file are automatically executed at startup.

To create a **startup.quartus** memo, go to the MemoPad app and create a new memo. The very first two characters in this memo must be a backslash (\) and a space, followed by the filename – in this case, **startup.quartus**. Here is an example **startup.quartus** memo:

```
\ startup.quartus
.( Hello!) cr
```

(This startup.quartus memo will print the word Hello! on the screen each time Quartus Forth starts up. We'll be using this startup.quartus for the remainder of the examples here.)Restarting Quartus Forth with this **startup.quartus** memo will show the following screen:



```
Welcome to Quartus Forth 2.0.0R.
Build: 2005.04.23 12:28:00pm
© Neal Bridges, 1997, 2005.
All rights reserved.

Hello!
```

Console features:

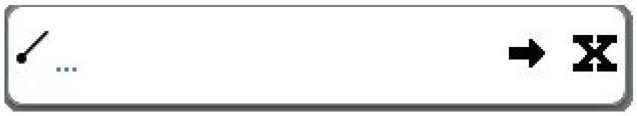
Quartus Forth provides a single-line input field on a scrolling screen, with command-line history accessed via the PageUp and PageDown buttons. Command history is retained between sessions.

On handheld devices with a 5-way selector, the center button doubles as an Enter key.

The Quartus Forth console provides a drop-down menu with the following options:

Program:	Abort (A)	Performs an ABORT back to the console prompt.
	Cold	Restarts Quartus Forth, clearing all memory.
Edit:	Undo (U)	The standard Palm OS editing Undo function.
	Cut (X)	The standard Palm OS editing Cut function.
	Copy (C)	The standard Palm OS editing Copy function.
	Paste (P)	The standard Palm OS editing Paste function.
	Select All (S)	The standard Palm OS editing Select All function.
	Keyboard (K)	Brings up the pop-up keyboard.
	Graffiti (G)	Brings up the Graffiti reference screens.
	Clear History	Clears the Quartus Forth command-line history buffer.
Go:	Last Error (E)	Launch the MemoPad app at the exact point of the last error.
Options:	About	Brings up the Quartus Forth “About” dialog box.

For convenience, on Palm OS 3.5 and later devices, a pop-up menu command-bar is also provided with the Palm OS editing icons, and also Abort (an X) and Last Error (an arrow), as shown here:



Creating “Hello World!”

Let's create a simple source file containing a 'Hello World!' program.

First, create a new memo as follows:

```
\ hello

: go  ." Hello World!" cr ;
```

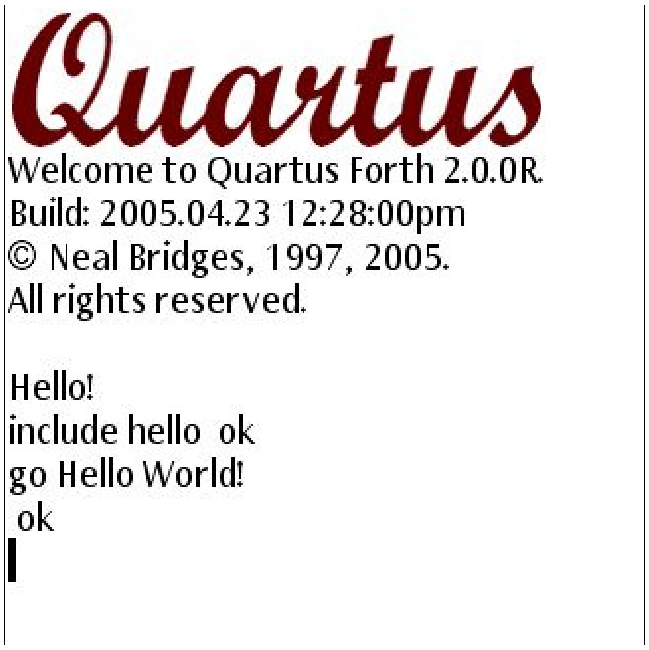
Then, from the Quartus Forth console:

include hello *<enter>*

Then,

go *<enter>*

You'll see the following:



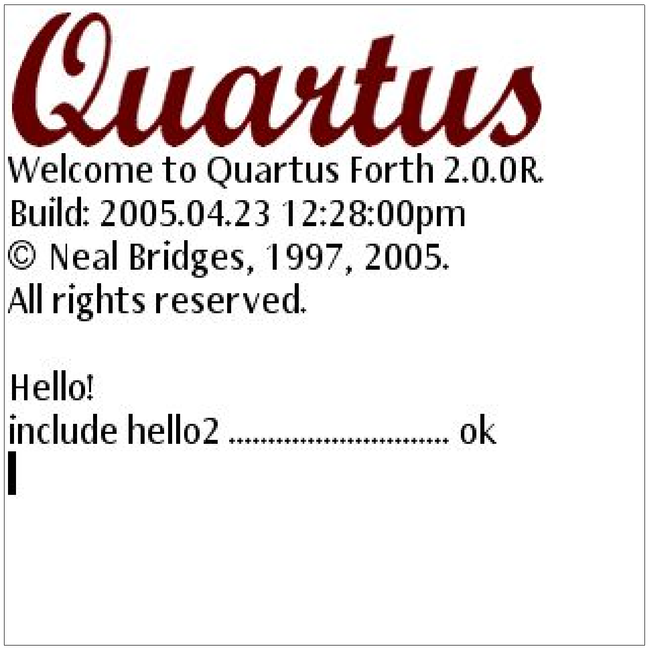
```
Welcome to Quartus Forth 2.0.0R.
Build: 2005.04.23 12:28:00pm
© Neal Bridges, 1997, 2005.
All rights reserved.

Hello!
include hello  ok
go Hello World!
 ok
```

Note that the first “Hello!” comes from the customized startup.quartus we set up earlier.

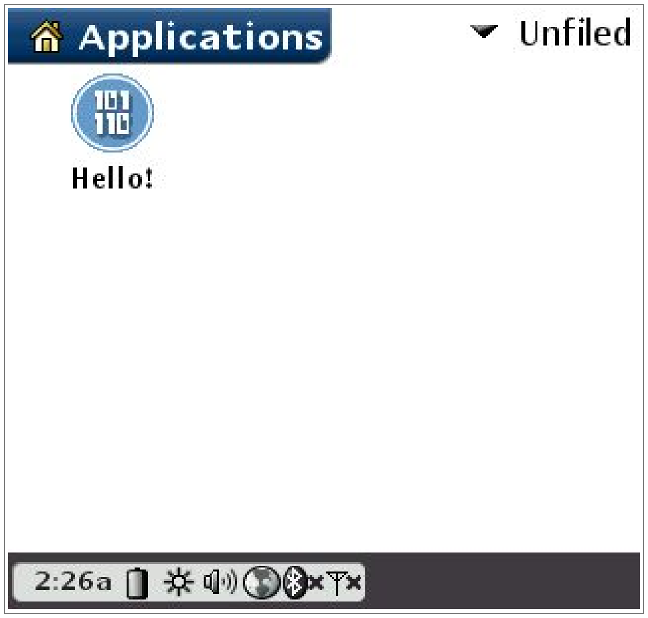
For those using the registered version of Quartus Forth, here is slightly different code to generate a complete, optimized, stand-alone application:

```
\ hello2

: go  MainForm  ." Hello World!"
  begin  key drop  again ;

' go (id) demo MakePRC Hello!
```

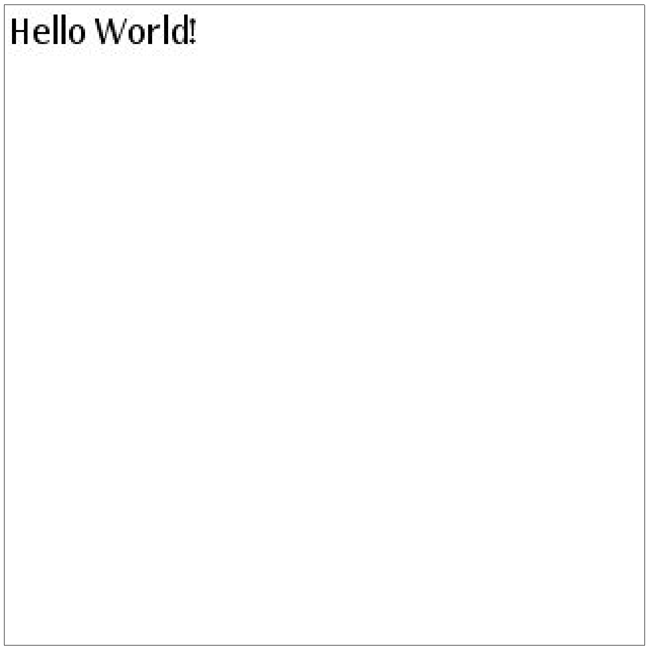
Including **hello2** from the Quartus Forth console gives this:


```
Welcome to Quartus Forth 2.0.0R.
Build: 2005.04.23 12:28:00pm
© Neal Bridges, 1997, 2005.
All rights reserved.

Hello!
include hello2 .............................. ok
```

Then, in the Palm Launcher, you'll see a new application:



Tapping on the Hello! icon will show you the “Hello World!” app in all its simple glory:


Hello World!

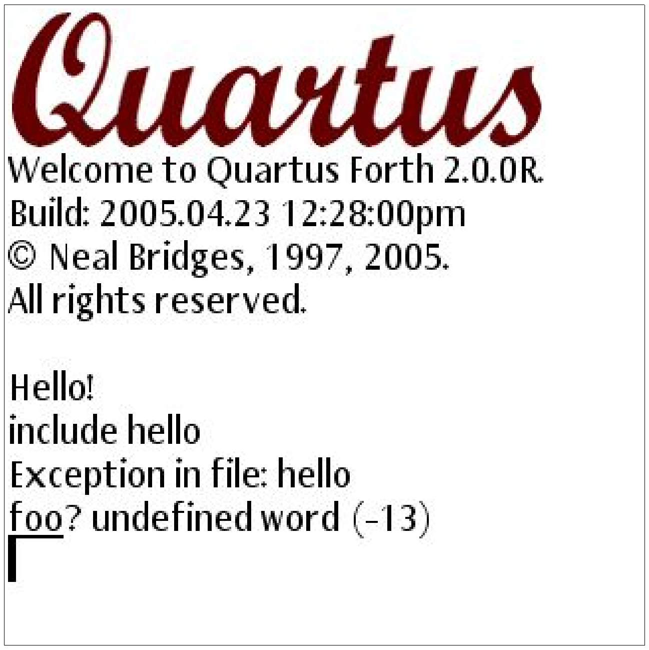
During the next HotSync operation, the `Hello!.prc` stand-alone executable will be copied automatically from your handheld to your computer's hard drive, in the `Palm\`*`username`*`\Backup` folder. This .prc file can be redistributed freely, requires no run-time library, and will run on all Palm devices back to Palm OS version 1.0.

How errors are handled

If an error is encountered while including a file, the compiler will issue an appropriate message at the console. To see the error, use the Last Error function, either from the menu or from the menu command-bar, and you'll be taken directly to the error in the appropriate MemoPad memo, with the location highlighted for editing. Here's the hello example with an error added – 'foo', which isn't a defined word:

```
\ hello
: go  foo  ." Hello World!" ;
```

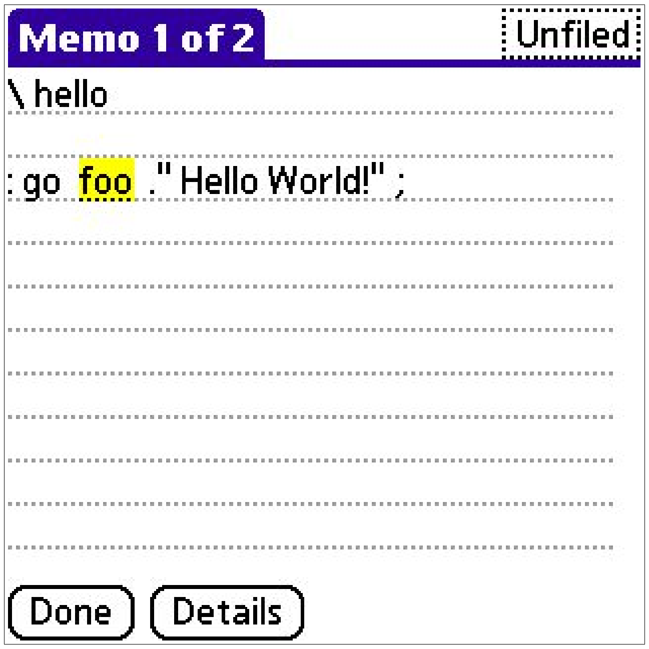
Including this at the console looks like this:



```
Welcome to Quartus Forth 2.0.0R.
Build: 2005.04.23 12:28:00pm
© Neal Bridges, 1997, 2005.
All rights reserved.

Hello!
include hello
Exception in file: hello
foo? undefined word (-13)
```

Note: -13 here is the ANS Standard Forth exception number for the 'undefined word' error. Note also that the first “Hello!” on the screen is from our custom startup.quartus, from earlier in the manual.

Now, “Last Error” (either from the menu, or from the pop-up menu command-bar) takes you directly to the highlighted error:



```
\ hello

: go  foo ." Hello World!" ;
```

Working with Palm OS Resources

There are at least three ways to create new Palm resource databases:

- Quartus RsrcEdit (on-board the Palm)

  Quartus RsrcEdit is an interactive application for creating and modifying resource databases on-board the Palm itself. Although Quartus RsrcEdit is not specifically written for Palm OS 5, version nnnnn works fine under Palm OS 5 to create and modify all the bread & butter resource types – forms, etc.... For extended resource editing (the creation of large bitmaps, for instance), other tools are available:

- Palm Resource Editor (Windows – other?) Part of PODS 1.1. The Palm Resource Editor is an interactive application for editing XRD (XML Resource Description?) files. XRD files describe Palm resources. More details at:...

  - After creating a new XRD file, or creating one from an existing PRC with the GenerateXRD Wizard, to generate an installable resource PRC from the XRD (from Palm Resource Editor or GenerateXRD Wizard):

    Use PalmRC to generate a TRC (Temporary Resource Container) from the XRD. A TRC contains only the resources in binary form, and cannot be installed directly on a Palm device.

    ```
    palmrc -p 68k my.xrd -o my.trc
    ```

    Use PRCMerge to create an installable resource PRC from the TRC: (notes on -c -t and -n)

    ```
    prcmerge my.trc -o my.prc -c p4ap -t rsrc -n MyResources
    ```

  Install this on your Palm device using PalmOne Quick Install.

- PilRC under Windows, Mac, or Linux/BSD

  - To generate a resource PRC from an RCP (PilRC 3.2):

    ```
    pilrc -ro my.rcp my.prc
    ```

  Install the PRC on your Palm device using PalmOne Quick Install.

- To extract resource descriptions from an existing PRC:

  Option A: Use GenerateXRD Wizard to convert resources from a PRC into an XRD and associated files. The XRD can then be edited using the Palm Resource Editor and then recreated as a PRC for installation on the Palm.

  Option B: PRCExplorer can create a PilRC .RCP file from an existing PRC (File->Save Source option). The RCP can be edited with any text editor and then recreated with PilRC as a PRC for installation on the Palm.

Sample Application

The Sample folder under the “Quartus Forth” folder contains the complete sources of a Quartus Forth app called “Duco”. Duco is a four-function calculator with a twist: it operates in Roman numerals.

The files provided are as follows:

duco/

duco.prc:	The complete compiled application.
ducorsrc.prc:	The pre-compiled resources for the application.

Two options for generating the resources for the application:

ducorsrc.xrd:	The XRD file for re-compiling the resources, for use with the Palm Resource Editor.
ducorsrc.rcp:	The RCP file for re-compiling the resources, for use with PilRC.

ducolarge.bmp:	The bitmap for the large application icon.
ducosmall.bmp:	The bitmap for the small application icon.

source/

duco.txt:	The main application source file.
make-duco.txt:	A short source file that builds the stand-alone app.
roman.txt:	A library file for manipulating Roman numbers.
duco.csv:	All three source files as a Palm Desktop .csv file for easy import.

Compiling the sample app Duco

These instructions assume you've already installed Quartus Forth and its associated library files.

1. Install the three source files (`duco.txt`, `make-duco.txt`, and `roman.txt`) in your Palm MemoPad. This can be done either by importing the `duco.csv` file via the Windows Palm Desktop application, or by copy & paste from the three text files on platforms not supporting . csv import.
2. Install the resources on your handheld. They are provided already-compiled as `ducorsrc.prc`, which can be installed directly using the Palm Quick Install. Should you wish to rebuild the resources, there are two options for re-generating the resources. Here's the command for PilRC (the Palm Resource Editor usage is left as an exercise for the student):

   ```
   pilrc -ro -creator Duco -type rsrc -name "Duco Resources" ducorsrc.rcp
   ```

   The command above generates `ducorsrc.ro`. Rename this to `ducorsrc.prc` and install on your Palm using PalmOne Quick Install.

3. From the Quartus Forth console, enter

   include duco *<enter>*

   This should compile without error. Enter

   go *<enter>*

   This will start the application, which will run normally within the Quartus Forth console until you exit.

4. With the registered version, you can generate a stand-alone version of Duco – this will recreate the `duco.prc` provided in the Sample/Duco folder. To do this, re-start Quartus Forth, and enter

   include make-duco *<enter>*

   This should compile without error and generate a stand-alone app, which will appear on the Palm Launcher screen, and subsequently be transferred to your desktop computer during the next HotSync.

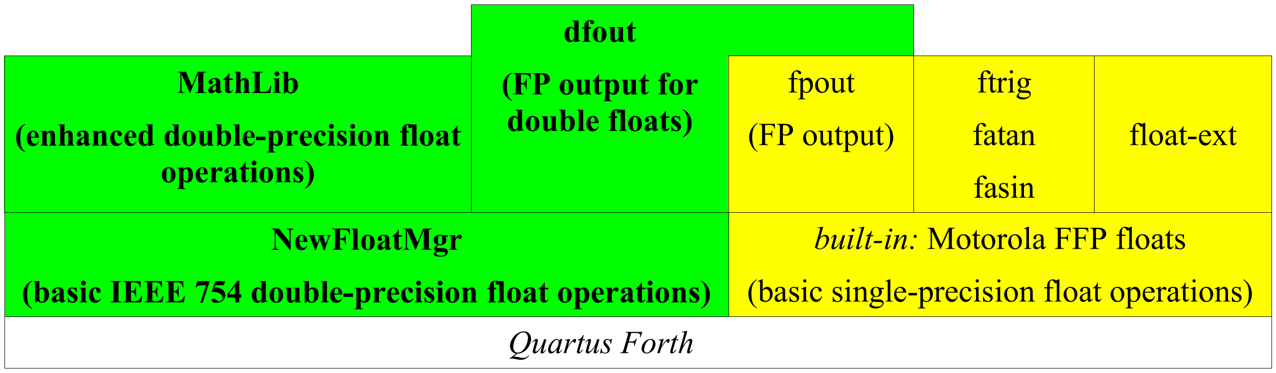
Working with Floating-Point Values

Should you wish to work with floating-point values, Quartus Forth offers a wealth of options:

- Motorola FFP single-precision floats that work on all Palm OS versions (Palm OS 1.0 and up)
- Single- and double-precision IEEE 754 floats that work on Palm OS 2.0 and later
- MathLib access for additional double-precision operations (Palm OS 2.0 and later)

(Note: Palm OS 1 offered rudimentary floating-point support of its own (called *FloatMgr*) with many problems and limitations.)

The following diagram illustrates the various levels of floating-point support available to you as a Quartus Forth developer:


dfout
(FP output for double floats)
MathLib
(enhanced double-precision float operations)
fpout
(FP output)
ftrig
fatan
fasin
float-ext
NewFloatMgr
(basic IEEE 754 double-precision float operations)
built-in: Motorola FFP floats
(basic single-precision float operations)
Quartus Forth

Legend:

🟩 Palm OS 2 and up 🟨 Palm OS 1 and up

The Quartus Forth built-in floating-point format is known as "Motorola Fast Floating Point (FFP)". It's a single-precision format, fast, and accurate within its range; it has a small footprint, and works on any Palm all the way back to version Palm OS 1 devices. It is limited to six or seven places of precision.

Each FFP float is 32 bits in size. Quartus Forth provides a separate float stack that holds 8 FFP values.

Example:

```
\ MyApp

: go
  MainForm
  355e 113e f/ fs.
  begin  ekey drop  again ;
```

This displays:

```
0.31415929E1
```

As the example shows, floats are entered by using 'e' exponent notation. If the exponent is 0, it can be omitted (as shown in the example above).

At startup with no library modules loaded, Quartus Forth provides:

- A floating-point stack with room for 8 FFP floats
- Automatic conversion of literal floats encountered in source to FFP format

As with all Standard Forths, floating-point values are differentiated from integers because they contain an 'e', for example:

```
3.14159e
-112E-4
```

Standard words:

- **F+ F- F* F/ FSQRT FLOOR FROUND FNEGATE FABS**
- **F< F0< F0= FMIN FMAX**
- **F@ F! F>D D>F**
- **FS.**
  FLOATS FLOAT+ FALIGN FALIGNED FROT FDUP FDROP FSWAP FOVER FDEPTH
- **FCONSTANT FVARIABLE FLITERAL >FLOAT**
- **SET-PRECISION PRECISION**

Quartus Forth extensions:

- **fp0 fpdissect**

In the library module float-ext:

Standard words:

- **F.** A rudimentary implementation – use fpout for a full-featured version of F.
- **F~**

Quartus Forth extensions:

- **(f.) -frot #trailing0 places set-places**

In the library module ftrig:

Standard words:

- **FCOS FSIN**

Quartus Forth extensions:

- **fsgn**

In the library modules fasin and fatan:

Standard words:

- **FASIN FATAN**

Simple floating-point output is provided by the built-in **FS.** and by **F.** in float-ext.

For fancier output options, use the fpout module. This provides:

Standard words:

- **F. FE. FS.**

Quartus Forth extensions:

- **g. g.r f.r fe.r fs.r**
- **(fs.) (fe.) (f.) (g.)**

fpout can be re-vectored work for more than one type of float. To use these output words on FFP floats in a stand-alone app, it is necessary to have output-is-f in your initialization routine, before using any of the output words.

Example:

```
\ MyApp
```

```
needs fpout

: go
  MainForm
  output-is-f
  355e 113e f/ f.
  begin  ekey drop  again ;
```

This displays:

```
3.1415929
```

Double-Precision Floating-Point

For higher-precision floating-point operations, NewFloatMgr provides access to the IEEE-754 single- and double-precision floating-point support built into Palm OS 2.0 and later. NewFloatMgr is based on the excellent work of Chapman Flack.

These words operate on 64-bit double-precision floats (four cells each) that reside on the data stack, rather than the floating-point stack.

Use: **needs NewFloatMgr**

Example:

```
\ MyApp

needs NewFloatMgr

: go
  MainForm
  (dfloat) 355  (dfloat) 113  df/  dfs.
  begin  ekey drop  again ;
```

This displays:

```
3.1415929e00
```

As the above example shows, double floats are entered using **(dfloat)**. **(dfloat)** does not require an 'e' in the number, but will accept it if present.

Note that although the double-float calculation in the example is carried to at least 15 digits of precision, the default **DFS.** provided by NewFloatMgr displays only 8 significant digits.

NewFloatMgr provides the following words:

For entering literal floating-point values:

(dfloat)

(sfloat)

For floating-point conversion:

DF>D	Like Standard **F>D**
DF>SF	
D>DF	Like Standard **D>F**
D>SF	Like Standard **D>F**
SF>D	Like Standard **F>D**
SF>DF	
SF>F	
F>SF	

Double-Precision Float words:

DF!	Standard
DF@	Standard
DFLOAT+	Standard
DFLOATS	Standard
DFALIGN	Standard **FALIGN**
DFALIGNED	Like Standard **FALIGNED**
DF*	Like Standard **F***
DF+	Like Standard **F+**
DF-	Like Standard **F-**
DF/	Like Standard **F/**
DFABS	Like Standard **FABS**
DFNEGATE	Like Standard **FNEGATE**
DF<	Like Standard **F<**
DF<=	
DF<>	
DF=	Like Standard **F=**
DF>	
DF>=	
DFMAX	Like Standard **FMAX**
DFMIN	Like Standard **FMIN**
DF!DF	
DF@DF	
DFDROP	Like Standard **FDROP**
DFDUP	Like Standard **FDUP**
DFOVER	Like Standard **FOVER**
DFROT	Like Standard **FROT**
DFSWAP	Like Standard **FSWAP**
DF,	Like Standard **,**
DFVARIABLE	Like Standard **FVARIABLE**
DFCONSTANT	Like Standard **FCONSTANT**

DFLITERAL	Like Standard **FLITERAL**
(DFS.)	Implementation factor of **DFS.**
DFS.	Like Standard **FS.** -- different format, uses FplFToA

Single-Precision Float Words:

SF!	Standard
SF@	Standard
SFLOAT+	Standard
SFLOATS	Standard
SFALIGN	Standard
SFALIGNED	Standard

SF*	Like Standard **F***
SF+	Like Standard **F+**
SF-	Like Standard **F-**
SF/	Like Standard **F/**
SFABS	Like Standard **FABS**
SFNEGATE	Like Standard **FNEGATE**
SF<	Like Standard **F<**
SF<=	
SF<>	
SF=	Like Standard **F=**
SF>	
SF>=	
SFMAX	Like Standard **FMAX**
SFMIN	Like Standard **FMIN**
SF!SF	
SF@SF	
SF,	
SFDROP	Like Standard **FDROP**
SFDUP	Like Standard **FDUP**
SFOVER	Like Standard **FOVER**
SFROT	Like Standard **FROT**
SFSWAP	Like Standard **FSWAP**
SFCONSTANT	Like Standard **FCONSTANT**
SFLITERAL	Like Standard **FLITERAL**
SFVARIABLE	Like Standard **FVARIABLE**
(SFS.)	Implementation factor of **SFS.**
SFS.	Like Standard **FS.** -- different format, uses FplFToA

Palm OS floating-point routines:

FlpAToF
FlpBase10Info
FlpCorrectedAdd
FlpCorrectedSub
FlpFToA
FlpVersion

Miscellaneous words:

```
(fpcheck)
4>R
4R>
```

Floating-Point Output:

The **DFS.** provided by NewFloatMgr does not output in quite the same format as the Standard **FS.** does; it piggybacks on the Palm OS routine **FlpFtoA**.

For fancier output of double floats, use dfout.

This makes use of the re-vectorable library file fpout, and provides:

Standard words:

- **DFS.**

Quartus Forth extensions:

- **dfe.r dfe. df.r df. dg.r dg.**
- **(df.) (dfe.) (dg.)**

Example:

```
\ MyApp

needs NewFloatMgr
needs dfout

: go
  MainForm
  (dfloat) 355  (dfloat) 113  df/  df.
  begin  ekey drop  again ;
```

This displays:

```
3.141592920353983
```

For fancier functions, the shared library MathLib is available.

MathLib is a library of mathematical functions that operate on IEEE 754 double-precision floating-point values. The MathLib support in Quartus Forth is based on Chapman Flack's original implementation.

Mathlib requires Palm OS 2.0 or later, and also requires that the freely-available MathLib library be installed on your handheld device. MathLib is available from http://www.radiks.net/~rhuebner/mathlib.html.

To use MathLib functions, it is necessary to have **do-ini** in your initialization routine.

Example:

```
\ MyApp

needs MathLib
needs dfout

: go
  MainForm
  do-ini
  (dfloat) 2 dfsqrt df.
  begin  ekey drop  again ;
```

This displays:

```
1.414213562373095
```

MathLib routines named to match their Standard Forth equivalents:

DF**	Like Standard **F****
DFACOS	Like Standard **FACOS**
DFACOSH	Like Standard **FACOSH**
DFASIN	Like Standard **FASIN**
DFASINH	Like Standard **FASINH**
DFATAN	Like Standard **FATAN**
DFATAN2	Like Standard **FATAN2**
DFATANH	Like Standard **FATANH**
DFCOS	Like Standard **DFCOS**
DFCOSH	Like Standard **DFCOSH**
DFEXP	Like Standard **DFEXP**
DFEXPM1	Like Standard **DFEXPM1**
DFLN	Like Standard **FLN**
DFLNP1	Like Standard **FLNP1**
DFLOG	Like Standard **FLOG**
DFLOOR	Like Standard **FLOOR**
DFROUND	Like Standard **DFROUND**
DFSIN	Like Standard **FSIN**
DFSINCOS	Like Standard **FSINCOS**
DFSINH	Like Standard **FSINH**
DFSQRT	Like Standard **FSQRT**
DFTAN	Like Standard **FTAN**
DFTANH	Like Standard **FTANH**

Other MathLib routines, not renamed:

MthLabs

MthLcbrt
MthLceil
MthLcopysign
MthLdrem
MthLfinite
MthLfmod
MthLfrexp
MthLhypot
MthLilogb
MthLisinf
MthLisnan
MthLldexp
MthLlog2
MthLlogb
MthLmodf
MthLnextafter
MthLremainder
MthLrint
MthLscalb
MthLscalbn
MthLsignbit
MthLsignificand
MthLtrunc

Note: At the Quartus console, MathLib functions may be used as soon as the MathLib memo has been included (via needs). A program that will be made standalone must call **do-ini** once early (usually from go) before the functions may be used; do-ini will take care of initializing MathLib and any other modules that use the inifini mechanisms. Both standalone and at the console, inifini will take care of closing the library (and any cleanup needed by other modules using inifini) upon any of the usual ways to exit or switch away from the program.

Only programs that use PalmOS API directly to switch the current application may need to call **do-fini** explicitly. In the case of MathLib, the only downside of failing to close the library is that its reference count will be incorrect, preventing such operations as deleting the library until after a soft reset.

External links to other useful information:

- Quartus Handheld Software resources: (http://www.quartus.net):
    - Quartus Discussion Forums (where Quartus Forth developers hang out!) http://www.quartus.net/discus
- Useful sources of Palm development information (From the Development link at http://www.palmos.com, requiring free registration as a Palm developer):
    - Palm OS Companion & Reference
    - Palm OS SDK header files
    - Palm OS Recipes
    - Palm OS Knowledge Base
    - Palm OS Developer Archive

- Debugging Tools: (http://www.palmos.com/dev/dl/)

    - Palm OS Emulator (4.x and earlier) – gremlins support
    - Palm OS Simulator Garnet (5.4) Release
    - Palm OS Simulator Garnet (5.4) Debug – gremlins support
    - Palm OS Simulator Cobalt (6.1) Debug (no gremlins)
    - Palm Reporter – a trace tool that works with the Simulator

- Palm Application Explorer – aka PRC Explorer
  http://www.palmgear.com/index.cfm?fuseaction=software.showsoftware&prodid=40542

- For use with the Emulator - Palm Debuffer http://sourceforge.net/projects/debuffer/

- Quartus RsrcEdit http://quartus.net/products/rsrcedit

- Palm Resource Editor and PalmRc, GenerateXRD (and GenerateXRD Wizard) No-cost download (registration as a developer is required): http://www.palmos.com/dev/dl

- PilRC (open-source) http://sourceforge.net/projects/pilrc/

  You can download a Mac OS X binary of PilRC 3.2 from: http://www.gymace.co.uk/pilrc3_2

Reference Manual

Quartus Forth Wordsets: General

Defer Wordset

In general terms, to defer something is to put it off until a later time. Deferred words are no different. A deferred word is named when it is first created, but its action is assigned at a later time. Quartus Forth provides a Defer wordset to manage deferred words.

- **defer** ("***name***" --) Creates a new deferred word called *name*.
- **is** (*xt* "***name***" --) Assigns an action to ('re-vector') the deferred word *name*.
- **action-of** ("***name***" -- *xt*) Retrieves the xt assigned to the deferred word *name*.
- **defer@** (xt_1 -- xt_2) Retrieves the xt assigned to a deferred word specified by xt_1.
- **defer!** (xt_2 xt_1 --) Stores xt_2 in the deferred word specified by xt_1.
- **deferred?** (*xt – 0|1|2*) Determines if a given xt is deferred or not.

Put succinctly, **defer** creates a deferred word, and **is** assigns an action to it.

Example:

```
defer myword

: definition1  ." This is the primary action." ;
: definition2  ." This is an alternate action."

' definition1 is myword

myword <enter> This is the primary action. ok

' definition2 is myword

myword <enter> This is an alternate action. ok
```

By default, a new deferred word performs **ABORT** until another action is assigned to it.

Assigning an action to a deferred word automatically changes the action of that word in any definition that uses it, whether that definition comes before, or after, the assignment of the action.

Other helper words in the wordset:

- **action-of** is the reverse of **is**, and returns the xt of the action currently assigned to a deferred word.
- **defer@** and **defer!** are provided for circumstances where it might be necessary to access a deferred word by xt, rather than by name.
- **deferred?** returns `1` if a given xt is a kernel-deferred word, `2` if it is a user-deferred word, and `0` if it is not a deferred word.

Notes:

- The action of a user-defined deferred word that is used in a stand-alone app *must* be assigned by the app at run-time before the deferred word is first used, or you'll get unexpected results. Any initialization of user-deferred words performed at compile-time is *not* carried over into a stand-alone app.

- For reasons of simplicity and efficiency in the design of the compiler, the deferred word implementation in Quartus Forth is a hybrid one; there are two kinds of deferred words, those provided by the kernel, and those defined by you. The difference is transparent at the console. Both kinds can, where appropriate, be used in your stand-alone apps. However, only those defined by you can be directly re-vectored by your stand-alone app. In contrast, deferred words exported from the kernel will, in your stand-alone app, carry with them the last action assigned to them at compile-time, and cannot be re-vectored by the app – at least not directly.

All is not lost, however. Should you need to re-vector a kernel-deferred word from within your stand-alone app, you still can. The library module `redefer` is provided for this purpose; not only does it allow kernel-deferred words to be re-vectored at run-time, but also can make many kernel words into deferred words, even if they are not deferred to start with. The following illustrates the use of `redefer`:

> **needs redefer**
> ...
> **redefer** *<name>*

the word *<name>* is now a user-deferred word.

There are two situations where **redefer** is needed: first, to defer a kernel word that is not already deferred, and second, to defer kernel words (deferred already or not) that you wish to re-vector within a stand-alone app.

Events Wordset

Event-handling in a Palm Powered application involves a fairly complex ritual of retrieving events from the system event queue, and passing each event through a series of Palm OS handler routines in a certain preordained order. This is an unchanging aspect of all Palm apps, and so Quartus Forth encapsulates all of the required steps in a few easy-to-use words; all your app has to do is request events, and respond to them as they appear on the stack.

Via **EKEY** and related words, all available events are visible to a Quartus Forth app – normally excluding any pen and key events that the Palm OS has already handled itself. This keeps your app from receiving unwanted pen events from menus and the silkscreen area, unwanted key events from the menubar, etc. Should you wish to receive even those events the Palm OS handles itself, you can – see the "Lower-Level Words" section, below.

Implementation details for Standard event-handling words:

- **EKEY** (-- *event-type*) Retrieves the next event from the Palm OS event queue, returning the type of the event on the stack. The event itself is recorded in a static buffer named **event**. If no event is available on the event queue within a half-second interval, **EKEY** returns a zero (`nilEvent`). Equivalent to:

```
: EKEY  50. (ekey) ;
```

- **EKEY>CHAR** (*event-type -- event-type false | char true*) Converts the most recently-received event into a character, if applicable. Quartus Forth also returns the character value `11` for the PageUp key, and `12` for the PageDown key.
- **KEY** (-- *char*) Built using **EKEY>CHAR**, and thus also returns the `PageUp` and `PageDown` characters.

Quartus Forth Extensions:

- **(ekey)** (*timeout. -- event-type*) Like **EKEY**, but takes a double-cell timeout value measured in Palm OS *ticks*, each 1/100th of a second. A timeout of `-1.` makes it wait until an event occurs. A timeout of `0.` means no wait. If no event is available on the event queue before the specified timeout, it returns a zero (`nilEvent`). (In later Palm OS versions, a `nilEvent` occurs every few seconds even with a timeout of `-1.`). Equivalent to:

```
: (ekey)
   begin
      2dup get-event  handle-event
   0=  swap  keyDownEvent >  or   until
   2drop  event @ ;
```

- **event** (-- *addr*) Returns the dataspace address of the event buffer, which is an `EventType` structure that holds the details of the most recent event returned by **(ekey)**, **EKEY**, or **KEY**.
- **event-id** (-- *id*) Returns the ID of the form/control/field/list/etc. associated with the latest received event (applicable for more than 80% of event types – excluding those that don't have an associated ID, notably `pen`, `key`, and `winEnter`/`winExit` events). Equivalent to:

```
: event-id  event EventType.data + @ ;
```

This is provided for convenience, because so many event types contain an ID. For details of other types of events, you'll need to interrogate the fields of the event buffer directly. See the "Palm OS 5 Events" chart for further details.

This functionality was formerly provided by the now-deprecated **itemid** (*&event. -- item-id*) (previously in the Events library file).

Lower-level words for managing events:

Unless you require access to pen and key events which the Palm OS normally handles itself, or you need to intercept events before the Palm OS is given an opportunity to handle them, you'll never need these words, but they're present if needed:

- **get-event** (*ticks. -- event-type*) Like **(ekey)**, but it only retrieves an event, it does not handle it. Calls the Palm OS routine `EvtGetEvent`.
- **handle-event** (-- *handled?*) Allows all appropriate Palm OS event-handling routines an opportunity to handle the most recently received event. Returns a flag on the stack indicating whether or not the event was handled by one of those routines.

Should your app need to see every event with no exclusions, use **get-event** and **handle-event** as per the following outline:

```
: all-events
    begin
      -1. get-event  ( -- event-type )
        ... each event is is available here, with no exclusions ...
      handle-event ( -- handled? )  drop
    again ;
```

Deprecated/obsolete words:

- **(handle-event)** (--) Like **handle-event**, but the handled/not-handled status is *not* returned on the stack, and must be accessed via the **d0.L** from the Systrap wordset. This word was named **HandleEvent** in previous versions, and is now renamed to indicate that it is an implementation-factor of **handle-event**. Use **handle-event** instead.
- **eventhandler** (--) An old vectoring mechanism for handling Palm events, dating back to PilotFORTH; now unnecessary and thus made obsolete. Throws `-32` "obsolescent feature". Use **EKEY** and related words instead.

See also:

- Library module Events

ID Wordset

Palm OS development frequently requires the use of four-character identifiers known (depending on context) as types, or creator IDs. For instance, each Palm app is identified by its own unique four-character creator ID, and each resource in a Palm resource database is identified by a specific four-character *type* that indicates the purpose of that resource.

Any four-character identifier is equivalent to a specific 32-bit double-cell value. For example, the four-character identifier `psys` is equivalent to the 32-bit double-cell value 1886615923. Clearly it's easier to refer to identifiers by name rather than value, and so Quartus Forth provides an ID wordset to facilitate this.

- **(id)** (*"xxxx"* -- *d.*) Converts the four characters ***xxxx*** to a 32-bit double-cell value.
- **[id]** ("***xxxx***" --)(**run-time:** -- *d.*) Same as **(id)**, but for use in a definition.
- **id** ("***xxxx***" --) Creates a **2CONSTANT** named ***xxxx*** with the 32-bit value of ***xxxx***.

Example:

```
(id) xxxx 2CONSTANT creator

: go
  MainForm ... [id] psys ... FtrGet ... ;

' go creator MakePRC MyApp
```

(id) reads the next four-character word in the input line and converts it to a 32-bit double-cell value on the stack. **[id]** does the same, but for use within a definition (similar in function to the Standard Core words **CHAR** and **[CHAR]**).

id creates a double-cell constant with the same name as the four-character id itself.

Example:

```
id psys
```

psys can then be used in place of **[id] psys** in a definition, or in place of **(id) psys** outside of a definition.

Notes:

- "ID" means more than one thing in the Palm world. A "creator ID" is a four-letter 32-bit value, but the term "ID" is also used when referring to the 16-bit numeric identifiers of Palm GUI objects – e.g. "form ID", "field ID", "menu ID", etc. The context usually makes it easy to differentiate.
- **id xxxx** is equivalent to **(id) xxxx 2CONSTANT xxxx**

See also:

- The Stand-Alone Wordset section

Interpreter Wordset

Quartus Forth provides several words that extend the Forth interpreter.

Quartus Forth Extensions:

- **(binary)** (*"..."* --) Switches the value of **BASE** to `2`, and evaluates the next word in the input buffer. After evaluating, **BASE** is restored to its previous value. This is an immediate word.
- **(octal)** (*"..."* --) As **(binary)**, but using **BASE** `8`. This is an immediate word.
- **(decimal)** (*"..."* --) As **(binary)**, but using **BASE** `10`. This is an immediate word.
- **(hex)** (*"..."* --) As **(binary)**, but using **BASE** `16`. This is an immediate word.
- **(float)** (*"..."* --) Parses the next word in the input buffer and converts it to a Quartus Forth floating-point value. If compiling, the floating-point value is compiled as an **FLITERAL**. This is an immediate word.
- **(radix)** (*base "..."* --) A factor of **(binary)**, **(octal)**, **(decimal)**, and **(hex)**. Sets **BASE** to the specified value, evaluates the next word in the input buffer, and restores **BASE** to its previous value.
- **skip** (*char* --) Skips leading characters in the input buffer that match the specified character.
- **parse-word** (*char "...<char>"* -- *c-addr u*) Skips leading characters in the input buffer that match the specified character, and then returns the next string from the input buffer delimited by the specified character.

For advanced developers:

- **unknown** (--) This is a deferred word called by the Quartus Forth interpreter when a specified word be found in the dictionary when interpreting or compiling source. By default, it first tries to convert the word to a number; if that fails, it searches for the word in the integrated database of systraps. If that fails, it reports `-13` ("undefined word").

  If you choose to re-vector it, your code will receive a (*c-addr u*) on the stack; you can then attempt to resolve the unknown word and handle it appropriately. If you cannot, pass the *c-addr u* on to the original vector of **unknown**.

Memory-Access Wordset

The Palm OS provides two kinds of read/write memory, called heaps: *dynamic*, and *storage*. Both can be read without restriction, but only memory allocated from the dynamic heap by and for your application can be written to directly. The Palm OS protects the storage heap from accidental overwrite; it must be accessed for writing via Palm OS routines such as `DmWrite`.

The region of the dynamic heap allocated by Quartus Forth and available to your apps at run-time is a 32K region known as *dataspace*. Dataspace is freely accessible for reading and writing via single-cell 16-bit addresses, each relative to a 32-bit base pointer established by the Palm OS when it launches your app. These are known as dataspace-relative addresses.

At the Quartus Forth console, and while compiling source, you are able to read and write *codespace*, which is a specific region of the storage heap allocated by the console at startup as working space for the duration of the Quartus Forth session. Codespace is where apps are compiled. Once an app has been made stand-alone, it cannot write to its own codespace.

Quartus Forth provides the Standard words for memory access, with extensions to allow you to access all required Palm and Forth memory spaces.

Implementation details for Standard words:

These eight Standard words act on 16-bit dataspace-relative addresses.

- **@ ! C@ C! 2@ 2! F@ F!**

Dataspace in a stand-alone app cannot be resized. Thus, the following seven Standard words can be used at the Quartus Forth console, and while compiling source, but not in stand-alone apps:

- **HERE , C, ALLOT ALIGN FALIGN UNUSED**

When the Quartus Forth console starts, in order to minimize dynamic memory use, the allocated dataspace region is less than `1K`, and **UNUSED** reports less than `0.5K`. The words that move the dataspace pointer (**, C , ALLOT**) automatically resize the dataspace region as required, plus a small margin; **UNUSED** will, therefore, always report less than `0.5K` available. In total, for any given app/console session, **HERE** cannot exceed `32K`.

Quartus Forth Extensions:

For converting dataspace addresses from 16-bit relative to 32-bit absolute:

- ***>abs*** (*addr -- addr.*) Converts a single-cell 16-bit dataspace-relative address to a double-cell 32-bit absolute dataspace address.
- **>rel** (*addr. -- addr*) Converts a double-cell 32-bit absolute dataspace address to a single-cell 16-bit dataspace-relative address.

>abs and **>rel** are used within your code to convert addresses in dataspace to the 32-bit absolute address format expected by certain Palm OS routines, and to translate the addresses returned by certain Palm OS routines back into dataspace-relative addresses.

For accessing 32-bit absolute addresses:

The following six words work on double-cell 32-bit absolute addresses. The words that fetch data (**@a c@a 2@a**) can be used to read memory anywhere in Palm address space, whether in the dynamic or the storage heaps, or in ROM The words that store data (**!a c!a 2!a**) can only be used to write to addresses in dynamic memory.

These words are named as per their Standard counterparts, but with an **a** suffix.

- **@a** (*addr. -- x*) Fetch a 16-bit value from a 32-bit absolute address.
- **!a** (*x addr. --*) Store a 16-bit value at a double-cell 32-bit absolute address.
- **c@a** (*addr. -- char*) Fetch an 8-bit character from a 32-bit absolute address.
- **c!a** (*char addr. --*) Store an 8-bit character at a 32-bit absolute address.
- **2@a** (*addr. --* x_1 x_2) Fetch a double-cell 32-bit value from a 32-bit absolute address.
- **2!a** (x_1 x_2 *addr. --*) Store a double-cell 32-bit value at a 32-bit absolute address.

For accessing 16-bit single-cell addresses in codespace:

Of these words, those that fetch data from codespace (**cs@ csc@**) will work both at the Quartus Forth console and in your stand-alone apps. The words that store data in codespace (**cs! csc!**) and the words that access codespace sequentially (**cshere cs, csc, csunused**) can only be used at the Quartus Forth console, and while compiling source, but not in stand-alone apps.

These words are named as per their Standard counterparts, but with a **cs** prefix.

- **cs@** (*csaddr -- x*) Fetch a 16-bit value from a codespace-relative address.
- **cs!** (*x csaddr --*) Store a 16-bit value at a codespace-relative address.
- **csc@** (*csaddr -- char*) Fetch an 8-bit character from a codespace-relative address.
- **csc!** (*char csaddr --*) Store an 8-bit character at a codespace-relative address.
- **cshere** (*-- cshere*) Return the current value of the codespace pointer. **cshere** acts the same way as the Standard word **HERE**, but in codespace rather than dataspace.
- **cs,** (*x --*) Store a 16-bit value at the current codespace address returned by **cshere**. Increment **cshere** by one cell-width (2 bytes).
- **csc,** (*char --*) Store an 8-bit character at the current codespace address returned by **cshere**. Increment **cshere** by one char-width (1 byte).
- **csunused** (*-- u*) Returns an unsigned value, the amount of codespace still available.

For accessing the data stack and return stack directly:

In some circumstances, it's useful to know the 32-bit addresses of the pointers to the data and return stacks, and to be able to set them. These words are provided for advanced use.

- **sp@** (*-- addr.*) Returns the double-cell 32-bit absolute address of the top-most item on the data stack.
- **sp!** (*addr. --*) Sets the data stack pointer to the double-cell 32-bit absolute address of the top-most item on the data stack.
- **rp@** (*-- addr.*) Returns the double-cell 32-bit absolute address of the top-most item on the return stack.

- **rp!** (*addr.* --) Sets the return stack pointer to the double-cell 32-bit absolute address of the top-most item on the return stack.

Notes:

- All words that read or write 16-bit values require that the addresses passed to them be aligned on two-byte boundaries; that is, they must be even, rather than odd. The Standard word **ALIGNED** can be used on a 16-bit address to correctly align it on an even boundary. **ALIGN** is the Standard word that aligns the dataspace pointer, **HERE**.

  Note that the PACE environment under Palm OS 5 and later does not enforce the requirement for even-boundary accesses; however, Palm OS 4.x (and older) devices running on actual Motorola DragonBall processors will display a 'fatal error' dialog if you attempt to access a 16-bit value on an odd boundary. If developing under Palm OS 5, before releasing your app be sure to test under the Palm OS Emulator using a Palm OS 4.x (or earlier) ROM in order to detect this type of error.
- Only use **>abs** on addresses that originate in dataspace. Only use **>rel** to convert such addresses back into 16-bit dataspace-relative addresses. Do not use **>rel** to convert any other absolute addresses to 16-bit addresses, as the results will be meaningless and will not result in usable addresses for accessing memory. Likewise, do not use **>abs** on addresses that originate anywhere other than in dataspace, or the results will be equally meaningless.
- The addresses returned by **sp@** and **rp@** are not guaranteed to be in dataspace, and hence **>abs** and **>rel** cannot be used on them.
- The top-most item on the data stack is normally cached in a data register; **SP@** first flushes it to memory before returning the address, and **SP!** re-caches it after setting the address.

Miscellaneous Wordset

These are words that don't fit neatly into other wordsets.

Common-usage words:

The following are non-Standard words that are in common usage in many Forth compilers. Some are discussed in the Appendix of the Standard Forth document.

- **for**/**next** An optimized looping construct. Counts backward from n to `0`. Similar to the Standard **DO**/**LOOP**/**+LOOP** looping constructs, but faster, with a fixed decrement and direction, and taking only one parameter.

  for (*n* --) Begin a **for**/**next** loop.
  next (--) End a **for**/**next** loop.

  Example:

  : ten 10 for i . next ;

  ```
  ten <enter> 9 8 7 6 5 4 3 2 1 0 ok
  ```

- **m/mod** ($ud_1. u_1$ -- u_2 $ud_2.$) Divide unsigned 32-bit dividend ud_1 by unsigned 16-bit divisor u_1 to give a 32-bit quotient ud_2 and 16-bit remainder u_2. Used by the Standard word **#**.
- **enough?** (*n* --) Checks the data stack for at least *n* items; throws `-4` ("stack underflow") if not enough items are present.
- **noop** (--) Does nothing.
- **cold** (--) Resets the Quartus Forth console (or the app, in a stand-alone app); it is the same as exiting and re-entering.
- **alias** (*xt "name"* --) Duplicate the specified xt under a different name. All flags and attributes of the original are reflected by the new word, including the xt. The new word takes no codespace in the dictionary.

Double-Cell Common-Usage Words:

- **dand** (*xd1. -- xd2.*) As the Standard word **AND**, but operating on a double-cell value.
- **dor** (*xd1. -- xd2.*) As the Standard word **OR**, but operating on a double-cell value.
- **dinvert** (*xd1. -- xd2.*) As the Standard word **INVERT**, but operating on a double-cell value.
- **dlshift** (*xd1. n -- xd2*) Shift the double-cell value *xd1.* left by (*n* bits.
- **drshift** (*xd1. n -- xd2.*) Shift the double-cell value *xd1.* right by *n* bits.
- **swapends** (*x1 -- x2*) Convers the single-cell value *x1* from big-endian to little-endian, or from little-endian to big-endian. (Not a double-cell word, but placed here by association with **dswapends**.)
- **dswapends** (*xd1. -- xd2.*) Converts the double-cell value *xd1.* from big-endian to little-endian, or from little-endian to big-endian

Additional miscellaneous words:

- **hash** (*c-addr u -- u.*) Case- and accent-insensitive hash function.
- **shash** (*c-addr u -- u.*) Case-sensitive hash function.
- **registered** (***"registration-key"*** --) Sets your Quartus Forth registration key, which fully activates all the features of Quartus Forth. This belongs in your startup.quartus file.
- **mem** (-- *free. max.*) Compacts the dynamic heap and returns the total number of free bytes in the heap, and the size of the largest free chunk in the heap. Will be deprecated in future versions.
- **about** (--) Displays the Quartus Forth "About" box. Will display any custom `Talt 1000` alert in your app. Equivalent to **: about 1000 FrmAlert drop ;**
- **fp0** (--) The dataspace address of the bottom of the internal floating-point stack.
- **fpdissect** (**F:** *r* --) (-- *sign exponent unsigned-mantissa.*) Dissects an Quartus Forth floating-point value into its component parts.
- **(2>r)** (x_1 x_2 x_3 -- $x3$ **R:** -- x_2 x_1) An optimized factor of the Standard word **2>R**.
- **(2r>)** (x_3 -- x_1 x_2 x_3 **R:** x_2 x_1 --) An optimized factor of the Standard word **2R>**.

- **select**/**xt**/**end-select** An optimized selection control-structure.
  - **select** (*n* --) Begin a **select**/**end-select** structure.
  - **xt** (***"name"*** --) Add a selection to the **select**/**end-select** structure.
  - **end-select** (-- *xt*) End the **select**/**end-select** structure.

  A **select**/**end-select** control-structure takes a selection parameter (0-n) on the stack, and calls the selected **xt**. Be sure that there are at least as many **xt**s in the structure as the highest possible parameter value you pass to **select**.

  Example:

```
: zero  ." Zero" ;
: one  ." One" ;
: two  ." Two" ;

: go
   select
      xt zero
      xt one
      xt two
   end-select ;
```

```
0 go <enter>  Zero  ok
1 go <enter>  One  ok
2 go <enter>  Two  ok
```

Notes:

- Do not place anything other than **xt** statements between **select** and **end-select**.

Module Wordset

Quartus Forth provides support for multiple namespaces in the form of the the Standard Search-Order wordset, but it can be somewhat cumbersome for casual use. Developers frequently want data encapsulation, so there's a desire for easy-to-use modular public/private namespaces. To that end, Quartus Forth provides a Module wordset.

The Module wordset allows for named modules with two scopes, *public* and *private*.

- **module** (***"name"*** -- *module-sys*) Starts a named module.
- **public:** (--) Switches to public scope.
- **private:** (--) Switches to private scope.
- **end-module** (*module-sys* --) Ends a named module.
- **expose-module** (***"name"*** --) Accesses a named module's private words.

Typical use is simple, as per this outline:

```
module <name>
private:
    internal words here
public:
    words for external use here, built using the private words above
end-module
```

You can switch back and forth between **private:** and **public:** as often as required inside a module. At the beginning of a module, the scope is private by default.

A module's private words are normally only visible in the public scope of the same module.

After **end-module**, the module's public words are still visible and accessible, but the private words are no longer visible.

To gain access to the private words of a module from outside of that module:

```
expose-module <name>
    internal words are visible here
previous
```

(**previous** is from the Standard Search-Order wordset; whereas **expose-module** adds the specified module's private wordlist to the search order, **previous** removes it when access to the private words is no longer required.)

The benefits are two-fold: different modules can have identically-named private words without conflict, and module-specific private words are hidden from other modules and application code.

Examples:

```
module foo
public:
    : aaa ." Visible outside of the module." ;
private:
    : bbb ." Not visible outside of the module."
    : ccc ." Also not visible outside of the module."
public:
    \ aaa, bbb and ccc are visible here:
    : ddd aaa bbb ccc ;
end-module

\ aaa and ddd are visible here for use in subsequent definitions.
\ bbb and ccc are not visible.

\ To access the private word 'bbb':

expose-module foo
bbb
previous
```

Modules can be nested:

```
module bar
private:
    : hhh ." Private to the bar module."
public:
    : eee ." Visible outside the module."
    module moo
    private:
        \ hhh and eee are visible here:
        : fff ." Not normally visible outside the moo module."
    public:
        \ hhh, eee, and fff are visible here
        : ggg ." Visible outside the module."
    end-module
\ eee and ggg are visible here, but not hhh or fff
end-module
\ eee and ggg are visible here, but not hhh or fff
```

Any module's private words can be exposed, whether or not the module is nested:

```
expose-module moo
fff
previous
```

Something interesting:

```
module bob
private:
    module alice
    private:
        : xxx ." This is in the private scope of module alice."
    public:
        : yyy ." This is in the private scope of module bob."
    \ yyy is visible here...
end-module
\ ...and here...
end-module
\ ...but not here.
```

Because module **alice** is declared in the private scope of module **bob**, all of **alice's** public words are private to **bob**. **xxx** will not be visible outside of **alice**, and **yyy** will not be visible outside of **bob**.

Notes:

- The *public* scope is the compilation wordlist and search order that is active before the module is declared. Each module's *private* scope is a new wordlist specific to the module, added to the search order for the duration of the module. Quartus Forth supports the creation of up to 112 new wordlists, and hence up to 112 named modules, at any given time.
- Inside a module, the private scope is searched first, followed by the public scope.
- Quartus Forth supports a maximum search order of 16 wordlists; this means modules may potentially be nested 15 deep.
- The Module wordset manipulates the Forth search order. Should you need to make changes to the search order inside a module, the search order should be restored to its former state before **end-module** occurs, or the results may not be as expected. Ensure that any **expose-module** command is matched with a **previous**, etc.

Output Wordset

Quartus Forth supports the Standard words for text output, and adds extensions for managing output on Palm Powered devices.

Implementation details for Standard words:

- **EMIT?** (-- *flag*) Returns a **TRUE** flag when the active form is also the active draw window.
- **CR** (--) Calls **EMIT?** to determine if it is currently able to output to the active form. Provides automatic scrolling within the **window-bounds** region when text output would exceed the lower edge of the region. Updates **currentx** and **currenty** to point to the start of the next output line. Responds appropriately to the current pagination setting of **more**. This word is kernel-deferred.
- **PAGE** (--) Calls **EMIT?** to determine if it is currently able to output to the active form. Erases the **window-bounds** screen region. Sets **currentx** and **currenty** to the top-left corner.
- **TYPE** (*c-addr u* --) Calls **EMIT?** to determine if it is currently able to output to the active form. Updates **currentx** and **currenty** after displaying the string specified by *c-addr u*. Proportional and non-proportional fonts are both handled properly. Responds appropriately to the current setting of **wrap**. This word is kernel-deferred.
- **EMIT** (*char* --) Built using **TYPE**.

Quartus Forth Extensions:

- **ShowForm** (*form-id* --) Displays and makes the specified GUI form active, and sets **window-bounds** according to the dimensions of the form. Sets **currentx** and **currenty** to the top-left corner. Establishes a default form handler.
- **MainForm** (--) Displays a blank form with a Graffiti shift-state indicator. Equivalent to **MainFormID ShowForm**, with a reduction in the **window-bounds** y-extent by eleven pixels to accommodate the Graffiti shift-state indicator when scrolling. At the Quartus Forth console, it also enables the Quartus Forth menu and menubar functions.
- **currentx** (-- *addr*) A variable holding the current window-relative x (horizontal) pixel coordinate for output.
- **currenty** (-- *addr*) A variable holding the current window-relative y (vertical) pixel coordinate for output.
- **window-bounds** (-- *addr*) A `RectangleType` (four 16-bit values) defining the screen-relative x- and y-position and x- and y-extent of the draw window of the currently active form. Set automatically by **ShowForm**. It is not prudent to modify these values, as dynamic form resizing in later OS versions will reset them.
- **?cr** (--) Performs **CR** if **currentx** is not at the left-hand edge of the **window-bounds** region.
- **wrap** (*newflag* -- *oldflag*) Controls output wrapping. When enabled, if the width in pixels of the text to be displayed by **TYPE** exceeds the right-hand edge of the **window-bounds** region, **CR** is performed first. When disabled, text is allowed to overrun the right-hand edge of the region. **wrap** is enabled by default.

- **more** (*newflag -- oldflag*) Controls pagination for scrolling text, with a localized [more...] message. When [more...] is displayed, a `penDownEvent` will display the next line of output. A `keyDownEvent` will display the next page of output. If the key is <*enter*>, further pagination is disabled. **more** is disabled by default.
- **font** (*newfont -- oldfont*) Sets a new font for text output; returns the font number of the previously-active font.
- **MainFormID** (-- *form-id*) Returns the form ID of a built-in blank form with a Graffiti shift-state indicator. Used by **MainForm**.
- **BlankFormID** (-- *form-id*) Returns the form ID of a built-in blank form without a Graffiti shift-state indicator. For use with **ShowForm**.
- **TitledFormID** (-- *form-id*) Returns the form ID of a built-in blank form with an empty title bar, and without a Graffiti shift-state indicator. For use with **ShowForm**.
- **beep** (--) Plays a `sndInfo` beep.

Notes:

- **more** is intended to control pagination when scrolling output at the Quartus Forth console, and is not recommended for use in stand-alone apps. In fact, to meet recommended guidelines for application design, the screen as a whole should not be allowed to scroll at all in stand-alone apps.
- The console functions **WORDS** and **allwords** use **more** to paginate the list of words.
- The built-in forms identified by **MainFormID**, **BlankFormID**, and **TitledFormID** are intended for use in quick-and-dirty apps. Nothing prevents you from using them in production apps, but generally you'll want to define your own specialized set of form resources for the apps you write. See the “Stand-Alone Wordset” section for information on how to remove these forms from your stand-alone apps if you don't need them.
- The default **ShowForm** handler returns not-handled for menu items `10000-10007` (the Palm OS `sysEditMenu` items) so that the OS can perform the appropriate edit functions.
- Console warning/error messages use an internal version of **TYPE** that is not deferred.
- Fonts numbered 0-7 are available back to Palm OS 3.0, and are available as named constants. Palm OS 2.x and 1.0 only support fonts 0-6. **ABORT** (or any uncaught **THROW**) restores the active console font to font number 0 (`stdFont`).
- Use the value returned by **EMIT?** to keep any drawing you do on a form out of menu windows and alerts.
- Under Palm OS Cobalt, the Graffiti shift-state indicator is in the input area and does not appear on the console form, so the **MainForm** y-extent is not reduced.

See also library modules: facility textalign thintype

Source Wordset

Quartus Forth reads and compiles source directly from Memo Pad memos. A memo is identified as a file if it begins with the character \ and a space, followed by a filename.

With the inclusion of the library module docinc, Quartus Forth can also read source from Doc-format files, both compressed and uncompressed.

The Standard words **INCLUDED** and **REFILL** are provided, along with a few extensions.

Implementation details for Standard words:

- **REFILL** (-- *flag*) **REFILL** handles input lines of up to 256 characters long. Tab characters in source are treated as spaces.

Quartus Forth Extensions:

- **include** (*"filename"* --) Like the Standard word **INCLUDED**, but parses the filename from the input buffer. Discards the rest of the line after the filename.
- **needs** (*"filename"* --) Like **include**, but does not re-include a file that has already been included by **INCLUDED**, **include** or **needs**.
- **echo** (*newflag* -- *oldflag*) If enabled, all source lines are displayed as they are read. Disabled by default.
- **set-memodb** (*creator-id. type.* --) Changes the database from which source files are read (default is **(id) memo (id) DATA set-memodb**). The format of the database must be the same as that used by the Memo Pad app – a database of text records. If used, place it at the very end of your startup.quartus file in the Memo Pad.

See also:

- Library module docinc

Notes:

- The Standard word **MARKER** also restores **needs** tracking.

Stand-Alone Wordset

Once you've written an app, the Quartus Forth can create a complete stand-alone executable (called a *PRC* in the Palm world – short for "Palm Resource Container") that is optimized to contain only the code required. The resulting PRC executable is automatically copied to your desktop during the next HotSync operation.

Stand-alone apps created using Quartus Forth are turnkey, ready for immediate use. They require no run-time library and can be beamed, copied, and handled just like any other Palm PRC executable. The Stand-Alone wordset provides all the tools needed to create a stand-alone app.

- **MakePRC** (*xt creator. "**name**"* --) Generates a stand-alone app.
- **CopyRsrc** (*number type.* --) Copies a selected resource into the stand-alone app.
- **DelRsrc** (*number type. -- err*) Deletes a selected resource from the stand-alone app.
- **NewRsrc** (*number type. -- err*) Creates a new resource in the stand-alone app.
- **generate-symbols** (*boolean* --) Controls the creation of debug symbols in the app.

MakePRC is the cornerstone of the Stand-Alone wordset; it creates a new application in Palm storage memory with the specified name and creator ID, beginning with the specified xt and recursively extracting all required code from that point. It displays a series of dots in the console as it completes its task.

Important: Creator IDs must be unique; each app must have a different one. If your app is to be distributed to others, the creator ID passed to MakePRC ***must*** be registered with Palm at: **http://www.palmsource.com/developers/**
You'll first need to set up a profile as a developer (click on "Members Area", and then "Register"). Registration is free, and ensures that your creator ID will not conflict with any other developer's app.

CopyRsrc copies a specified resource from any currently-open resource database into the stand-alone app generated by **MakePRC**. See the resources library module documentation for details on how to open your own custom resource databases for use with **CopyRsrc**. If used, it must occur *after* **MakePRC**.

DelRsrc deletes a specified resource from the stand-alone app generated by **MakePRC**. If used, it must occur *after* **MakePRC**.

NewRsrc creates a new, empty resource in the stand-alone app generated by **MakePRC**. If used, it must occur *after* **MakePRC**.

generate-symbols is for advanced use. It generates debug symbols in the stand-alone app, for use with special debugging tools in conjunction with the Palm OS Emulator and Simulator. It includes the name of each word during compilation so debugging tools can use the names when reporting errors, running profiles, etc. If used, **generate-symbols** must occur *before* **MakePRC**. **generate-symbols** is false (disabled) by default.

Here's a complete “Hello World!” example:

```
\ hello

: go
   MainForm  \ Set up a form to draw on.
   ." Hello, World!"  \ Display a message.
   begin  ekey drop  again  \ Do nothing until the user starts another app.
;

' go  (id) demo  MakePRC Hello!

\ Optional deletion of default forms we don't need for this app:
1001  frmRscType  DelRsrc throw
1002  frmRscType  DelRsrc throw
```

Then, at the Quartus Forth console:

```
include hello <enter> ................................ ok
```

Now, in the Launcher, you'll see a new app named Hello!. This app is flagged for backup, and will be automatically copied to your desktop during the next HotSync operation. It will appear in the appropriate Backup folder, named `Hello!.PRC` with a size of just over 4K. (It is interesting to note that the default icons make up nearly 1.5K of that.)

Restrictions of a Stand-Alone App:

Stand-alone apps contain only the code and data that the app requires to run, and so certain functions of the Quartus Forth compiler and console are not available to stand-alone apps at run-time, to wit:

- There's no Forth dictionary in a stand-alone app, so you can't search for words, or manipulate the search order, or **EVALUATE** source, or load source from files. Therefore, no **INCLUDED**/**INCLUDE**/**NEEDS**, no **SOURCE-ID**, no **SAVE-INPUT**/**RESTORE-INPUT**, no **REFILL**.
- Codespace is only writable from within the Quartus Forth compiler, so your stand-alone app cannot write to its own codespace, or compile new code, or create new **CONSTANT**s or **VARIABLE**s. You also cannot directly change the action of kernel-deferred words from within a stand-alone app (but see the Defer wordset documentation for a solution to this).
- There's no Quartus Forth console in a stand-alone app, so you can't **QUIT**.
- Dataspace cannot be resized from within a stand-alone app – this means no **ALLOT**, so therefore no **,** **C,** or **ALIGN**. **ALLOT** appropriate space for your app before invoking **MakePRC**, and/or use the memory library module to allocate new memory at run-time, as required.

Notes:

- **true echo drop** before **MakePRC** will cause it to display the name of each word traversed during the stand-alone generation process.

- By default, a stand-alone app generated by **MakePRC** automatically contains certain resources, some of which you may wish to change (or to delete if you don't need them, as may be the case with some or all of the default `tFRM` resources):
  - `tAIB 1000` – A large Launcher icon, in both monochrome low-res and color low-res and hi-res versions, with appropriate transparency. You'll likely want to create your own icon for any app you distribute to others. See "Creating Your Own Resources" in the Quartus Forth User Manual for details.
  - `tAIB 1001` – A small Launcher icon, in low-res only:
  - Three default forms are provided. For convenience, each of these forms also contains a single-line text field with an ID of 1003; **ACCEPT** uses this field. (Note, however, that **ACCEPT** as defined by the Forth Standard does not meet Palm guidelines for user interfaces, and is not recommended for use in stand-alone apps; use your own custom fields instead.)
    - `tFRM 1000` – A blank form with a Graffiti shift-state indicator (used by **MainForm**). If you use **MainForm** in your app, you'll need this resource (or an alternate `tFRM 1000` resource). If you don't need this resource, it can be safely deleted with **MainFormID frmRscType DelRsrc throw** after **MakePRC**.
    - `tFRM 1001` – A blank form without a Graffiti shift-state indicator (used by **BlankFormID ShowForm**). If you are not using this form, it can be safely deleted with **BlankFormID frmRscType DelRsrc throw** after **MakePRC**.
    - `tFRM 1002` – A blank form with a title bar, with no title by default (used by **TitledFormID ShowForm**). If you are not using this form, it can be safely deleted with **TitledFormID frmRscType DelRsrc throw** after **MakePRC**.
  - `Talt 1005` – The alert used by **THROW** to report errors.
  - `tver 1` – A version resource, used by the Launcher Info dialog to show the version of the app. Defaults to "1". You'll likely want to change this in an app you distribute to others.
  - `pref 1` – A preference resource that launches the app with expected parameters (such as a return stack and globals). It would be unusual to want to change this.
- **true generate-symbols** makes the code segment of a stand-alone app approximately 10% larger.
- The data segment of a stand-alone app (resource `DATA 1`) contains all dataspace contents up to **HERE** at the time of **MakePRC**.

See also:

- The "Launchcode Wordset" section.

Quartus Forth Wordsets: Advanced

Callback Wordset

Certain Palm OS routines (list drawing routines, for instance) allow for the use of *callbacks*. Callbacks are words in your app that you set up to be “called back” by the Palm OS at certain times by certain routines. Quartus Forth provides a Callback wordset to make this easy; parameters for callback words appear on the data stack just as they would for any other Forth word.

- **callback** (--) Establishes a callback environment.
- **end-callback** (--) Ends a callback environment.
- **callback-stack** (*u* --) Establishes a private return stack for callbacks, with a size of *u* bytes.
- **handled** (*flag* --) Sets the 32-bit `D0` register to the 16-bit value of *flag*.

Here's an outline:

```
: mycallback
   callback
      ... callback code here...
   end-callback ;

: go
   1024 callback-stack
   ... code that sets up the callback, etc. ... ;
```

In **go**, **1024 callback-stack** creates a private return stack for use during callbacks, along with additional space to preserve register values across the callback. `1024` is a reasonable value; however, if your callback words make heavy use of the return stack (nested calls, etc.) you may need to increase it. **callback-stack** must occur before any callback routines are installed and used. The callback stack is deallocated automatically when your app terminates.

At the start of **mycallback**, **callback** establishes the return stack as the data stack so the code in the callback word can access parameters passed to it by the Palm OS, and establishes the private callback stack as the return stack for the duration of the callback. At the end of the callback word, **end-callback** reverses this.

callback must *always* have a matching **end-callback**. No code in a callback word can come before **callback**, or after **end-callback**.

handled is useful for advanced use in certain kinds of event-handling callbacks.

Notes:

- It is not necessary to leave the same number of parameters on the stack when exiting a callback word as were there when it was entered; as long as you don't underflow the stack, you can manipulate the passed parameters on the stack as required.
- Under Palm OS 5 and later, callback code will often run fine even if you don't set up a **callback-stack**. However, all Palm OS versions before 5 absolutely require a **callback-stack**, so be sure to use it every time you use callbacks.

Dictionary Wordset

Quartus Forth incorporates a considerable number of compiler-security measures; these are checks to ensure that all control structures are appropriately matched – that is, that each **BEGIN** has a matching **AGAIN** or **WHILE**/**REPEAT**, each **IF** has a matching **THEN**, each **DO** has a matching **LOOP** or **+LOOP**, etc. Mismatched control structures will cause `-22` ("control structure mismatch") errors.

Implementation details for Standard words:

- **FIND** (*c-addr -- c-addr 0 | xt 1 | xt -1*) In Quartus Forth, **FIND** returns 3 for words flagged as inline.

Quartus Forth Extensions:

- **allwords** (--) An extension of the Standard word **WORDS**. Whereas **WORDS** only displays the words in the first wordlist in the current search-order, **allwords** displays all the words from all the wordlists in the current search order.
- **inline** (--) Flags the most current word as an *inline* word.
- **warnings** (*newflag -- oldflag*) Controls the display of compiler warnings (for example, 'redefined' messages during compilation). Enabled by default.
- **(header)** (*c-addr u -- colon-sys*) Begins a new definition with the name specified by *c-addr u*.
- **(find)** (*c-addr u -- 0 | lfa flag*) Similar to the Standard word **FIND**, but takes the name to search for as a string on the stack instead of parsing it from the input buffer, and returns an *lfa* (link field address) instead of an xt. Returns `0` if the name is not found in the dictionary under the current search-order. The flag: `1` for a normal word, `-1` for an immediate word, and `3` for an inline word.

Advanced developers sometimes wish to access underlying components of the compiler; to that end, Quartus Forth provides several low-level words. These words let you access internal details in a manner that will still work should a future version of the Quartus Forth change its internal structures.

- **lfa>xt** (*lfa -- xt*) Converts a*n lfa i*nto an xt. For use with **(find)**.
- **literalxt** (*xt --*) Compiles an xt into a definition as a literal value. An immediate word.
- **xt>abs** (*xt -- addr.*) Converts an xt to a double-cell 32-bit absolute address in the storage heap.
- **xt>size** (*xt -- u*) Returns the size (in bytes) of a word. Returns `0` if the word is an internal kernel function without a header; throws `-12` ("argument type mismatch") if the xt is otherwise not present in the dictionary.
- **.name** (*xt --*) Displays the name associated with the specified xt. If the xt is invalid, no name is displayed.

Launchcode Wordset

Whenever the Palm OS launches an application, it sends along a *launchcode* and certain other flags and information. When you start an app normally from the Launcher, it is launched with a `CmdNormalLaunch` launchcode. The Palm OS sends various other launchcodes under certain circumstances (after a device reset, in response to a Find request, and so on).

Quartus Forth provides a Launchcode wordset to allow your app full access to launchcodes and associated information.

- **activate-launchcode** (*u* --) Sets a stand-alone app to respond to the specified launch code.
- **launchcode** (-- *u*) The current launch code.
- **launchflags** (-- *flags*) The current launch flags.
- **cmdPB** (-- *addr.*) The 32-bit address of the current command parameter block.

One cumbersome aspect of the Palm OS is that the OS does not make global variables available to apps for most launchcodes other than `CmdNormalLaunch`; however, Quartus Forth overcomes that limitation. In Quartus Forth stand-alone apps, global variables are always available for all launchcodes.

Here's an outline of an app that handles multiple launchcodes:

```
: main  MainForm ... ;  \ normal function of the app with an event loop, etc.

: reset  ... ;  \ to be performed in response to a CmdSystemReset

: isactive? ( -- bool )
  launchflags sysAppLaunchFlagSubCall and ;

: go
    launchcode sysAppLaunchCmdNormalLaunch = if  main  then
    launchcode sysAppLaunchCmdSystemReset = if  reset  exit  then
    launchcode sysAppLaunchCmdGoto = if
        isactive? if  \ the app is being called as a subroutine of itself
            ... exit  \ use exit or (bye) to terminate, either will work
        then  ...
    then
    ... additional launchcode-handling code... ;

\ To test launchcode handling, a stand-alone app must be generated:
sysAppLaunchCmdGoto activate-launchcode  \ these must come before MakePRC
sysAppLaunchCmdSystemReset activate-launchcode
' go (id) .... MakePRC MyApp
```

cmdPB and **launchflags** return additional information from the Palm OS about the current launchcode. See the Palm OS Reference docs for further details as to what information is passed with each launchcode.

See also:

- the Finder example

Systrap Wordset

In the Palm world, Palm OS routines are known as *systraps*, short for “system traps”. A trap is an internal table of vectors used as an entry-point to all of the Palm OS system routines.

Quartus Forth provides seamless access to nearly 900 Palm OS systraps by name; all of them have names identical to the name given the underlying routine by PalmSource in their reference documentation available at **http://www.palmos.com/dev/support/docs/68k_books.html**.

All systraps integrated in Quartus Forth are listed with their arguments and Palm OS version numbers in the Quartus Forth Systrap Reference.

For advanced users who want direct access to the underlying systrap mechanism, Quartus Forth provides a lower-level Systrap wordset.

- **systrap** (*n* --) Calls the specified Palm OS routine by number, with applicable parameters read from the Quartus Forth data stack. The parameters are not removed from the stack after the call. Use **a0.L** and **d0.L** to retrieve return-values as required. The values for *n* are listed in the Palm OS SDK header files, available from **http://www.palmos.com/dev/dl/dl_sdks/**.
- Examples:

  The built-in systrap named **SysReset** is equivalent to:

```
: SysReset ( -- )  (hex) A08C systrap ;
```

  The built-in systrap named **FntGetFont** is equivalent to:

```
: FntGetFont ( newfont -- oldfont )
    >byte  \ Convert the byte parameter for the systrap.
    (hex) A163  \ The number of the FntGetFont systrap, from the Palm OS SDK.
    systrap  \ Call the Palm OS routine.
    drop  \ Drop the newfont parameter.
    d0.L  \ Retrieve the oldfont return-value.
    drop  \ Drop the unneeded high-part of the 32-bit value.
;
```

- **>byte** (*n* -- *n<<8*) Converts an 8-bit systrap parameter into the format expected by the Palm OS. Fewer than 70 of the built-in systraps require one or more of their parameters to be massaged in this way; the parameters are identified in the Quartus Forth Systrap Reference with a `[>byte]` tag. Equivalent to **: >byte 8 lshift ;**
- **a0.L** (-- *addr.*) Returns the double-cell 32-bit value of the system `a0` register. Used to read address return-values from systraps that return an address return-value.
- **d0.L**(-- *x.*) Returns the double-cell 32-bit value of the system `d0` register. Used to read non-address return-values from systraps that return a non-address return-value.
- **d0.L!** (*x.* --) Stores a double-cell 32-bit value in the system `d0` register. Formerly used with the now-obsolete **eventhandler**. Not useful for systrap access, but placed here with **d0.L** for convenience.

Library Files

ANS Library Files

These files contain additional ANS Standard Forth words not present by default in the Quartus Forth kernel.

allans

This library file loads all other ANS Standard Forth words (excluding the floating point words provided by NewFloatMgr).

case

This library file provides a control structure described in the Standard CORE-EXT wordset:

- **case**
- **of**
- **end-of**
- **end-case**

core-ext

Provides the following additional words from the Standard CORE-EXT wordset:

VALUE TO 2>R 2R> C" ERASE D>S .R U> U.R WITHIN [COMPILE]

Note:

- **PAD** is provided as part of the file library.

environment

This library file provides all **ENVIRONMENT?** strings reporting the state of the Quartus Forth kernel at startup. It's unlikely you'll have any use for this; it is provided for Standard compliance.

Notes:

- Other modules do not add or update **ENVIRONMENT?** strings.

facility

Provides the little-used **KEY?** and **AT-XY** from the Facility wordset. **AT-XY** is intended for use on a system with fixed-width fonts; the version here approximates that with the Palm OS proportional fonts.

facility-ext

Provides **TIME&DATE** from the FACILITY-EXT wordset.

fasin

Provides the Standard FLOATING-EXT word **FASIN**, for use with internal Quartus Forth FFP floats.

fatan

Provides the FLOATING-EXT word **FATAN**, for use with Quartus Forth FFP internal floats.

fel

Float exponents & logarithms for use with Quartus Forth internal FFP floats.

Provides the following Standard words from the FLOATING-EXT wordset:

- **FEXP FLN**

Provides the following extension words:

- **F>** (*F: r -- -- r.*) Moves an FFP float from the floating-point stack to the data stack.
- **>F** (*r. -- -- F: r*) Moves an FFP float from the data stack to the floating-point stack.
- **@F** (*F: r -- r -- r.*) Copies an FFP float from the floating-point stack to the data stack.
- **FX^Y** (*F: x y -- x^y*) Raises the FFP float *x* to the power denoted by the FFP float *y*.
- **FX^N** (*F: r -- r^n n --*) Raises an FFP float to the integer power denoted by *n*.

file

Provides **S"** from the FILE wordset.

Also provides **PAD** from the CORE EXT wordset.

Notes:

- **S"** as specified in the FILE wordset can be used to hold a transient string at the console, outside of a definition, unlike the default CORE **S"** that can only be used inside a definition.

float-ext

Additional words from the FLOATING-EXT wordset.

Provides the following Standard words for use with Quartus Forth internal FFP floats:

- **F. F~**

Provides the following extensions:

- **-frot** (**F:** *r1 r2 r3 -- r3 r1 r2*) Reverse rotates the top three items on the internal floating-point stack.
- **#trailing0** (*c-addr u1 -- u2*) Returns the number of trailing '0' characters in the provided string.
- **places** (-- *#places*) Returns the current setting for the number of places past the decimal point in numbers displayed by **F.**
- **set-places** (*#places* --) Sets the value for the number of places past the decimal point in numbers displayed by **F.**
- **(f.)** (**F:** *r* --) (-- *c-addr u*) As F., but returns the string representing the float, rather than displaying it.

Notes:

The **F.** provided in this library file is an implementation that works only with Quartus Forth internal floats. The fpout library file provides a full implementation of floating-point output words.

fpout

A full-featured set of words for displaying floating-point values. These words are vectorable, and can be used to display either double-precision IEEE floats, or Quartus Forth internal FFP floats. See the "Floating Point" section for more details.

Provides the Standard FLOAT words:

- **F. FE. FS.**

Provides the following extensions:

- **G.**
- **G.R**
- **F.R**
- **FE.R**
- **FS.R**

String versions of **FS.** **F.** **FE.** and **G.**:

- **(fs.) (f.) (fe.) (g.)**

ftrig

Additional FLOAT EXT words for use with Quartus Forth internal FFP floats:

Provides the Standard words **FSIN** and **FCOS**.

Provides the following extensions:

- 2pi
- pi
- fsgn

memory

Provides the following Standard words from the MEMORY wordset:

- **ALLOCATE FREE RESIZE**

Provides the following extension:

- **size**

Notes:

ALLOCATE and **RESIZE** can fail if the allocated block is not within +/- 32K of the data space base pointer. Total space allocatable using these Standard words is less than 32K per region, and less than 64K overall. For reliable allocation of memory regions, use the mem words.

string

Provides the Standard words:

- **/STRING**
- **BLANK**
- **-TRAILING**

tools

Provides the following words from the Standard TOOLS-EXT wordset: **? DUMP**

tools-ext

Provides the following words from the Standard TOOLS EXT wordset:

- **[IF] [ELSE] [THEN]**

Notes:

These words only work in sources stored in the MemoPad, and not in Doc-format files.

Miscellaneous Library Files

These library files contain words that do not fit readily into other sections.

125words

This library module provides backward compatibility with apps written for Quartus Forth 1.2.5.

Provides:

- **a0** Alias for **a0.L** (see the Systrap wordset)
- **d0** Alias for **d0.L** (see the Systrap wordset)
- **handleevent** Alias for **(handleevent)** (see the Events wordset)
- **_hash** Alias for **shash** (see the Miscellaneous wordset)
- **itemid** Outdated word for accessing the item ID of an event (see the Events wordset)

arcfour

An implementation of the Arcfour stream cipher encryption algorithm, also known as RC4. This is a small, fast, strong encryption facility.

- **arcfourkey** (*k-addr length --*) Sets the encryption/decryption key for **arcfour**.
- **arcfour** (*m-addr length --*) Encrypts/decrypts the specified message block using the Arcfour stream cipher encryption algorithm.
- **MD5** (*c-addr u -- c-addr2 16*) Calculate the MD5 checksum value for the specified block. Note: MD5 is only available on Palm OS 2.0 and up.

assert

- **assertion-level** (-- *addr*) A variable holding the current assertion level.
- **assert(** (*"...<close-parenthesis>"* --) Parses up to the next close-parenthesis. If the assertion level is non-zero, the code between the parentheses is compiled with an **ABORT"** test following it; if the code results in a **FALSE** result, the **ABORT"** is performed.
- **assert0(** Same as **assert(**.
- **assert1(** Same as **assert(**, but only if **assert-value** is `1` or greater.
- **assert2(** Same as **assert(**, but only if **assert-value** is `2` or greater.
- **assert3(** Same as **assert(**, but only if **assert-value** is `3` or greater.

Example:

```
: main
    ...
    assert( FrmGetActiveFormID 1000 = ) \ Assert that the current form id is 1000
    ...
;
```

Assertions are a very handy development tool. The code between the parentheses is evaluated and tested; if it evaluates as false, an "Assertion failed!" abort message appears.

Assertions compile nothing unless the variable assert-level is set as high or higher than the assertion level selected. The condition must evaluate as non-false, or an "Assertion failure!" error is generated. Use a non-zero assertion-level for for debugging. Set assert-level to 0 for a production build.

Be sure that the code between the parentheses has no side-effects – that is, it does not do anything other than a static test, and does not alter stack contents, does not change variables, does not establish new GUI state, etc. Your code should run identically whether or not the assertion level is zero.

calendar

An implementation of selected Gregorian and ISO date routines from N. Dershowitz & E.M. Reingold's 1997 book, *Calendrical Calculations*.

Provides:

- **dmy>date** (day month year – date.) Converts a date in day/month/year format to a double-cell date value.
- **date>dmy** (date. -- day month year) Converts a double-cell date value to day/month/year format.
- **date>day-of-week** (*date – day-of-week*) Converts a double-cell date value to its respective weekday.
- Finds a weekday relative to a given date. All have the stack diagram (*d1 k -- d2*), where d1 is the specified double-cell date value, k is the required weekday, and d2 is the resulting double-cell date value:
  - **before**
  - **on-or-before**
  - **nearest**
  - **on-or-after**
  - **after**
- Finds the nth weekday relative to a given date. All have the stack diagram (*d1 k -- d2*), where d1 is the specified double-cell date value, k is the required weekday, and d2 is the resulting double-cell date value:
  - **last**
  - **first**
  - **second**
  - **third**
  - **fourth**
  - **fifth**
- **leapyear?** (*year -- bool*) Returns **TRUE** if the specified year is a leap year.
- **iso>date** (*day week year -- date.*) Converts an ISO date (in day/week/year format) to a double-cell date value.
- **date>iso** (*date. -- day week year*) Converts a double-cell date value to an ISO date (in day/week/year format).

Example:

This example calculates the date of the first Sunday after April 15, 2004:

```
needs calendar <enter>  ok

15 April 2004  dmy>date <enter>  ok..
Sunday after  date>dmy  . . . <enter>  2004 4 18  ok
```

Notes:

- Constants for weekdays and months are built-in constants. See the "List of Built-In Constants" for details.

comma

Formats double-cell integers with separators.

Provides:

- **thousands-separator** (-- *char*) A VALUE, initially set to ','.
- **ud.comma** (*ud.* --) Format and display an unsigned double-cell number.
- **d.comma** (*d.* --) Format and display a signed double-cell number.
- **.comma** (*n* --) Format and display a signed single-cell number.

condthens

This is a shorthand for deeply-nested IF/ELSE conditionals. Ahead of the first IF, case sets a sentinel marker; after all the IF/ELSE conditionals, thens compiles the required number of THENs.

- **cond**
- **thens**

dblmath

Provides:

- **dm***
- **dm/**
- **HiBit**
- **ud/mod**
- **dmod**
- **admod**

dblmath-ext

Provides:

- **extract-signs**
- **d***
- **d/**

input

Helper words for accepting user input in a popup box.

Note: This section is currently being revised.

memo

Helper words for working with the MemoPad database.

Provides:

- **OpenMemoDB**
- **CloseMemoDB**
- **WriteNewMemo**

mersenne

A high-strength pseudo-random number generator.

Provides:

- **sgenrand** (*seed.* --)
- **lsgenrand** (*&seed-array* --)
- **genrand** (-- *u.*)

opg

Also: opg.1 opg.2 opg.3 opg-docs opg-extras

Wil Baden's Operator Precedence Grammar. Allows for infix floating-point syntax; it converts infix notation to postfix notation.

See opg-docs for details on the words provided.

Example: (pending)

Notes: This module works with Quartus Forth FFP floats only, but that doesn't stop it from being a valuable tool for double-precision floating-point operations. When DEBUG is set to TRUE, LET displays the infix version of any formula; this can then easily be converted to work with double-precision floats.

ran4

Gordon Charlton's random number generator.

Provides:

- **seed**
- **ran4**

Note: This section is currently being revised.

regs

Displays the value of all the 68K regs.

Provides:

- **.regs**

Notes: Uses the asm68k assembler.

This section is currently being revised.

string>float

Provides:

- **string>float**

Notes: This section is currently being revised.

struct

- **struct**
- **field**
- **end-struct**

Notes: Unlike the built-in constants, structure offsets created with this module are self-adding.

This section is currently being revised.

tester

An implementation of John Hayes' tester module. This is an excellent facility for instrumenting your code with automated inline tests in the form **{ test -> result }**

- **TESTING**
- **VERBOSE**
- **show-test**

Example:

```
TESTING ADDITION
{ -> }  \ Test for an empty stack
{ 3 5 + -> 8 }
```

This section is currently being revised.

textalign

Provides:

- **type.left**
- **pixel-width**
- **type.right**
- **type.center**

This section is currently being revised.

tinylocals

A simple facility for locals.

Notes: This locals module is non-Standard. Quartus Forth does not provide the LOCALS wordset; however, for rare circumstances where locals are desired, this module should fit the bill.

This section is currently being revised.

toolkit

Provides a number of common-usage words:

- **place**
- **place,**
- **bounds**
- **>lower**
- **>upper**
- **append**
- **array**
- **under+**
- **enum**
- **[end]**
- **[defined]**
- **0allot**
- **3drop**
- **4drop**
- **6drop**
- **string:** Shorthand for creating named literal strings. A 2CONSTANT called <name> is created that points to the following string, delimited by the first character encountered.

Notes: This section is currently being revised.

trig

Integer trig operations.

Notes: This is an antique module that has been in the system since it was named PilotFORTH. It is used by turtle.

turtle

Simple turtle-graphics functions.

Notes: This is an antique module that has been in the system since it was named PilotFORTH. It uses trig.

udmultiply

- **udm***
- **ud***

This section is currently being revised.

zstrings

For creating literal zero-terminated strings, required by some Palm OS API functions.

Provides:

- **z"**

System Library Files

These library files all provide interfaces with different parts of the Palm OS API.

armasm

...

The armasm ARM Assembler Module

The armasm module provides an inline symbolic ARM assembler with structured conditionals and helper words for working with large immediate values.

Also provided are defining words for creating PACE Native Objects (PNOs). See the PNO section for further details.

armasm Glossary:

- **pno** (*"name"* --) Begins a named PNO definition. The resulting word will automatically execute the ARM subroutine defined by the assembler instructions between **pno** and **end-pno**. It will have a stack diagram of (*addr -- result.*), expecting a 16-bit parameter data address, and returning a 32-bit value on the stack.
- **end-pno** (--) Ends a **pno** definition.
- **arm?** (-- *boolean*) Returns a **TRUE** flag if the CPU is an ARM. Your applications should provide non-ARM subroutines that perform the same functions as its **pno**s, and use them when the CPU is not ARM. See the pno-demo for an example of how to do this.
- **armassembler** (--) Replaces the first wordlist in the current search order with the **armassembler** wordlist. Typical use: **also armassembler**

The asmassembler wordlist:

The following words are in the **armassembler** wordlist, accessible from within a **pno** word:

ARM assembly mnemonics:

ADC, ADD, AND, B, BIC, BL, BX, CMN, CMP, EOR, LDM, LDR, MLA, MOV, MUL, MVN, ORR, RET, RSB, RSC, SBC, STM, STR, SUB, SWI, TEQ, TST,

Macros that generate 1-4 instructions to handle large immediate values (all take double-cell immediate values):

#ADD, #AND, #BIC, #EOR, #MOV, #MVN, #ORR, #SUB,

Addressing modes:

#+@ #+@! #-@ #-@! #@+ #@- +@ +@! -@ -@! 0@ @+ @-

Shifts:

ASL ASR LSL LSR ROR RRX #ASL #ASR #LSL #LSR #ROR

Single-cell and double-cell immediate values:

# #.

LDM/STM addressing modes:

DA DB EA ED FA FD IA IB

{ } ^ @!

Loading and storing:

PCR PC-Relative

BYTE Byte-width memory access

Registers:

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15

SP Stack Pointer: alias for **R13**

LR Link Register: alias for **R14**

PC Program Counter: alias for **R15**

Condition codes:

AL CC CS EQ GE GT HI HS LE LO LS LT MI NE PL VC VS

REVERSE Reverses the sense of a condition code.

Shorthand for moving immediate values on/off a stack: **POP, PUSH,**

Structured conditionals:

AHEAD, AGAIN, BEGIN, ELSE, IF, REPEAT, THEN, UNTIL, WHILE,

Miscellaneous armasm words:

CODE, (*d.* --) Writes a double-cell value to data space in little-endian format.

RESET (--) Resets the assembler state.

Miscellaneous generic helper words:

?INVERT

BITCOUNT DBITCOUNT

DRROTATE DLROTATE

>-< -ROT

-2ROT 2NIP 3DUP D<> UD. UD< UD>

asm68k and asm68k.part2

Note: This section is currently being revised.

asm68k General Information

asm68k is a full-featured symbolic assembler for the Motorola 68000 processor, ported to run in Quartus Forth. With it you can write assembler code directly for the CPU in the PalmPilot, either as wholly-assembler code words, or as inline assembler sequences embedded within Forth words. Features of *asm68k* include:

- 119 mnemonic instructions
- 13 addressing modes
- Structured conditionals
  - `IF...THEN...ELSE`
  - `BEGIN...AGAIN`
  - `BEGIN...UNTIL`
  - `BEGIN...WHILE...REPEAT`
  - `FOR...NEXT`
- Prefix or postfix operation

As at version 1.21, the source for the assembler is in two files, **asm68k** and **asm68k.part2**, together totalling 7191 bytes. When compiled it occupies approximately 7516 bytes of code space, and 292 bytes of data space.

To use it, download both of the files and import each of them to your pilot as memos using the Pilot Desktop software. From the Quartus Forth command line, type: **`include asm68k`** <*enter*>

You will see: `Loading asm68k v1.21...done.`

Check the File Area for sample assembler code to get you started.

The original author of *asm68k* was Michael Perry of **F83** fame (an early and very popular implementation of the Forth-83 standard, together with Henry Laxen). The listing appeared in a special edition of Dr. Dobb's Journal, *the Toolbook of Forth*. His source (F83 compatible, in block format) can be found at ftp://ftp.taygeta.com/pub/Forth/Compilers/cross/68000/68kasm.arc.

asm68k Implementation Details

68000 mnemonics are implemented in *asm68k* as Forth words that process operand information from the data stack and compile machine instructions into code space. In `prefix` mode, the remainder of the current line is evaluated before operand information is processed.

The mnemonics are normal non-immediate words, which means that in order to assemble instructions, they must execute either in interpretation state, or from within another immediate word. Here's an example:

```
code under+ ( a b c -- a+c b )
  tos 2 sp d) add
  ] drop [
end-code
```

Note that because `drop` is a Forth word, we must switch into compilation state with `]`before it and back into interpretation state with `[` after. Here's a differently-coded but otherwise identical implementation of `under+`:

```
: under+ ( a b c -- a+c b )
  [ also assembler  tos 2 sp d) add  previous ]
  drop
;
```

Here we switch into interpretation state with `[` to assemble the `add` instruction, and then back into compilation state with `]` to compile `drop`.

Another word:

```
code 2- ( n -- n-2 )
  2 tos subq
end-code inline
```

asm68k Caveats

When mixing Forth and assembler within a definition, bear in mind that the following words have a different meaning when found in the `ASSEMBLER` wordlist:

```
# 0< 0= < > A0 AGAIN AND BEGIN D0 ELSE FOR IF MOVE NEXT OR REPEAT SWAP THEN
UNTIL WHILE
```

asm68k Addressing Modes

Addressing Mode	Generation	asm68k Syntax	asm68k Example (Postfix)
Register Direct Addressing			
Data register direct	ea = Dn	**`Dn`**	`d0 d1 move`
Address register direct	ea = An	**`An`**	`a3 a0 move`
Absolute Data Addressing			
Absolute short	ea = (next word)	**`n #)`**	`3700 #) d0 move`
Absolute long	ea = (next two words)	**`n L#)`**	`123456. l#) jmp`
Program Counter Relative Addressing			
Relative with offset	ea = PC + d_{16}	**`n PCD)`**	`d0 56 pcd) .b move`
Relative with index and offset	ea = PC + Xn + d_8	**`n Xn PCDI)`**	`100 d1 pcdi) a0 lea`
Register Indirect Addressing			
Register indirect	ea = (An)	**`An )`**	`d0 a0 ) .b move`
Postincrement register indirect	ea = (An), An := An + N	**`An )+`**	`a7 )+ d7 .w move`

Predecrement register indirect	An := An – N, ea = (An)	**`An -)`**	`d0 a6 -) .w move`
Register indirect with offset	ea = (An) + d_{16}	**`n An D)`**	`15 a1 d) .b clr`
Indexed register indirect with offset	ea = (An) + (Xn) + d_8	**`n Xn An DI)`**	`16 d0 a0 di) .l neg`
Immediate Data Addressing			
Immediate	data = next word(s)	**`n #`**	`42 # d0 move`
Quick immediate	Inherent data	**`n`**	`7 d1 addq`

asm68k Assembler Directives

Assembler Directive	**Action**
`.W`	Cause subsequent generation of word-sized operations (2-byte)
`.L`	Cause subsequent generation of long-sized operations (4-byte)
`.B`	Cause subsequent generation of byte-sized operations (1-byte)
`ASSEMBLER`	Replace the first wordlist in the search-order with the ASSEMBLER wordlist.
`FORTH`	Replace the first wordlist in the search-order with the FORTH wordlist.
`CODE` <*name*> ... `END-CODE`	Creates <*name*>, saves the current search-order, performs `.W ALSO ASSEMBLER` a code until `END-CODE`, which restores the former search-order. <*name*> becomes a Forth word that can be flagged `IMMEDIATE` or `INLINE`.
`POSTFIX`	Switch the assembler to postfix mode (the default), where operands preceed instru
`PREFIX`	Switch the assembler to prefix mode, where operands follow instructions. Note tha mode, each mnemonic/operand sequence must be on its own line.
Structured Conditionals (8-bit displacement)	
`0= 0<> 0< 0>= < >= <= >`	Branch conditions; use where <*condition*> appears below. Note: these test the flags in the 68000 status register, and do not consume cells from the stack as do th Forth counterparts.

<condition> `IF` ... `ELSE` ... `THEN`	Conditional branching, as in Forth.
`BEGIN` ... `AGAIN`	A simple loop. As in Forth.
`BEGIN` ... *<condition>* `UNTIL`	As in Forth.
`BEGIN` ... *<condition>* `WHILE` ... `REPEAT`	As in Forth.
`D`*n* `FOR` ... `NEXT`	Loops backwards from D*n*-1 to 0.

asm68k Assembler Mnemonics and Addressing Modes

Addressing Modes	**Instructions**
( )	`RESET NOP RTE RTS`
(n)	`ANDI>SR EORI>SR ORI>SR STOP TRAP`
(n ea)	`ORI ANDI SUBI ADDI EORI CMPI ADDQ SUBQ MOVEM> MOVEM<`
(n An)	`LINK`
(n Dn)	`MOVEQ`

(ea)	`SET SNI SLS SCC SCS SNE SEQ SVC SVS SPL SMI SSE SLT SGT SLE JSR JMP MOVE`
	`TAS CLR NOT NEG NEGX TST`

(ea ea)	`MOVE`
(ea An)	`ADDA CMPA LEA SUBA`
(ea Dn)	`CMP CHK DIVU DIVS MULU MULS`
(ea Dn) (Dn ea)	`ADD AND OR SUB`
(ea Dn) (ea n #)	`BCHG BCLR BSET BTST`
(An)	`MOVE<USP MOVE>USP UNLK`
(Dn)	`EXT SWAP`
(Dn ea)	`EOR`
(Dm Dn) (m # Dn) (ea)	`ASL ASR LSL LSR ROL ROR ROXL ROXR`

(Da d An) (d An Da)	`MOVEP`
(Dn Dm) (An@- Am@)	`ABCD SBCD ADDX SUBX`
(An@+ Am@+)	`CMPM`
(Xn Xa)	`EXG`
(target)	`BRA BSR BHI BLS BCC BCS BNE BEQ BVC BVS BPL BMI BGE BLT BGT BLE`
(target Dn)	`DBRA DBHI DBLS DBCC DBCS DBNE DBEQ DBVC DBVS DBPL DBMI DBGE DBLT DBGT D`

Quartus Forth Registers

Register	Symbolic name	Purpose within Quartus Forth
A2	CS	Codespace segment pointer
A4	SP	Data stack pointer
A5	DS	Dataspace segment pointer
A7	RP	Return stack pointer
D7	TOS	Top element of the data stack

bitmap

For Palm OS 1 bitmaps (monochrome, 72 dpi).

Provides:

- **bitmap** (width height "name" --)

This section is currently being revised.

color

Routines for using color.

Provides:

- **color-depth** (*bits* --)
- **grayscale** (--)
- **monochrome** (--)
- **>rgb** (*r g b* -- *rgb.*)
- **rgb>** (*rgb.* -- *r g b*)
- **gray** (*gray* -- *rgb.*)
- **color>gray** (*r g b* -- *gray*)
- **set-colors** (*fore-rgb. back-rgb.* --)
- **get-colors** (*fore-rgb. back-rgb.* --)
- **foreground** (*rgb.* --)
- **background** (*rgb.* --)
- **color:** (*r g b "name"* --)
- **black** (-- *rgb.*)
- **dark-gray** (-- *rgb.*)
- **light-gray** (-- *rgb.*)
- **white** (-- *rgb.*)

Notes: This section is currently being revised.

colornames

A series of constants defining named colors, for use with color.

DataMgr

Simple wrappers for Palm DataMgr calls.

Provides:

- **cardnum**
- **UseCard**
- **OpenDB**
- **CloseDB**
- **CreateDB**

This section is currently being revised.

disasm and disasm.part2 disasm.part3 disasm.part4

- **SEE**
- **seecs**
- **seeany**
- **seebase**
- **(seeany)**
- **thrw**
- **systrap**

This section is currently being revised.

doc

- **GetRecord**
- **Decompress**
- **CloseDocDB**
- **OpenDocDB**

This section is currently being revised.

dspaces

This is an implementation factor of inifini.

*** add Chapman Flack's info here

docinc

- **DocIncluded**
- **DocInclude**
- **DocNeeded**
- **DocNeeds**
- **DocInclude”**
- **DocNeeds”**

This section is currently being revised.

events

Provides:

- **coords@**

This section is currently being revised.

fields

Helper words for working with field GUI objects.

Provides:

- **FieldFocus**
- **FieldReleaseFocus**
- **string>Field**
- **Field>string**

See also: string2anyfield

This section is currently being revised.

float-aliases

Original MathLib names for a number of MathLib functions that are given Forth-like names in the MathLib module.

This section is currently being revised.

float.h

Provides a number of floating-point-related constants. A Forth trnslation of the C header file, float.h.

floodfill

A non-recursive flood fill algorithm. Fills any bounded area with the current drawing color.

Notes: Do not fill any unbounded area, or area right on the edge of the screen.

OS 3.5 (or up) required.

This section is currently being revised.

forms

Helper words for working with GUI form objects.

Provides:

- **GetObjectIndex**
- **GetObjectPtr**
- **GetControlValue**
- **SetControlValue**
- **SetLabel**
- **until-drawn**
- **PopupForm**

This section is currently being revised.

graphics

Graphics words.

- **line**
- **point**
- **circle**
- **rounded-rectangle**
- **rectangle**
- **erase-rounded-rectangle**
- **erase-rectangle**
- **frame**
- **at**
- **cursor-position**

Notes:

This section is currently being revised.

inifini

This section is currently being revised. inifini is used by the MathLib library module.

MathLib

MathLib is a library of mathematical functions that operate on IEEE 754 double-precision floating-point values. This code is directly based on the excellent work of Chapman Flack.

Please consult the Quartus Forth User Guide: Floating Point section for further details on the MathLib library module.

NewFloatMgr

The Quartus Forth built-in floating-point format is known as "Motorola Fast Floating Point (FFP)". It's a single-precision format, fast, and accurate within its range; it has a small footprint, and works on any Palm all the way back to version 1.0 "Pilot" devices. It is, however, limited to six places or so of precision; while this is enough for many apps, some do require greater precision. This library is directly based on Chapman Flack's excellent work, and provides Quartus Forth apps access to the IEEE single and double-precision floating-point libraries in Palm OS 2.0 and later, and also provides the foundation for the wide range of functions in the freely-available MathLib library.

NewFloatMgr overview:

NewFloatMgr provides the SF and DF (single and double floating) data types, which are 32-bit and 64-bit floating point numbers in the industry standard IEEE 754 format, as specified in the ANS94 FLOATING EXT word set. It additionally provides words for native arithmetic and comparison in both formats, conversion between formats and between either format and signed or unsigned double-cell integers, and between DF format and printable strings, all courtesy of the built-in NewFloatMgr routines of PalmOS 2.0 and later. This support, built in to every 2.0 and later device, also allows four rounding modes to be selected (ToNearest, TowardZero, Upward, and Downward), and five exceptional conditions to be detected (Invalid, Overflow, Underflow, DivByZero, and Inexact). It provides words to convert between SF and DF and the Quartus-provided float datatype (which uses a non-IEEE-standard format). With Rick Huebner's MathLib installed, it also provides a full complement of native DF (53-bit precision) trig, hyperbolic, logarithmic, exponential and power, and miscellaneous functions.

Note: MathLib is available for free download: http://www.probe.net/~rhuebner/mathlib.html

Relevant library files:

dfout

Floating-point output.

fpround

A version of **_fp_round** that works around a known Palm OS bug.

fsfdf

Conversion between the Quartus float format and the IEEE standard.

FLOATING EXT words: **SF@ SF!** -- convert between a Quartus float on the float stack and a standard single float in memory.

also

F>SF (-- *sr*) (*f: r* --) always succeeds

SF>F (*sr* --) (*f:* -- *r*) throws appropriate exceptions for any standard float that cannot be represented as a Quartus float.

NewFloatMgr and NewFloatMgr.2

Interface words to the floating point support built into PalmOS 2.0 and later.

D>SF D>DF SF>DF DF>SF SF>D DF>D SF= SF<> SF< SF<= SF> SF>= DF= DF<> DF< DF<= DF> DF>= SFNEGATE SF+ SF* SF- SF/ DFNEGATE DF+ DF* DF- DF/

Note: don't be too quick to want to pare the comparisons to a minimal set; remember that IEEE floats include Not-a-Numbers, which are not <, =, or > any number, and therefore you cannot assume trichotomy.

- For completeness, the rest of the NewFloatMgr traps by their PalmOS names: **FlpBase10Info FlpFToA FlpAToF FlpCorrectedAdd FlpCorrectedSub FlpVersion _fp_get_fpscr** (return fp status code register bits to check if Overflow, Underflow, Invalid, DivByZero, or Inexact conditions have been detected since last check) `_fp_set_fpscr` (set the fpscr, say to zero the exception bits before a stretch of computation)
- The rest of the conversion words: **_f_utof _f_ulltof _f_lltof _d_utod _d_ulltod _d_lltod _f_ftoq _f_qtof _d_dtoq _d_qtod _f_ftou _f_ftoull _f_ftoll _d_dtou _d_dtoull _d_dtoll**
- The rest of the comparison words: **_f_cmp _d_cmp _f_cmpe _d_cmpe** (return one of the flags flpEqual, flpLess, flpGreater, or flpUnordered; the 'e' versions also set flpInvalid in the fpscr if the result is flpUnordered)

  _f_fun _d_fun (true if operands unordered, i.e. at least one is NaN)

  _f_for _d_for (true if the operands are ordered, i.e. neither is NaN)

  _f_feq _f_fne _f_flt _f_fle _f_fgt _f_fge (like **SF=** etc. but return a 32-bit **_d_feq _d_fne _d_flt _d_fle _d_fgt _d_fge** instead of a 16-bit flag)

  _f_neg _d_neg (like **SFNEGATE DFNEGATE** but not inlined, just for completeness)
- Also **DF@ DF! DFABS DFMIN DFMAX SFABS SFMIN SFMAX** (fpcheck) – read and zero fpscr, then throw appropriate exception if the Overflow, Underflow, DivByZero, or Invalid status flag was set. The Inexact flag is ignored (happens too often to worry about except in very specialized software).

WARNING: The `FlpFToA` and `FlpAToF` functions built into PalmOS do not properly handle the full range of a double float. AtoF truncates the exponent at two digits, so if you convert 1.7e308 you will get 1.7e30. AtoF also does not like a + in the exponent: 3e6 and 3e-6 are ok, but 3e+6 will be read as 3e0 (without error indication). In fact AtoF has no way ever to indicate an error; it will return some double floating number no matter what string you feed it. FtoA seems to put the right text in the buffer for any magnitude from 1e-99 to 9e99 – but returns flpErrOutOfRange (1665) if the magnitude is less than about 1e-91 or more than about 9e98. Go figure. Computations done on double floats work fine for the full range (about 2.23e-308 to 1.797e308) but to convert or display very large or small values will take custom code.

AtoF will properly convert -0, but FtoA renders it the same as 0. That's almost backwards; you hardly ever need to enter a -0, but you might want to see when you've got one.

sfdf

Provides the FLOATING EXT words related to allocation and manipulation of the SF and DF data types:

SFLOAT+ DFLOAT+ SFLOATS DFLOATS SFALIGNED DFALIGNED SFALIGN DFALIGN SF, DF, SFDROP DFDROP SFDUP DFDUP SFLITERAL DFLITERAL SFCONSTANT DFCONSTANT SFOVER DFOVER SFSWAP DFSWAP SFROT DFROT SFVARIABLE DFVARIABLE

also

SF@SF SF!SF DF@DF DF!DF – ANS FORTH requires **SF@ SF! DF@ DF!** to perform automatic conversion between the IEEE standard format stored in memory and the format used internally by FORTH (Motorola FFP in Quartus Forth' case), so these new words are provided to fetch and store IEEE values directly to/from the data stack for use with the IEEE-native PalmOS and MathLib routines.

4>R 4R> – for manipulation of DF values.

mem

- **(allocate)**
- **(free)**
- **(size)**
- **(resize)**

This section is currently being revised.

random

Provides:

- **seed**
- **rand**

This section is currently being revised.

redefer

This section is currently being revised.

resources

Provides:

- **use-resources** (*creatr-id. type.* --) Opens a specified resource database in read-only mode so your app can make use of the resources therein. After a MakePRC, it is from such open resource databases that resources are copied into your stand-alone apps with CopyRsrc.

  The database is not closed – the Palm OS closes it automatically upon exit from Quartus Forth. Note that the debug Emulator and Simulator will indicate the open database status on exit.

serial

This section is currently being revised.

sound

This section is currently being revised.

string2anyfield

This section is currently being revised.

syncname

This section is currently being revised.

systraps

systraps is used by the disasm library module.

This section is currently being revised.

ver

Provides OSVersion (-- minor major), returning the version number of the Palm OS.

xts

This section is currently being revised.

Obsolete Library Files

These files were in earlier versions of Quartus Forth, but have now had their words moved into the kernel.

double

fonts

ids

safe

simple-handler

Advanced Topics

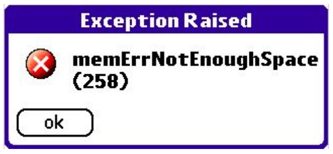
Exception and Error Handling

The Standard Exception wordset **THROW** and **CATCH** words provide an excellent facility for managing unexpected conditions within your applications. If a **THROW** goes uncaught, Quartus Forth provides a default mechanism for reporting the exception, as follows:

- At the Quartus Forth console, with the **MainForm** active, an uncaught **THROW** displays a readable error message (if available) directly. For example:

```
memErrNotEnoughSpace throw <enter>
throw? memErrNotEnoughSpace (258)
```

- At the Quartus Forth console, but not currently showing **MainForm**, **THROW** pops up a Palm alert dialog with the error number, and an error message (if available):

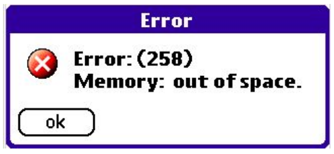


Exception Raised

memErrNotEnoughSpace
(258)

After selecting ok, the console **MainForm** is re-established, and then the throw is redisplayed in the console screen.

- In a stand-alone app running interactively, an alert pops up. For example:



Error

Error: (258)
Memory: out of space.

After selecting ok, the app terminates via **(bye)**.

Notes:

- **ABORT"** custom error messages are also handled cleanly, and will appear in pop-up error alerts as an error type of `-2`.
- Error numbers are always displayed in decimal.
- At the Quartus Forth console, error messages come from internal Quartus Forth error string tables first; if no message is available there for a specified error, or if a **THROW** occurs in a stand-alone app where the Quartus Forth error tables are not available, the message comes from the Palm OS error tables. This provides messages suited to a developer within the Quartus Forth console, while providing explanatory language-localized error messages for end-users of stand-alone apps.
- Palm OS 1 does not contain any error messages. Errors in stand-alone apps running under Palm OS 1 will only display the error number.
- If a **THROW** goes uncaught when an app is not launched as a GUI application, it will not display a pop-up alert, but instead will simply beep and then exit.
- The error alert resource is `Talt 1005`, automatically built into each stand-alone app. This alert can be replaced with your own customized alert, to provide additional information (contact info, etc.).

Application Termination

On a Palm Powered device, only one GUI application is running at any given time. Palm's development guidelines say that apps are not supposed to exit deliberately; instead, they should loop, processing events until they receive an `AppStopEvent` from the Palm OS event manager asking them to terminate so that another app can be launched.

Quartus Forth provides a set of words to manage application termination.

- **BYE** (--) A Standard word. Exits the app by simulating a tap on the Launcher icon.
- **(bye)** (--) Terminates the current application immediately.
- **byeThrow** (-- *byeThrow*) The Quartus Forth termination throw code; a constant.

When an `AppStopEvent` is received by the Quartus Forth event handler, **byeThrow THROW** is performed automatically; when the **byeThrow** exception is caught by the default exception handler, control is transferred to **(bye)**, and the app terminates. Normally this is completely transparent to you as a developer.

However, should it be required, your app can use **CATCH** to intercept requests to terminate and first perform any required cleanup, close any files or forms, save preferences, or whatever else is required before exiting – either directly by calling **(bye)**, or indirectly by re-throwing the **byeThrow** to be caught by the default exception handler, which will then call **(bye)**.

Here's an outline:

```
: main   ... your application here... ;

: go
   ['] main catch
   dup byeThrow = if ... perform any required cleanup here ... then
   throw ;
```

Another way to intercept a request for application termination is to redefer the kernel-deferred word **(bye)**. See the discussion of the redefer module in the Defer wordset section for more information.

Notes:

- If your app has any open forms, **(bye)** calls the Palm OS routine `FrmCloseAllForms` to close them for you.
- If you want your GUI app to terminate itself despite Palm's guidelines, use **BYE** to exit. If, instead, you were to simply let your main definition fall through to its end, the cleanup actions of **(bye)** would not be performed.
- An app must eventually respond to a **byeThrow** by terminating, or the Palm OS cannot start another application.

PNOs - PACE Native Objects (ARM Subroutines)

Palm applications are predominantly written to run on the Motorola 68K CPU found in all Palm-Powered devices from Palm OS 1 up to Palm OS 4. Palm Garnet (OS 5) and later devices are built using ARM CPUs that are not directly compatible with the 68K, but all such devices run a software subsystem (called PACE: Palm Application Compatibility Environment) that transparently emulates the 68K in order to run 68K Palm apps. This makes it possible to create and distribute Palm executables that are binary-compatible across all Palm-Powered devices.

For speed, the Palm OS on ARM-based devices is entirely written in native ARM code, so any overhead introduced by the emulation process is normally quite minor and unnoticeable. However, in some circumstances it is desirable to run certain computationally-intensive tasks at the full speed of the underlying ARM CPU. This is accomplished by writing special subroutines specifically for the ARM.

Palm calls an ARM subroutine a "PNO", which stands for 'PACE Native Object'. Quartus Forth provides `armasm`, an inline ARM assembler, for creating PNOs in your Quartus Forth apps.

Notes on Endianness:

The 68K is a big-endian environment -- that is, multi-byte values are stored in memory with the most significant digits first, and the least significant digits last. The ARM, on the other hand, is little-endian -- that is, multi-byte values are stored in memory with their least significant digits first, and their most significant digits last. This means endian conversion is required when passing data back and forth between a PNO and the emulated 68K environment.

Here's an example of the different endian storage schemes:

A 32-bit hex value: 12345678

For this value, 1 is the most-significant digit, and 8 is the least-significant digit.

Stored in big-endian 68K format:

    Byte Offset: value
    0: 12 <- most-significant first
    1: 34
    2: 56
    3: 78 <- least-significant last

Stored in little-endian ARM format:

    Byte Offset: value
    0: 78 <- least-significant first
    1: 56
    2: 34
    3: 12 <- most-significant last

About armasm:

The armasm module provided with Quartus Forth is a complete inline postfix assembler for the ARM. Along with ARM mnemonics, it also provides words for creating PNO definitions. See the armasm section for further details.

Deep details:

`PceNativeCall` (*&userData. &nativeFunc. -- ud.*)

`PceNativeCall` is the Palm OS system routine that temporarily transfers control to an ARM-native subroutine. It takes two parameters: the 32-bit absolute address of a parameter data block, and the 32-bit absolute address of the ARM subroutine itself. It returns a 32-bit result provided by the ARM subroutine. The ARM subroutine may also optionally modify data in the provided parameter data block.

The address of the parameter data is automatically little-endianed by `PceNativeCall`, and is available within the ARM subroutine in the `R1` register.

The ARM subroutine must return its 32-bit result in the `R0` register; `PceNativeCall` takes care of big-endianing this for the 68K environment and placing it on the stack upon return from the call.

Any other multi-byte values in the parameter data must be little-endianed for use by the ARM subroutine either before the `PceNativeCall`, or within the ARM subroutine itself. Quartus Forth provides the words **swapends** and **dswapends** for converting values from one endianness to another.

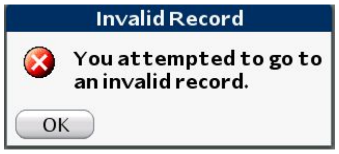
Notes:

- It appears to be necessary to preserve at least `R4` across an ARM subroutine. `R0`-`R3` can apparently be used within ARM code without preserving them.
- The ARM expects all code and multi-byte data to be aligned on 4-byte boundaries.

Known Issues

Palm OS Cobalt (Simulator), Version 6.1.0.0

- Under the Cobalt Simulator, the console "Last Error" menu option fails. The built-in Memo Pad app brings up an alert:

  
  Invalid Record

  You attempted to go to an invalid record.

  OK

  after which the Memo Pad launches normally and brings up its previous state.

  This appears to be an error in the current version of the Palm OS Cobalt Simulator PACE subsystem; it does not appear to be translating the command parameter buffer correctly for the Memo Pad app.
- Under the Cobalt Simulator, **THROW** error messages in the console are displayed from the Palm OS error tables first, and then from Quartus Forth internal error tables. This differs from the behavior under previous versions of the Palm OS. This appears to also be a Cobalt PACE error, relating to the order in which open resource databases are searched.
- Under the Cobalt Simulator, the Quartus Forth console correctly resizes its form when the screen is rotated, and when the input area is collapsed. It does not currently retain the existing content of the screen, nor is the input field relocated until <*enter*> is pressed. Should you wish to resize the screen or rotate the screen, it is presently best to perform a **COLD** afterward to reset the console state.

Fixes for these issues are deferred until after Palm OS Cobalt actually ships, as the issues may have gone away by then.