Steven Donahue

On-Board Development Tutorial

Developing Palm Pilot Applications On-Board Using Quartus Forth and RsrcEdit

Version 1.02

Quartus Forth Tutorial

Developing Palm Pilot Applications On-Board Using Quartus Forth and RsrcEdit

Introduction

Goals of this Tutorial

Welcome to my tutorial covering on-board application development using **Quartus Forth** and **OnBoard RsrcEdit**. I am developing this tutorial as a learning process for myself, but I hope that it will be useful to others like me who are just starting out.

Through the course of this tutorial, I will implement some basic applications to illustrate how to use the **Quartus Forth** development environment in conjunction with the **OnBoard RsrcEdit** resource editor to do on-board application development. In an effort to cover the use of several different types of resources, I will use subsequent examples to extend the functionality of these applications.

This tutorial is not really intended to be a Forth tutorial. However, the later examples cover several common areas of problems (strings, floats, etc) often discussed on Neal's Quartus Forth discussion forum (http://www.interlog.com/~nbridges/wwwboard/wwwboard.html).

The first two exercises focus primarily on the creation of the resources used for applications. The Forth code that drives this application is very trivial. The rest of the exercises will focus less on the resource creation and more on the Forth code required to drive an application that uses several different types of resources.

Finally, I make no claims to be a Forth expert. I have dabbled in Forth on and off since the mid-80's, but it has always been just for fun. I have never used Forth in any of my professional development efforts, primarily because I work for a company that uses a proprietary language for their telecommunications software development. I welcome constructive comments on any violations of Forth conventions or coding style in the examples used in this tutorial.

Credits

I am using two programs for this tutorial - **Quartus Forth** and **OnBoard RsrcEdit**. I have chosen these two programs as my development platform because they allow me to do all of my application development on-board. There are alternative resource editors/compilers available, but to the best of my knowledge, they do not allow on-board development. All examples and screen captures used in this tutorial are from one of these two programs.

Quartus Forth

At the time of development of this tutorial, I am using **Quartus Forth** v1.1.0R.

From Neal Bridges' **Quartus Forth** web page:

> Quartus Forth is an on-board ISO/ANSI Standard Forth optimizing native-code compiler for the Pilot™ , PalmPilot™ , Palm III™ , and IBM WorkPad™ connected organizers.

For more information about **Quartus Forth**, please visit:

http://www.interlog.com/%7Enbridges/quartus.html

OnBoard RsrcEdit

At the time of development of this tutorial, I am using **OnBoard RsrcEdit** v0.997.

From the Roger Lawrence's IndiVideo web page:

> OnBoard RsrcEdit is a Palm Pilot resource editor that runs on the Pilot and allows users to view and change application resources using a forms-based editing environment tailored for specific resource types.

For more information about **OnBoard RsrcEdit**, please visit:

http://www.individeo.net/

A Note About Creator IDs

Every Palm OS application, shared library, and feature has a four character Creator ID. This creator ID uniquely identifies the application, shared library, or feature to the Palm Operating System.

The examples in this tutorial use arbitrary and unofficial creator ID values. That is perfectly acceptable for testing or private development, as long as this creator ID value does not conflict with another application or shared library that is currently installed on your Palm Pilot.

However, if these applications were ever going to be released to the public, we would need to get an official creator ID. Palm Computing maintains the official list of Creator IDs. An official creator ID can be obtained at: http://palm.3com.com/devzone/crid/cridsub.html

The Basics

Where do all programmers start when exploring a new development environment? "Hello, World!" of course. This exercise will allow us to start with the most basic application possible - one that simply displays a message and allows no interaction. Some very basic interaction is added in the follow-up exercise.

HelloWorld - Exercise A

Exercise Goals for HelloWorld - Exercise A

- Develop a single form application that simply displays the "Hello, World!" message

Development Steps for HelloWorld - Exercise A

The following approach will be used to design and develop this application.

- Design and develop the resources to be used for this application
- Design and develop the Forth code to drive this application

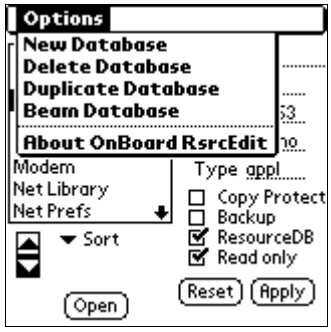
Resources for HelloWorld - Exercise A

Create the Resource Database

The first thing we need to do is create a resource database to be used for our application. The following steps outline how to create a new resource database.

- **Step 1 - Start RsrcEdit**: From your applications launcher, start the RsrcEdit application.

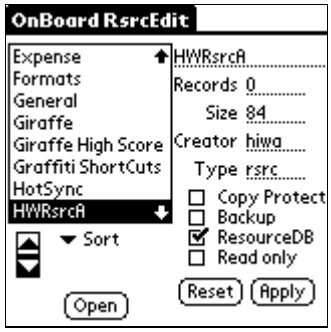
- **Step 2 - Create a New Resource Database**: From within the main RsrcEdit screen, tap the *MENU* silk screen button. Select *New Database* from the *Options* menu as seen below.



- **Step 3 - Supply Resource Database Information**: Fill in the critical information for the new resource database. ***Note:*** I am appending a letter ("A" in this exercise) to all resource database names, creator IDs, and Forth source files to help distinguish between tutorial exercises. Subsequent exercises will use these files as a starting point, and build on them.

    - fill in the *Name* field as "HWRsrcA"
    - fill in the *Creator* field as "hiwa"
    - fill in the *Type* field as "rsrc"
    - check the *Resource DB* box

- **Step 4 - Finish Resource Database Creation**: Tap the *OK* button to create the empty resource database. You should now see a similar screen display to the one shown below.



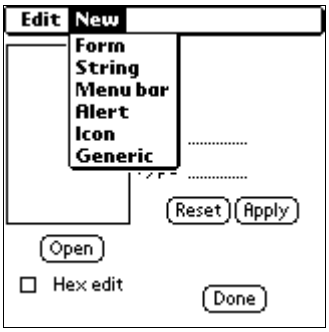
- **Step 5 - Set the Backup Bit**: In order for this resource to be backed up during a HotSync, the backup bit needs to be set. It is a good idea to backup the resource databases, in case of a hard reset. If the resource database was backed up, it can be re-installed from the backup directory.

- **Step 6 - Open the Empty Resource Database**: With the "HWRsrcA" database selected, tap the *Open* button to open the resource list. The box showing the list of resources will be empty for this new database, as illustrated below.

Add the Form Resource

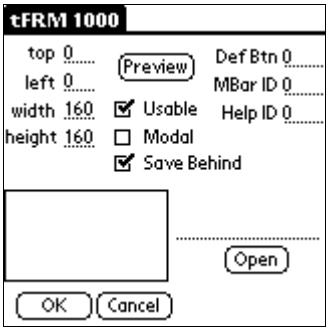
Now, we want to create the main screen (or "form") where we will display the "Hello, World!" message. The following steps outline how to add a form.

- **Step 1 - Add a Form to the Resource Database**: To do this, tap the *MENU* silk screen button. Select *Form* from the *New* menu as seen below.



- **Step 2 - Format the Form**: The main screen of our application must be formatted to look like we want it to. We are currently in the form properties screen. This screen allows us to change the look of the selected form. We want our form to take up the entire screen, so set the properties as follows.

  - set both the *width* and *height* fields to 160
  - set this form to useable by checking the *Useable* box
  - check the *Save Behind* box



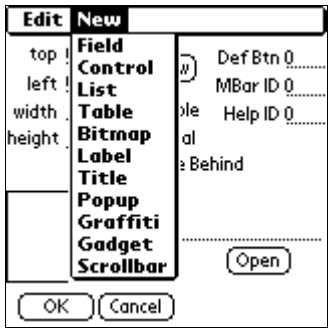
- **Step 3 - Preview the Form**: Now that our main form has been created, test it out with the *Preview* button. To return to RsrcEdit from the preview, tap the *APPLICATIONS* silk screen button.

  Hey! What's going on here? That was just a blank screen! The answer is, we need to define some more resources for our main form to give it some character.

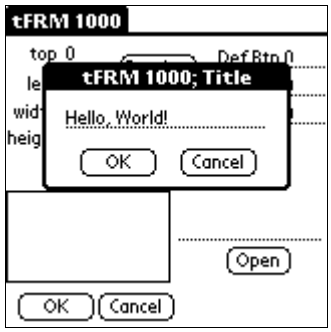
Add Resources to the Form

In order to add some character to our form, we need to create some additional resources associated with the form.

- **Step 1 - Add a Title to the Form**: Our form needs a title. To add a title, tap the *MENU* silk screen button from the form properties screen, and select *Title* from the *New* menu as seen below.



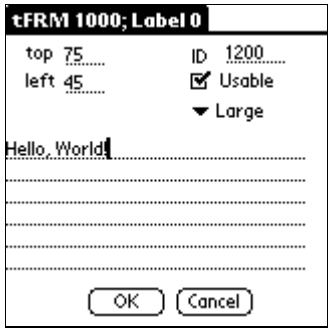
- **Step 2 - Fill in the Title**: Just fill in the string that we are going to use as our title, "Hello, World!", and tap the *OK* button.



- **Step 3 - Add a Label to the Form**: Let's also add a text label in the middle of the screen telling the world hello. Do this just like we did above for the title, except select *Label* from the *New* menu of the form properties screen.

- **Step 4 - Fill in the Label Properties**: Once inside the label properties screen shown below, fill in the fields as follows. The *top* and *left* fields specify the positioning on the screen where our label will be placed. The ID field is just a resource identifier that needs to be unique.

    - fill in the *top* field as 75
    - fill in the *left* field as 45
    - change the label *ID* to 1200
    - check the *Useable* box
    - select *Large* in the *Font* popup trigger
    - type the "Hello, World!" string in the text field



Tap the *OK* button to return to the form details screen when all of the fields are filled in.

- **Step 5 - Preview the Form Again**: Now that we have given our form some character, let's test it out with the *Preview* button again. Remember, to return to RsrcEdit from the preview, tap the *APPLICATIONS* silk screen button.

Save the Resources

Now that our screen looks like we want it to, we need to save all of these resources in the database.

- **Step 1 – Return to the Resource List Screen**: Tap the *OK* button on the form details screen to return to the resource list screen.

- **Step 2 – Update the Form Resource ID**: Change the *ID* field from "1000", to "1001" because we will be using "1000" for our icon bitmap resource ID later

- **Step 3 – Save and Exit**: Tap the *Apply* button, and then tap the *Done* button to save everything and return to the main RsrcEdit screen

We now have a resource database that is ready to be used by some Forth code.

Forth Code for HelloWorld - Exercise A

The Forth code for this program is very simple. We basically want to display our form, and ignore any subsequent input events. All exercises in this tutorial assume that you have imported the library.mpa file into your Memo Pad. If you have not already done this as part of the installation of Quartus Forth, you need to do so now. See the Quartus Forth documentation for more details on this.

The following source should be entered as a new entry in the Memo Pad application. I will attempt to explain what each line of Forth code is doing as I go.

- **Step 1 - Name the File**: The first part of the first line is important, as it distinguishes the name of the Forth "file". The date and initial time stamp can be modified or deleted, but the back-slash and the word following it designate the memo as a Forth file.

```
\ hello-a 1/20/99 10:20 pm - SCD
```

- **Step 2 - Include Necessary Files**: The following lines specify other library or source code files that are needed for this application.

```
needs ids
needs resources
```

- **Step 3 - Identify and Open Resource Database**: The following line identifies our resource database (remember naming it 'hiwa"?) and opens it for use.

```
(ID) hiwa (ID) rsrc  use-resources
```

- **Step 4 - Constant Definitions**: The next line identifies our main form id. If you used a number besides 1001 in the resource specification above, make sure to make the appropriate modifications here as well.

```
1001 constant HelloForm
```

- **Step 5 - Define Some Forth Words**: There are two new word definitions - *show-panel* and *go*.

    - *show-panel* simply puts the constant associated with our main form on the stack and invokes *ShowForm*. *ShowForm* displays the form identified on the stack, and establishes a default handler for it.

```
: show-panel ( -- )
  HelloForm ShowForm ;
```

- *go* is the main entry point. It executes the *show-panel* word and then goes into a loop that accepts event input via *ekey*, and immediately discards it.

```
: go  show-panel
  begin  ekey drop again ;
```

Full Source Listing

The entire source listing is shown below.

```
\ hello-a 1/20/99 10:20 pm - SCD

needs ids
needs resources

(ID) hiwa (ID) rsrc  use-resources

1001 constant HelloForm

: show-panel ( -- )
  HelloForm ShowForm ;

\ Main entry point:
: go  show-panel
  begin  ekey drop again ;
```

Test – Exercise A

We are finally ready to try out our first application! Enter Quartus Forth and type the following:

```
include hello-a
go
```

Voila! The first application works as we expected. Tap the *APPLICATIONS* silk button to exit the program.

HelloWorld - Exercise B

Exercise Goals for HelloWorld - Exercise B

- Extend the original HelloWorld program with a menu

- Link the new menu selection, "About HelloWorld", to a informational Alert window that tells about our program

- Associate a help string with the 'I" icon in the top right corner of the "About HelloWorld" box

- Create a standalone executable

Development Steps for HelloWorld - Exercise B

The following approach will be used to design and develop this extension of our existing HelloWorld application.

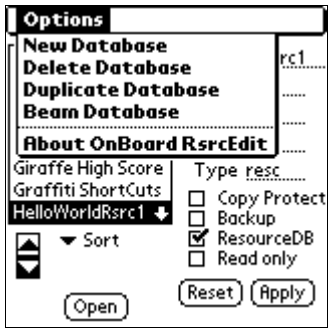
- Design and develop the screen resources to be used for this extension
- Change the Forth source code to handle the new menu event
- Create a standalone executable

Resources for HelloWorld - Exercise B

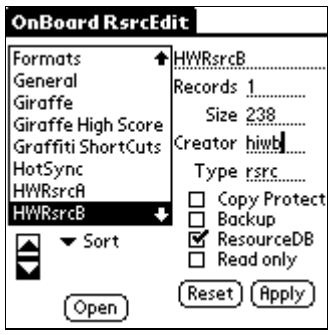
For this exercise, we will build on our existing resource file. Remember, because I wish to package these resource databases along with this tutorial, we will be making a copy of our original resources and using a different name. In a real-world scenario, we could just build on our original resource database instead of copying them.

Copy the Existing Resource Database

- **Step 1 – Start RsrcEdit**: Launch RsrcEdit from the application launcher.
- **Step 2 - Copy the Resource Database**: From within the main RsrcEdit screen, select the original "HWRsrcA" that we created in Exercise A. Now, tap the *MENU* silk screen button. Select *Duplicate Database* from the *Options* menu as seen below. This will create a resource database named "Copy ofHWRsrcA".



- **Step 3 – Rename the New Resource Database**: We now need to rename this new copy of the resource database. To rename the database, do the following sub-steps.
    - change the name of the new resource database from “Copy ofHWRsrcA” to "HWRsrcB"
    - change the *creator* id from “hiwa” to "hiwb"
    - tap the *Apply* button to accept these changes



Add the Menu Resource

We now need to add some new menu resources to the new resource database.

- **Step 1 – Open the Resource Database**: Make sure that “HWRsrcB” is selected, and tap the *Open* button to begin editing the resources for this exercise. We are now back in the resource list screen of RsrcEdit, and we should see the existing form created in Exercise A. We will now begin the process of creating the other resources described above in the goals for this exercise.

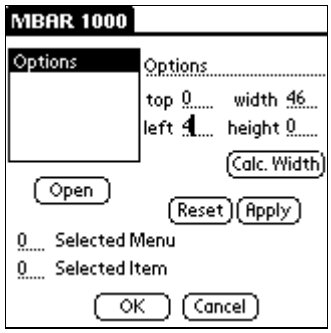
- **Step 2 – Add a Menu Bar Resource**: We need to create the menu bar for our main form. To do this, tap the *MENU* silk screen button. Select *Menu bar* from the *New* menu, just like we did with the main form in Exercise A. You will now be in the menu bar properties screen.

- **Step 3 – Add a Menu Category**: Now, create a menu category by tapping the *MENU* silk screen button. Select *New* from the *Edit* menu.

- **Step 4 – Update Menu Category Properties**: We want the menu category to be called “Options”, so we need to change the menu category properties. Update the properties as follows.

    - change the name from "New Menu" to "Options"
    - dhange the *left* field to 4
    - tap the *Calc Width* button.
    - tap the *Apply* button to accept these changes.

  The screen should now look something like the following.



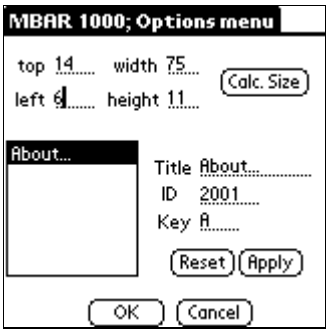
- **Step 5 – Open the “Options” Menu Category**: In order to add menu items to the menu category, we must go to the menu category details screen. To do this, select the "Options" menu category that we just created, and tap the *Open* button.

- **Step 6 – Add a New Menu Item Resource**: Now, let's add the "About…" menu item to the "Options" menu category. Tap the *MENU* silk screen button, and select *New* from the *Edit* menu.

- **Step 7 – Update the Menu Item Properties**: Update the menu item properties as follows.

    - change the name from "New Menu" to "About…"
    - change the top field to 14
    - change the *left* field to 6
    - tap the *Calc Size* button
    - give the selection a unique identifier by changing the *ID* field to 2001
    - let's make this menu option available with a command-A keystroke by filling in the *Key* field with an "A"
    - tap the *Apply* button to accept these changes

The screen should now look something like the following.



- **Step 8 – Save and Exit**: Tap the *OK* button to accept these changes and return to the menu bar properties screen. From there, tap the *OK* button to return to the familiar resource list screen.

- **Step 9 – Update the Menu Bar Resource ID**: On the resource list screen, select the "MBAR 1000" entry, and change the resource *ID* associated with it to "2000" so it is unique, and then tap apply.

Add the Menu Bar ID to the Main Form

We now need to tell the main from about it's new menu. The following steps outline this process.

- **Step 1 - Open the Main Form**: Tap the main form resource, "tFRM1001", and then tap the *Open* button.

- **Step 2 - Update the Menu Bar ID Field**: Change the *MBar ID* field to "2000" to associate the new menu bar with this form.

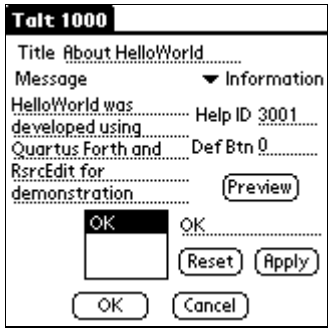
- **Step 3 - Save and Exit**: Tap the *OK* button to save and exit back to the resource list.

Add the Alert Box Resource

Now, we will create the "About HelloWorld" pop-up window.

- **Step 1 – Create the Alert Box Resource**: Tap the *MENU* silk screen button. Select *Alert* from the *New* menu, just like we did for the menu bar earlier. You will now be in the alert box properties screen.

- **Step 2 – Update the Alert Box Properties**: Update the alert box properties as follows.
    - fill in the *Title* field with "About HelloWorld"
    - type in the message we wish to display in the *Message* field (see screen capture below, or create your own display string)
    - change the *Help ID* field to a unique identifier, "3001"
- **Step 3 – Add an OK Button to the Alert Box**: Now, we need to add an OK button that will allow us to dismiss this alert box. Tap the *MENU* silk screen button and select *New* from the *Edit* menu.
- **Step 4 – Update the OK Button Properties**: Change the name from "Button" to "OK", and tap the *Apply* button to accept these changes. The screen should now look something like the following.



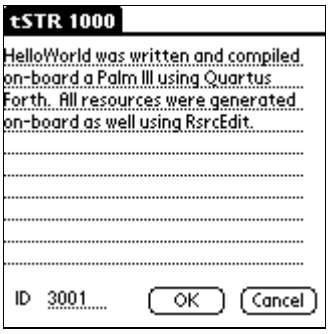
- **Step 5 – Save and Exit**: Tap the *OK* button to accept these changes and return to the resource list screen.
- **Step 6 – Update the Alert Box Resource ID**: Change the *ID* of our alert box to "3000", just like we did for the menu bar.

Add the Help String Resource

Now, we want to add an information string that is displayed when the "i" icon is tapped in top right-hand corner of the alert box. The following steps outline this process.

- **Step 1 – Create a String Resource**: Tap the *MENU* silk screen button. Select *String* from the *New* menu.
- **Step 2 – Fill in the Help Screen String**: Fill in the help screen string (see the screen capture below, or create your own string)

- **Step 3 – Update the String Resource ID**: Change the *ID* to "3001". The screen should now look like the following.



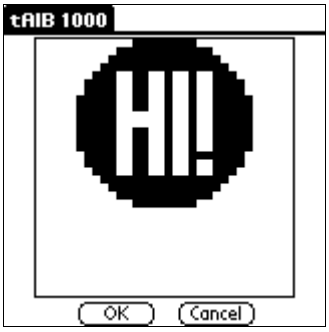
- **Step 4 – Save and Exit**: Tap the *OK* button to save the string resource and return to the resource list screen.

Create the Icon Bitmap

We now need to create an icon bitmap that will be associated with our standalone application. This step is optional. If no bitmap is created, the default icon will be used for our application.

- **Step 1 - Create the Icon Bitmap Resource**: To do this, select *Icon* from the *New* menu of the resource list screen of RsrcEdit.

- **Step 2 - Draw the Bitmap**: This is a 32X32 drawing area. The bottom 10 pixels are used by the program name, so anything drawn in these rows will not be displayed in the icon. A sample bitmap is shown below.



- **Step 3 – Save the Bitmap Resource**: Tap the *OK* button to return to the resource list screen.

- **Step 4 – Save All New Resources**: From the resource list screen, tap the *Done* button to save all of our changes to the database.

Now, let's check out the source code changes required to drive these new resources.

Forth Code for HelloWorld - Exercise B

The Forth code changes for this application are almost as simple as the original program.

- **Step 1 – Copy the Source File**: Copy the original source in “hello-a” to a new memo and change “hello-a” to “hello-b”. While we are at it, change the creator id from “hiwa” to “hiwb”.

- **Step 2 – Include Necessary Files**: Now that we will be dealing with events, we need to include the "Events" library file. In the section where we are including other library files, add the following.

```
needs Events
```

- **Step 3 – Constant Definitions**: We need to add in some constant definitions that correspond to the new resource id values defined above. If for some reason you used different numbers as identifiers, it must be reflected here. The constant definitions for the new resources are as follows.

```
1001 constant HelloForm
2001 constant AboutMenuItem
3000 constant AboutBox
```

- **Step 4 – Handle New Events**: Now our new code needs to check to see if the event on stack from *ekey* is a *menuEvent*. If so, it should verify that it is actually the menu entry we defined in our resources. If it is, it should put the “AboutBox” resource id on the stack and invoke *FrmAlert*, which will pop up our alert box. The following word, *do-event* handles this functionality.

```
: do-event ( ekey -- )
  menuEvent = if
    event >abs itemid
      AboutMenuItem = if
      AboutBox FrmAlert drop
    then
  then ;
```

- **Step 5 – Update the Main Loop**: Now we just need to update the code in*go* to deal with the new menu event. In our original HelloWorld exercise, we used the following line to keep the program running.

```
begin ekey drop again
```

In this original loop, if an event (see Events in the Forth library files) occurred, *ekey* put that event on the stack. Otherwise, *ekey* put *nilEvent* on the stack indicating that no event had occurred. However, because our original application had nothing to do, the *eventType* was just dropped, regardless of what it was. Now, instead of dropping the event, we want to invoke the new *do-event* word. So, in our main loop, just change *drop* to *do-event*.

The entire source listing is shown below. The changes from "hello-a" are shown in bold.

```
\ hello-b 2/1/99 3:47 pm -  SCD

needs ids
needs resources
needs Events

(ID) hiwb (ID) resc  use-resources

1000 constant HelloForm
2001 constant AboutMenuItem
3000 constant AboutBox

: show-panel ( -- )
  HelloForm ShowForm ;

: do-event ( ekey -- )
  menuEvent = if
    event >abs itemid
      AboutMenuItem = if
      AboutBox FrmAlert drop
    then
  then ;

\ Main entry point:
: go  show-panel
  begin  ekey do-event again ;
```

Test – Exercise B

We are now ready to try out our new menu event. Enter Quartus Forth and type the following:

```
include hello-b
go
```

If everything was done correctly, we should now be able to tap the *MENU* silk button and see our menu. Alternatively, we could use the command-A keystroke to invoke our new about box.

Create a Standalone Executable for HelloWorld - Exercise B

Now that everything tests out okay, we can generate a standalone executable thatcan be launched outside of Quartus Forth from the application launcher. This section is only applicable to registered users of Quartus Forth. The evaluation version does not generate standalone applications.

The main command for creating a stand-alone executable is *MakePRC*. *MakePRC* reads code from a specified execution token, and recursively extracts all required supporting code into a new PRC with the name provided. In our example, the main word we use to start our program, *go*, so we will use the execution token for *go* with *MakePRC*.

Remember: Each PRC that is released to the public must have a unique creator ID. See the section, A Note About Creator IDs, above for more information.

Create the Make File

We will now create new memo file that will allow us to generate a standalone executable.

- **Step 1 – Create a New Forth File**: In order for us to make a standalone executable, we need to create a new memo. Call it "make-hw".

```
\ make-hw 2/4/99 2:14 pm – SCD
```

- **Step 2 - Include Necessary Files**: This file should include "hello-b" because it contains the main word *go* that we will be using with *MakePRC*. We also include "ids"

because we have to specify resource and creator IDs in this file. This is done with the following Forth lines.

```
needs hello-b
needs ids
```

- **Step 3 - Define Constants**: If you will recall, our "hello-b" file contains some constant definitions that correspond to certain resource IDs defined in the "hiwb" resource database. However, we didn't define constants for all of these resources because it wasn't necessary at the time. For the purposes of making a standalone executable, we will need to specify resources that we did not use in "hello-b". So, for consistency, we need to define a few more constants.

```
2000 constant MenuBar
3001 constant HelpString
1000 constant IconBitmap
```

- **Step 4 - MakePRC**: The next line in the file is the one that really does the work.

```
' go (id) Helo MakePRC Hello
```

I will go through each part of this line and attempt to explain what is happening here.

  - *' go* leaves *xt* (the execution token for the word "go") on the stack
  - *(id) Helo* puts the creator ID that we want our standalone app to have on the stack (see the note above concerning creator IDs)
  - these two stack parameters are used by *MakePRC* to create an application with the name, *Hello*

- **Step 4 - Copy the Resources**: The next several lines copy our resources from our resource database into the target executable PRC file. This code must be done after *MakePRC*.

```
HelloForm (id) tFRM copyrsrc
MenuBar (id) MBAR copyrsrc
AboutBox (id) Talt copyrsrc
HelpString (id) tSTR copyrsrc
IconBitmap (id) tAIB copyrsrc
```

Full Source Listing

The entire source listing for "make-hw" is listed below.

```
\ make-hw 2/4/99 2:14 pm - SCD

needs hello-b
needs ids

2000 constant MenuBar
3001 constant HelpString
1000 constant IconBitmap

' go (id) Helo MakePRC Hello

HelloForm (id) tFRM copyrsrc
MenuBar (id) MBAR copyrsrc
AboutBox (id) Talt copyrsrc
HelpString (id) tSTR copyrsrc
IconBitmap (id) tAIB copyrsrc
```

Create the Standalone Executable

Now that the make file is complete, executing the following steps should result in the creation of a standalone PRC file.

- **Step 1 – Enter Quartus Forth**: Launch Quartus Forth from the application launcher.
- **Step 2 – Include the Make File**: Just enter the following line to include the make file.

```
include make-hw
```

Quartus Forth should respond with a string of ".....", followed by "ok". That's it. If you return to your launcher, you should now see the Hello application.

Congratulations!

Congratulations on your first application developed completely on-board! The next time you HotSync, this PRC file will be backed up to your backup directory. From there, it can be distributed to others as desired. However, remember that this is just a sample application because of the unofficial creator ID issue.

Now, let's move on to something a bit more challenging.

A More Advanced Application

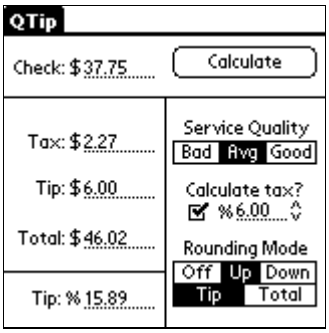
The next application we will develop will involve several more types of resources. Furthermore, this application will contain more functionality, which will require more Forth code. The focus of this section will be less on the resource creation, and more on the Forth code required to generate the application.

This application is a simple tip calculator that we will call “QTip”, the "Q" paying tribute to Quartus Forth. It will be developed over the course of several exercises. We will start with a very basic tip calculator with a few fields and a “calculate” button.

Each exercise will extend the functionality of this tip calculator. Some goals for the final application are:

- a basic tip calculator – calculate a specified percentage of the check total as tip, and calculate the new total
- the ability to include tax in the original check total, or calculate it based on a specified tax rate
- selectable service quality to determine tip percentage
- the ability to round, up or down, either the tip field or the total field
- a configurable preferences form that allows the user to specify the default modes of operation of the QTip interface (i.e. rounding mode, default tip percentages, default tax rate, etc.)

The final application should look something like the following.



To be continued…

This is where I am stopping for the first cut of this tutorial. I am well on my way to the first cut of QTip. I will update this tutorial and release new versions as I complete each exercise section.

Any constructive feedback on what I have done so far would be appreciated. I can be reached at –

steven.donahue@arris-i.com

or

sdonahue@mindspring.com