



Binary Runtime Environment for Wireless™

Creating a BREW™ Application from Scratch



QUALCOMM Incorporated
5775 Morehouse Drive
San Diego, CA. 92121-1714
U.S.A.

This manual was written for use with the BREW SDK™ for Windows. This manual and the BREW SDK software described in it are copyrighted, with all rights reserved. This manual and the BREW SDK software may not be copied, except as otherwise provided in your software license or as expressly permitted in writing by QUALCOMM Incorporated.

Copyright © 2002 QUALCOMM Incorporated

All Rights Reserved

Printed in the United States of America

All data and information contained in or disclosed by this document are confidential and proprietary information of QUALCOMM Incorporated, and all rights therein are expressly reserved. By accepting this material, the recipient agrees that this material and the information contained therein are held in confidence and in trust and will not be used, copied, reproduced in whole or in part, nor its contents revealed in any manner to others without the express written permission of QUALCOMM Incorporated.

Export of this technology may be controlled by the United States Government. Diversion contrary to U.S. law prohibited.

Binary Runtime Environment for Wireless, BREW, BREW SDK, TRUE BREW, BREWStone, CMX, MobileShop, Eudora, and PureVoice are trademarks of QUALCOMM Incorporated.

QUALCOMM is a registered trademark and registered service mark of QUALCOMM Incorporated.

Bluetooth is a trademark of Bluetooth SIG, Inc.

Microsoft, Windows, Visual Studio, and Sound Recorder are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Macintosh is a registered trademark of Apple Computer, Inc.

UNIX and X-Windows are trademarks of The Open Group.

Adobe, Acrobat, and Photoshop are trademarks of Adobe Systems Incorporated.

All trademarks and registered trademarks referenced herein are the property of their respective owners.

Creating a BREW™ Application from Scratch

80-D4369-1 A

October 23, 2002

Introduction 5

- The best way to use this manual 5
 - Prerequisites 5
 - A word about BREW versions 5
- BREW documentation 6
- Requesting new BREW features 7
- For more information 7

Overview of the BREW development process—from idea to Emulator 8

- Creating your project in Visual Studio 10
- Writing your BREW application 10
 - The MIF 11
 - Resources 12
- Testing your application on the Emulator 13

Tutorial: Creating Your First BREW Application 15

- Before you begin 15
- Your objective for MyFirstApp 15
- Creating the MyFirstApp project files 16
 - Initial source code for myfirstapp.c 23
- Creating resources for MyFirstApp 26
- Writing the source code for MyFirstApp 30
 - Adding #include statements 31
 - Adding the application-specific data structure 32
 - Adding function prototypes 33
 - Modifying the AEEClsCreateInstance function 33
 - Adding the MyFirstApp_InitAppData function 34
 - Adding the MyFirstApp_FreeAppData function 36
 - Modifying the MyFirstApp_HandleEvent function 36
 - Handling SUSPEND and RESUME events 39
 - Adding the CMyFirstApp_Move function 40
- Testing MyFirstApp on the Emulator 41

What's Next? 44

Learning BREW by example 44

Learning BREW using other resources 46

 BREW SDK documentation 46

 BREW Developer Extranet 46

 BREW training 46

 BREW Developer Conference 47

Index 48

Introduction

This guide walks you through the basic steps necessary to create your first Binary Runtime Environment for Wireless™ (BREW™) application using the tools provided in the BREW Software Development Kit (BREW SDK™). It also provides an overview of the BREW application development process, so you can understand how each step in the tutorial relates to the rest of the process.

The best way to use this manual

The best way to use this manual depends on how you like to work. In other words, it's entirely up to you. If you think you'll benefit by first gaining an understanding of the BREW development process before creating an application, start with [Overview of the BREW development process—from idea to Emulator](#) on page 8. If, on the other hand, you can't wait to roll up your sleeves and start cranking out your first BREW application, see [Tutorial: Creating Your First BREW Application](#) on page 15.

After completing the first two sections of this guide, proceed to [What's Next?](#) on page 44 to examine the sample applications included with the BREW SDK. Studying these applications can help you gain greater understanding of how various APIs and controls can be used to increase the functionalities of BREW applications.

Prerequisites

It is assumed that you already possess a working knowledge of C/C++ and the Microsoft Visual Studio development environment.

A word about BREW versions

This manual is essentially BREW version-independent. In instances where a feature described in this manual became available with a particular BREW software release, you will see an indicator similar to: **New in BREW x.x.**

BREW documentation

This guide is part of an information set that includes the following documents:

<i>BREW SDK User's Guide</i>	Introduces the components of the BREW SDK and their relationship to one another. The document also contains general instructions for developing BREW applications and for using the BREW Emulator.
<i>BREW API Reference</i>	Provides information about BREW Application Programming Interfaces (API), functions, and data structures needed to develop applications for BREW-enabled mobile platforms.
<i>BREW Sample Applications Guide</i> (New in BREW 2.0)	Describes several sample applications created with BREW that demonstrate the capabilities of BREW interfaces and controls commonly used to develop complex BREW applications.
<i>BREW Device Configurator Guide</i>	Describes how to use the BREW Device Configurator to create effective wireless devices for emulation by the Emulator.
<i>BREW Resource Editor Guide</i>	Describes how to use the BREW Resource Editor to create the text strings, images, and dialogs used by BREW applications.
<i>BREW MIF Editor Guide</i>	Describes how to use the BREW MIF Editor to create and modify Module Information Files (MIF)—a special type of BREW file that contains information about the classes and applets supported by a particular BREW module.
<i>BREW SDK Utilities Guide</i>	Describes how to use the utilities, such as the PureVoice Converter, 2Bit Tool, and NMEA Logger, included with the BREW SDK.
<i>BREW Compressed Image Authoring Guide</i>	Describes how to use the BREW Compressed Image Authoring Tool to create files for displaying and animating images in your applications.
<i>Creating a BREW Application from Scratch</i>	Provides a tutorial designed to guide new BREW application developers through the creation of their first BREW application. It also provides an overview of the components of the BREW SDK as they relate to application development. NOTE: This manual can be downloaded from the BREW Developer Extranet and the BREW web site.

Requesting new BREW features

Do you have ideas for features that would make the BREW SDK more valuable and useful to you? If so, send us email at brew-request@qualcomm.com. Each request is evaluated, and a member of the New Features Response Team will respond to your email.

For more information

Online information and support is available for BREW application developers. Please visit the BREW web site for details: www.qualcomm.com/brew/developer.

Overview of the BREW development process— from idea to Emulator

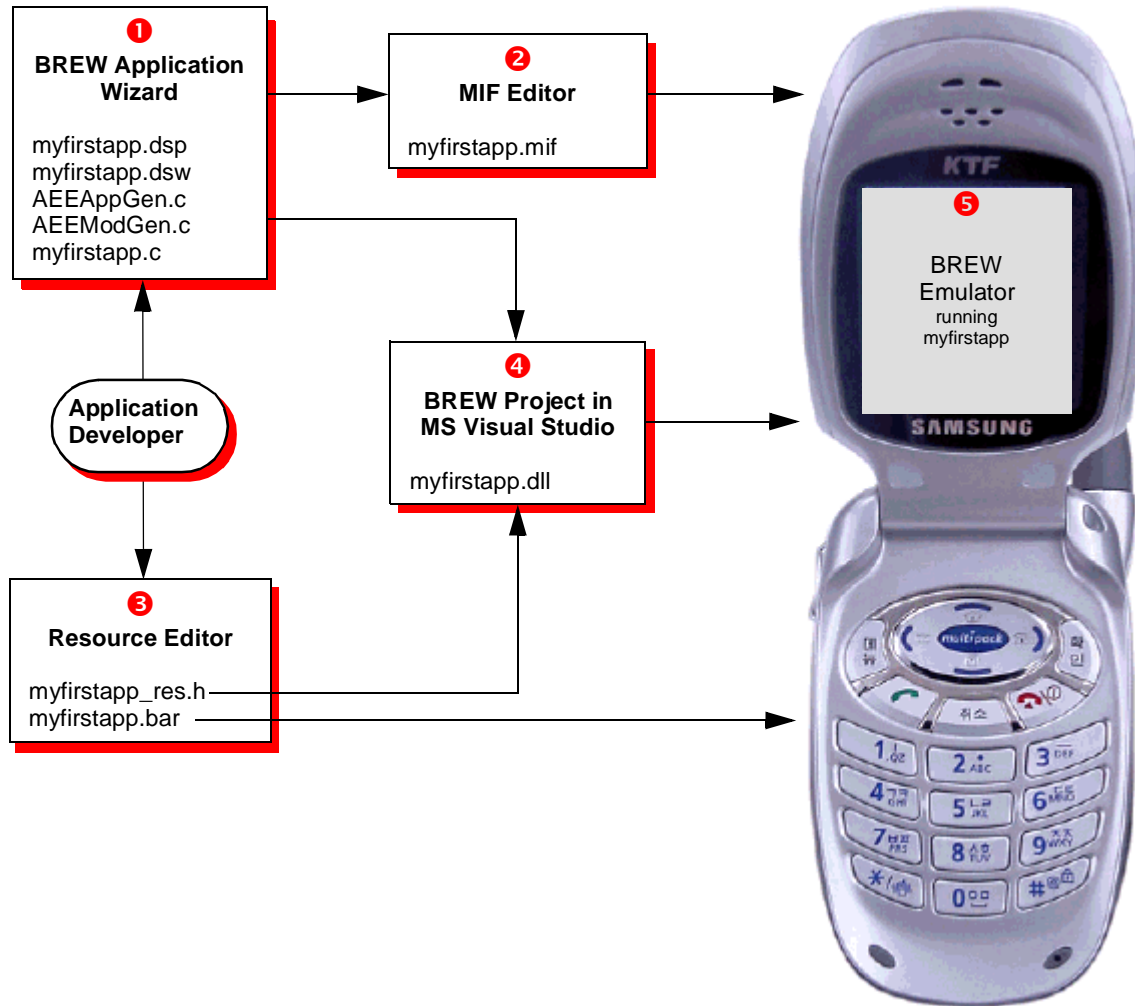
This section starts by offering a high-level view of the process of creating a BREW application, then examines each element more closely.

The BREW SDK is made up of several components, including the following:

- BREW Application Execution Environment (AEE) provides the foundation for BREW applications
- A set of tools that are the main development elements of the SDK (MIF Editor, Resource Editor, Emulator, and Device Configurator)
- BREW header files
- BREW utilities for adding functionality to your BREW applications
 - PureVoice Converter **(New in BREW 1.1)**
 - 2Bit Tool **(New in BREW 2.0)**
 - NMEA Logger **(New in BREW 2.0)**
- Add-ins to Microsoft Visual Studio to ease your application development
 - BREW Application Wizard add-in **(New in BREW 1.1)**
 - Automated ARM compiling add-in **(New in BREW 1.1)**
 - BREW Integrated Help add-in **(New in BREW 2.0)**
- Sample applications to demonstrate BREW capabilities
- Online help **(New in BREW 2.0)**

The following diagram shows how the pieces fit together, and identifies the files created by each tool for an application called MyFirstApp.

MyFirstApp



The process works like this:

- ❶ With your plan for the next killer application in mind, use the BREW Application Wizard, an add-in to Microsoft Visual Studio, to create the project file, workspace file, a skeletal source file, and two additional files (AEEAppGen.c and AEEModGen.c) that are essential to all BREW applications.
- ❷ Create a Module Information File (MIF) that includes all module- and applet-specific information required to load the applet.
- ❸ Create resources, such as strings, images, dialogs, and controls, to be used by the application.

- 4 Compile the source files generated by the BREW Application Wizard and the resource files to create the application DLL file.
- 5 Launch the BREW Emulator, specifying the MIF and Applet directories, and test your application.

NOTE: The process for moving a BREW application that has been tested on the Emulator to the actual handset is described in the tutorial section.

NOTE: Although the Configurator is an integral part of the BREW SDK, it is not discussed in this guide because users of the Configurator typically are device manufacturers. Application developers can also use the Configurator to change device settings to test their applications; however, that is beyond the scope of this guide. For information on the Configurator, see the *BREW Device Configurator Guide*.

Creating your project in Visual Studio

The BREW Application Wizard guides you through the tasks involved in creating a new BREW application project, including the creation of the following files:

- Project file (myfirstapp.dsp)
- Workspace file (myfirstapp.dsw)
- Application source file (myfirstapp.c)

NOTE: A sample of the source code generated by the BREW Application Wizard is shown on page 23.

Two other files required by BREW applications, AEEAppGen.c and AEEModGen.c, are included with the BREW SDK in the <BREW/Src> directory. These two files are automatically included in your new BREW project when you use the BREW Application Wizard.

Writing your BREW application

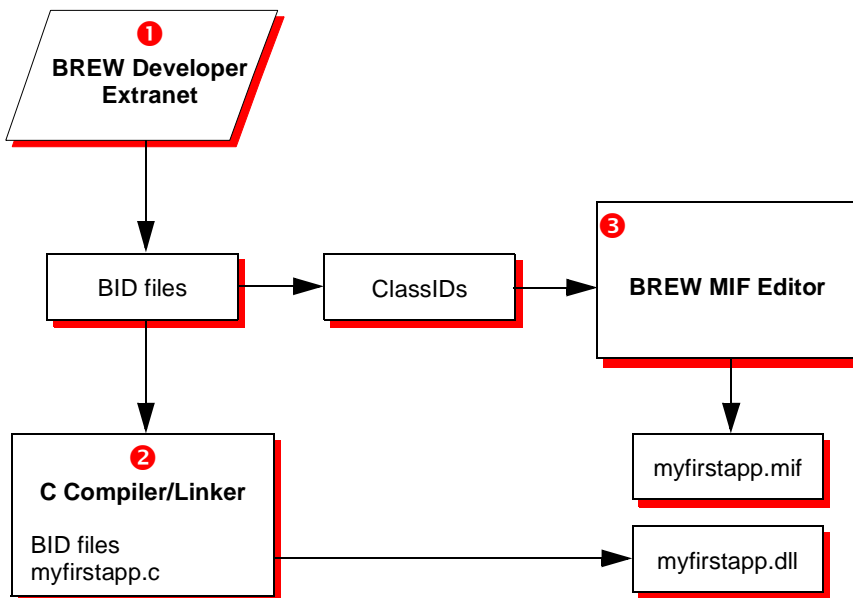
After creating your Visual Studio project and the skeletal application source file (myfirstapp.c), you can begin coding your first BREW application.

The MIF

If you were to immediately compile the application, you would receive one compilation warning and one error. These occur because all BREW applications require a MIF file containing unique BREW ClassIDs (CID) for each of the module's classes. After you create the MIF, the warning and error are removed.

The following diagram illustrates the steps you need to follow to create the MIF using the MIF Editor.

Creating the MIF



These are the steps required to create a MIF for a BREW module called myfirstapp:

- 1 From the BREW Developer Extranet, obtain a BID file for each class and applet that the module contains. Each BID file contains a unique, 32-bit ClassID that is stored in the MIF.
- 2 Compile and link the BID and application source files to create the application's DLL file (myfirstapp.dll).
- 3 In the MIF Editor, enter the required information, including the ClassIDs contained in the BID file(s), and save the MIF as myfirstapp.mif.

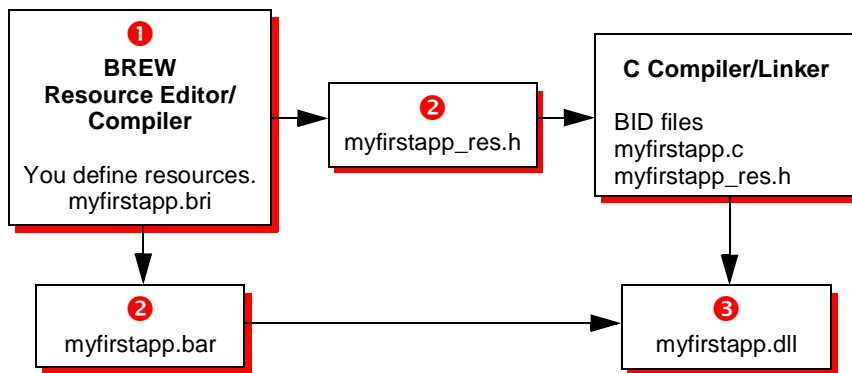
The MIF and DLL are used to test the application in the Emulator. This is discussed in [Testing your application on the Emulator](#) on page 13.

Resources

As you write MyFirstApp, you will also need to create resource files that contains character strings, images, and dialogs that are loaded at runtime. Resources are useful for storing language-specific strings, dialogs, and bitmaps. If you need to localize your application for use in a target device in a different language, you simply need to translate the resources in the resource file. It is not necessary to recompile the MOD or DLL file. So rather than hard coding controls into your application source code, consider using resources whenever possible.

The following diagram illustrates the steps you need to follow to create a resource file using the Resource Editor.

Creating resources



NOTE: The Resource Editor is made up of two integrated parts: the Resource Editor and Resource Compiler.

These are the steps required to create resource files for a BREW module:

- 1 Use the Resource Editor to create all character strings, images, and dialogs to be used in your application. The Resource Editor creates a BREW Resource Intermediate (BRI) file that contains all of the resource ID information for the application. As you create resources, you assign each a unique resource ID number.
- 2 The BRI file is then compiled by the Resource Compiler to produce the BREW Applet Resource (BAR) file and a resource header file, with the `_res.h` extension. The BAR file is used by the BREW executable file (MOD or DLL) at runtime to load resources.

- 3 When you compile your C source code, include the header file (`myfirstapp_res.h`) so that it is compiled and linked along with the other application source files to produce the DLL file.

Testing your application on the Emulator

After you have created a DLL and a MIF (and optionally, resource files) for your application, you can run it on the Emulator. The Emulator presents an image of a selected handset and emulates the running of your application as it will appear on the actual handset.

You can load your application in the Emulator by specifying the applet directory that contains the DLL. By default, the MIF directory is the same as the applet directory. When you run the Emulator, the BREW Application manager presents a series of applets contained in the applet directory. When you choose your application from the list, the Emulator executes the application's DLL, showing its initial screen on the device. If the application uses resources, it uses the resource-loading functions in the IShell interface to load the resources from the application's BAR file.

The Emulator, showing MyFirstApp in two different device configuration files



Tutorial: Creating Your First BREW Application

This section provides a step-by-step introduction to BREW application development. You will learn by doing. If you'd rather get some background on the process first, see [Overview of the BREW development process—from idea to Emulator](#) on page 8.

Before you begin

Before attempting to build your first BREW application, complete the following steps.

- Obtain Microsoft Visual Studio 6.0. Visual Studio.NET is not yet supported.
- Download and install the latest version of the BREW SDK. Here is the download site: <https://brewx.qualcomm.com/developer/sdk/download.jsp>.

NOTE: It is important that you install Microsoft Visual Studio before the BREW SDK. If you install the SDK first, BREW's add-ins to Visual Studio, including the BREW Application Wizard, will not be available.

Your objective for MyFirstApp

For your first taste of BREW, you will create a simple application, called MyFirstApp, that performs the following functions:

- It loads two text strings from a resource file that you define.
- It prints the strings in the upper half of the device screen.
- It draws a thin, color-filled, horizontal rectangle, dividing the screen into two halves.
- It loads a bitmap image, representing a cursor, from the resource file and draws the cursor in the middle of the bottom half of the screen.
- It allows the user to move the cursor bitmap using the up, down, left, and right arrow keys on the device.

Creating the MyFirstApp project files

The following instructions are designed to lead you through the creation of MyFirstApp. The tutorial begins with the BREW Application Wizard (**New in BREW 1.1**), describes the MIF Editor and Resource Editor, discusses the functions you need to define to meet the objectives stated above, and explains how you can compile and test the application. For more detailed instructions on using the BREW Application Wizard, or any of the other BREW SDK tools, see [BREW documentation](#) on page 6 for the list of documentation included with the BREW SDK.

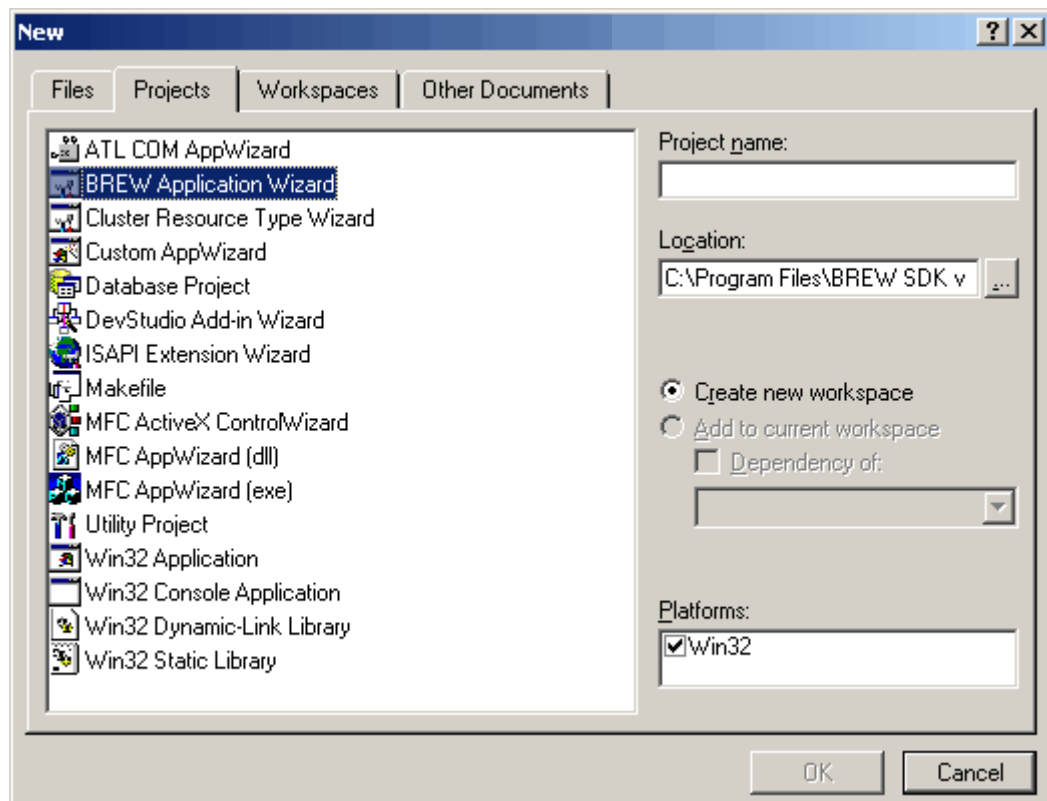
To create the MyFirstApp project with the Application Wizard

1. Start Microsoft Visual Studio, and choose **File > New**.

The New dialog box opens.

2. Click the **Projects** tab.

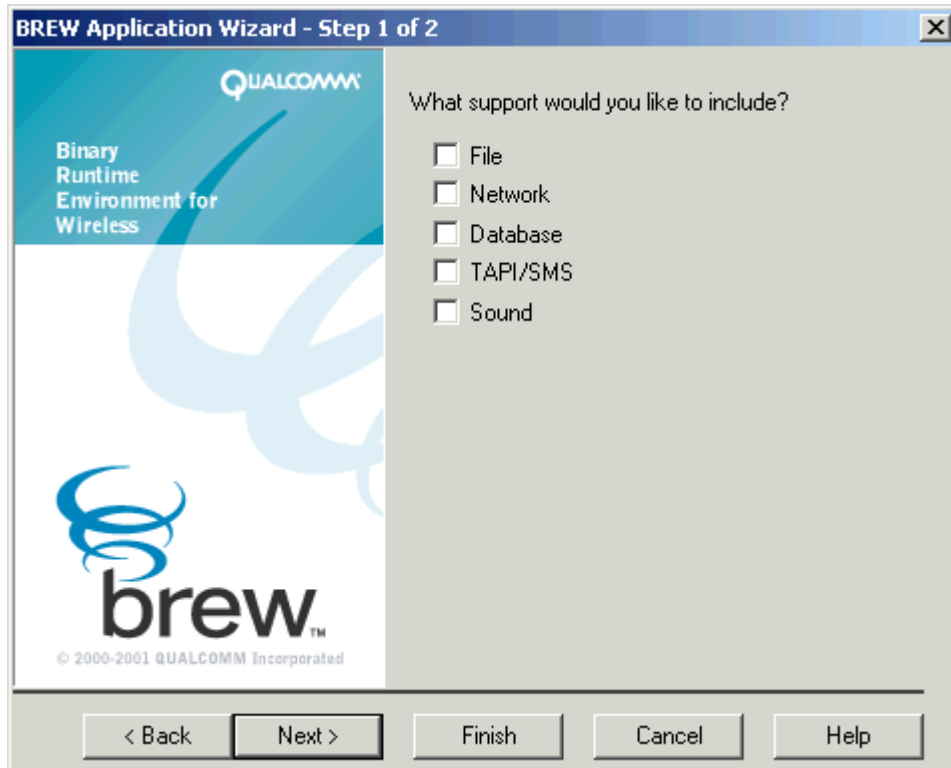
The following fields are shown.



3. Select BREW Application Wizard and do the following:
 - a. In Project name, type MyFirstApp.

- b. In Location, enter a path to the directory where you want to store the files for the project.
- c. Click **OK**.

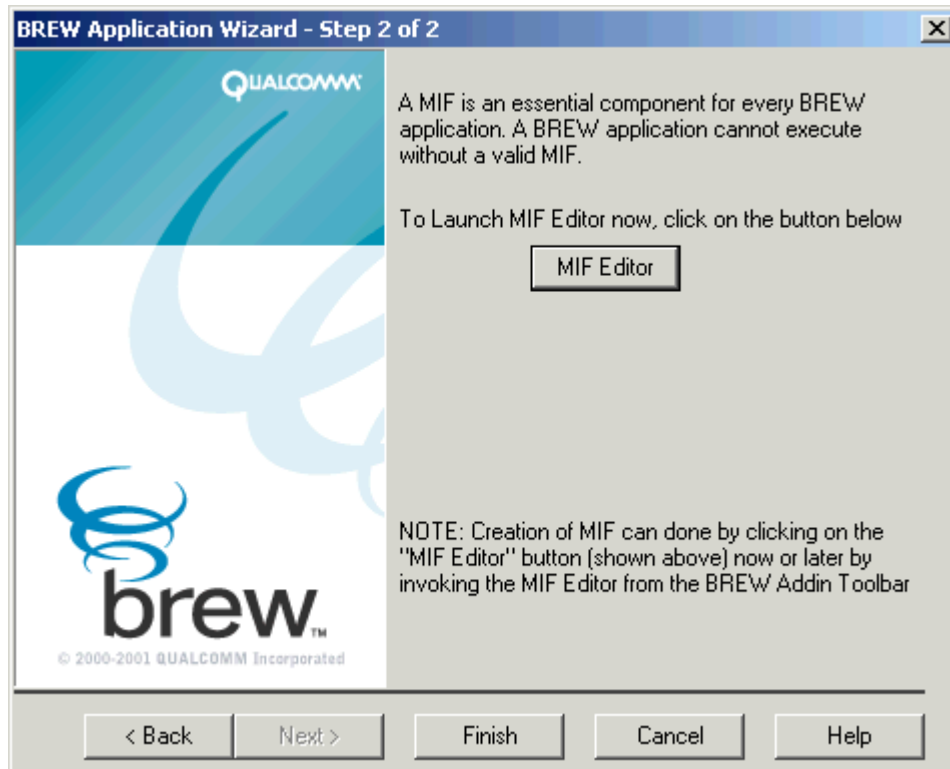
The first of two BREW Application Wizard windows opens.



This window allows you to define the types of interfaces you intend to use in your application.

- 4. Leave all of the check boxes unchecked, and click **Next**.

The second BREW Application Wizard window opens.



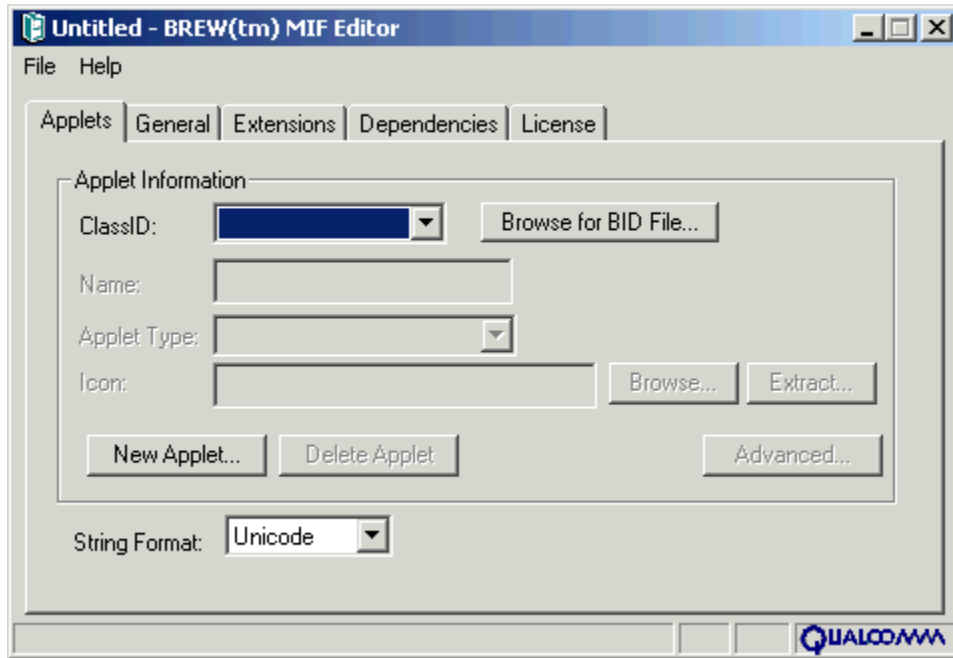
At this point, the experienced BREW developer has two options:

- Bypass the MIF Editor and complete the BREW Application Wizard to build the BREW application project file, leaving the creation of the MIF for a later time.
- Create the MIF now.

You have the option of skipping the MIF creation process now, but because the MIF is essential to all BREW applications, it is recommended that you create it before completing the BREW Application Wizard. For more information on the MIF, see [The MIF](#) on page 11.

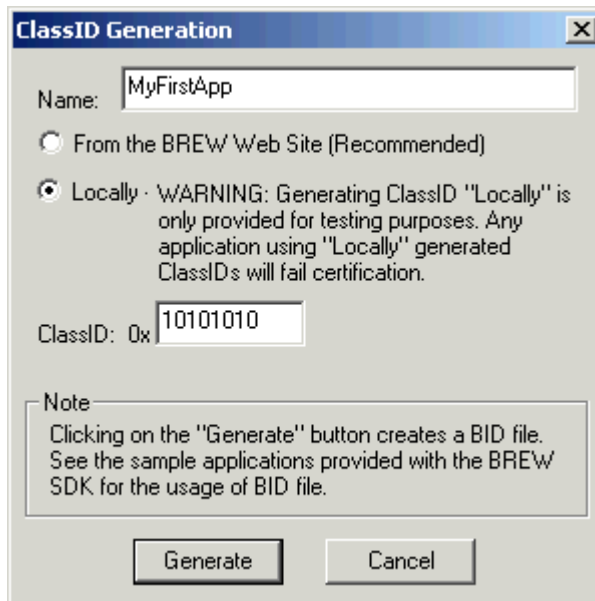
5. Click MIF Editor.

The BREW MIF Editor window opens.



6. Click **New Applet**.

The ClassID Generation dialog box opens.



7. Make the entries shown above.

NOTE: The name you assign the ClassID should be the same name you assigned the project in the BREW Application Wizard.

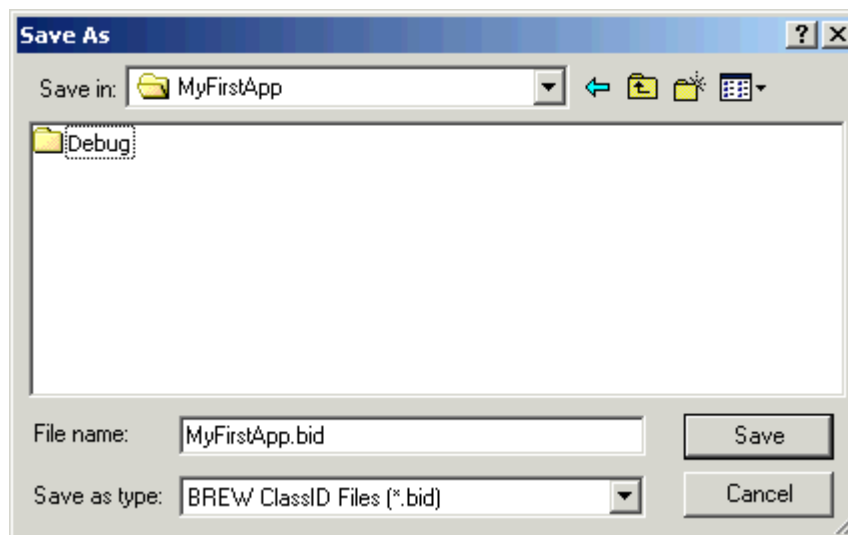
NOTE: The Locally option allows you to run your application for testing purposes. Before you can submit an application for TRUE BREW™ Testing, you are required to obtain a unique, 32-bit BID for each BREW class you create, including BREW system classes and all applet and non-applet classes, from the BREW Developer Extranet. For more information, see the *BREW MIF Editor Guide*.

8. Click **Generate**.

A confirmation box asks if you are sure you want to generate the ClassID locally.

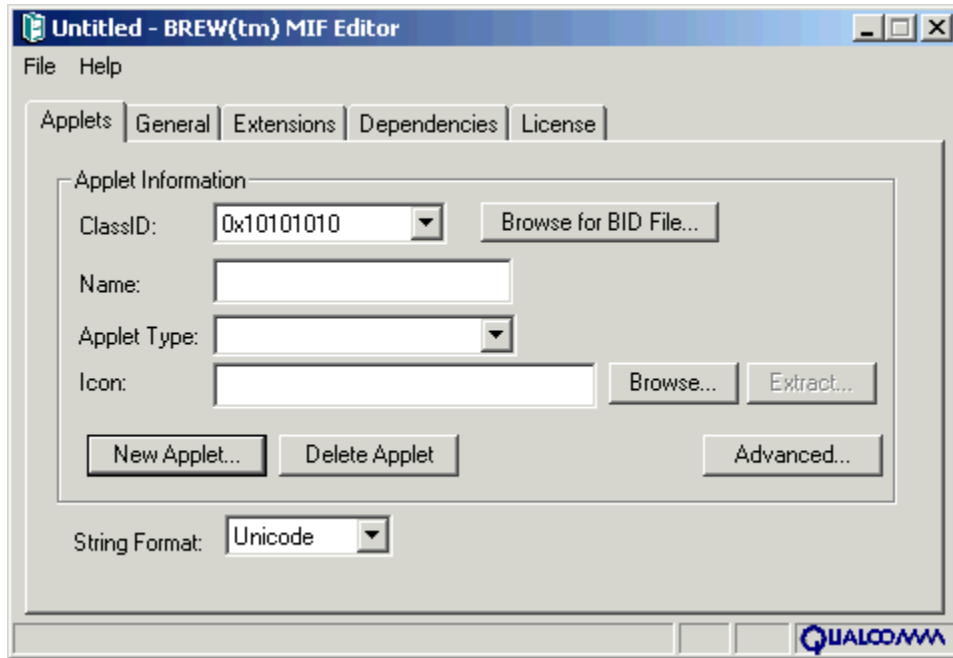
9. Click **Yes**.

The Save As dialog box opens, prompting you to specify a location for the myfirstapp.bid file.



10. Navigate to the <BREW/Examples/MyFirstApp> directory and click **Save**.

The ClassID you defined is added to the main MIF Editor window, as shown below.



11. In Name, type `MyFirstApp`. This name appears on the device display when the user chooses the applet from the device's applet menu.

NOTE: You must always ensure that the MIF filename is not made up exclusively of numbers, such as `1003.mif`. The name of the MIF must contain, and must begin with, alphabetical characters. Examples of valid MIF names are `helloworld.mif`, `b2b.mif`, and `good4u.mif`.

12. In the Applet Type combo box, select **Tools**.

NOTE: Applets can be grouped by applet type on the device's applet menu and on the carrier web site from which the user downloads applications for purchase. The supported applet types are Game, Tools, PIM, Web Applet, and Hidden. Hidden applets do not appear in the Application Manager on the device. They are activated when certain circumstances occur. A screen saver, for example, is a hidden applet. It is activated only after a predefined amount of inactivity occurs.

NOTE: For the purposes of `MyFirstApp`, leave the Icon field blank. This field allows you to specify the path and filename for a medium-sized graphic image (approximately 26x26 pixels) you want to use in the MIF. When you leave the field blank, a default image is supplied by BREW.

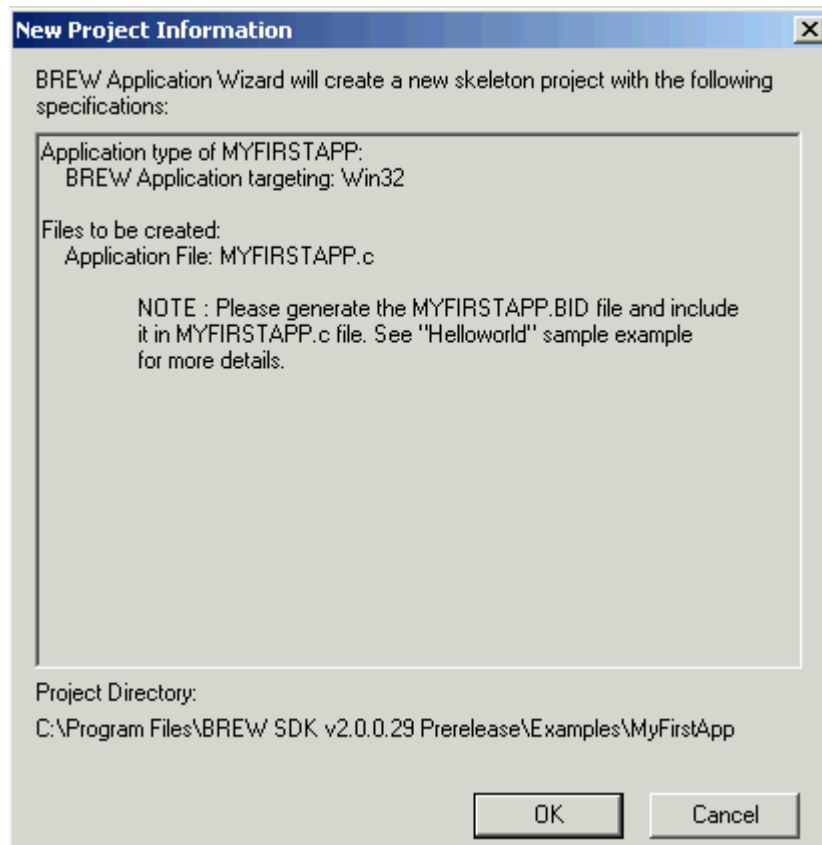
13. For `MyFirstApp`, it is not necessary to enter information on any of the other MIF Editor tabs. Choose **File > Save As**, assign the MIF the `myfirstapp` filename, and click **Save**.

The Save As dialog box closes.

14. In the MIF Editor window, click the Windows Close button.

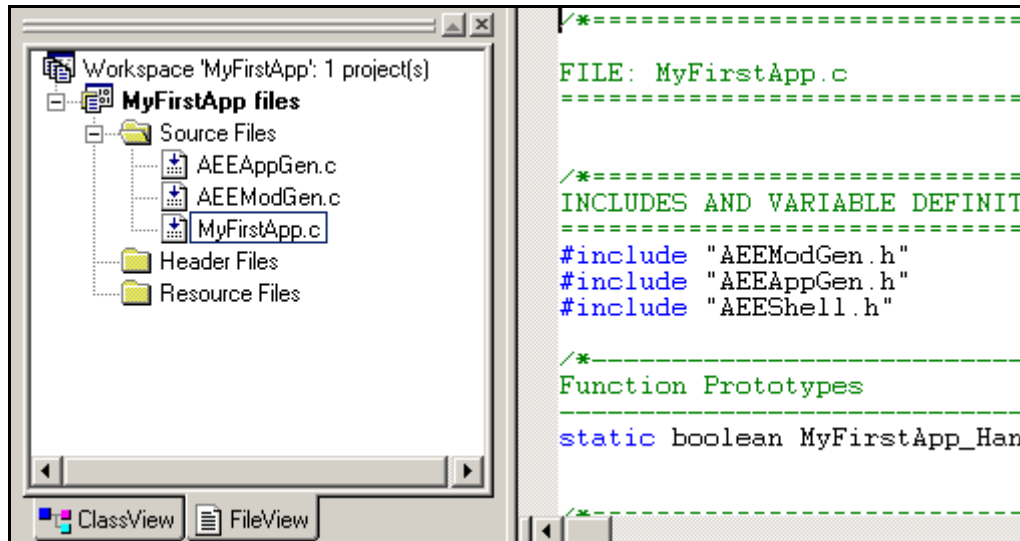
15. In the BREW Application Wizard window, click **Finish**.

The New Project Information dialog box opens.



16. Click **OK**.

The Microsoft Visual C++ window reopens with your new BREW application project loaded in the workspace.



The Application Wizard has created the following files:

- Project file (myfirstapp.dsp)
- Workspace file (myfirstapp.dsw)
- Application source file (myfirstapp.c)

NOTE: Two other files required by BREW applications, AEEAppGen.c and AEEModGen.c, are included with the BREW SDK in the BREW/Src directory. These two files are automatically included in your new BREW project.

17. Look over the source code for MyFirstApp and familiarize yourself with the information that was automatically generated by the BREW Application Wizard.
18. When you're ready to move on, proceed to [Creating resources for MyFirstApp](#) on page 26.

Initial source code for myfirstapp.c

When you have run the BREW Application Wizard, your source code looks like the following.

```

/*=====
FILE: myfirstapp.c
=====*/

/*=====
INCLUDES AND VARIABLE DEFINITIONS
===== */

```

```
#include "AEModGen.h" // Module interface definitions
#include "AEEAppGen.h" // Applet interface definitions
#include "AEEShell.h" // Shell interface definitions

/*-----
Function Prototypes
-----*/
static boolean MyFirstApp_HandleEvent(IApplet * pi, AEEEvent eCode,
                                     uint16 wParam, uint32 dwParam);

/*=====
FUNCTION DEFINITIONS
===== */

/*=====

FUNCTION: AEEClsCreateInstance
```

DESCRIPTION

This function is invoked while the app is being loaded. All Modules must provide this function. Ensure to retain the same name and parameters for this function. In here, the module must verify the ClassID and then invoke the AEEApplet_New() function that has been provided in AEEAppGen.c.

After invoking AEEApplet_New(), this function can do app specific initialization. In this example, a generic structure is provided so that app developers need not change app specific initialization section every time except for a call to IDisplay_InitAppData(). This is done as follows: InitAppData() is called to initialize AppletData instance. It is app developers responsibility to fill-in app data initialization code of InitAppData(). App developer is also responsible to release memory allocated for data contained in AppletData -- this can be done in IDisplay_FreeAppData().

PROTOTYPE:

```
int AEEClsCreateInstance(AEECLSID ClsId, IShell * pIShell, IModule * po,
void ** ppObj)
```

PARAMETERS:

clsID: [in]: Specifies the ClassID of the applet which is being loaded

pIShell: [in]: Contains pointer to the IShell object.

pIModule: [in]: Contains pointer to the IModule object to the current module to which this app belongs

ppObj: [out]: On return, *ppObj must point to a valid IApplet structure. Allocation of memory for this structure and initializing the base data members is done by AEEApplet_New().

DEPENDENCIES

none

RETURN VALUE

AEE_SUCCESS: If the app needs to be loaded and if AEEApplet_New() invocation was successful
EFAILED: If the app does not need to be loaded or if errors occurred in AEEApplet_New(). If this function returns FALSE, the app will not be loaded.

SIDE EFFECTS

none

```

=====*/
int AEEClsCreateInstance(AEECLSID ClsId,IShell * pIShell,IModule * po,void
    ** ppObj)
{
    *ppObj = NULL;

    if(ClsId == AEECLSID_MYFIRSTAPP){
        if(AEEApplet_New(sizeof(AEEApplet),ClsId,pIShell,po,(IApplet**)
            ppObj,(AEEHANDLER)MyFirstApp_HandleEvent,NULL)
            == TRUE)
        {
            // Add your code here .....

            return (AEE_SUCCESS);
        }
    }
    return (EFAILED);
}

/*=====

```

FUNCTION MyFirstApp_HandleEvent

DESCRIPTION

This is the EventHandler for this app. All events to this app are handled in this function. All APPs must supply an Event Handler.

PROTOTYPE:

```
boolean MyFirstApp_HandleEvent(IApplet * pi, AEEEvent eCode, uint16
wParam, uint32 dwParam)
```

PARAMETERS:

pi: Pointer to the AEEApplet structure. This structure contains information specific to this applet. It was initialized during the AEEClsCreateInstance() function.

ecode: Specifies the Event sent to this applet

wParam, dwParam: Event specific data.

DEPENDENCIES

none

```
RETURN VALUE
    TRUE: If the app has processed the event
    FALSE: If the app did not process the event

SIDE EFFECTS
    none
=====*/

static boolean MyFirstApp_HandleEvent(IApplet * pi, AEEEvent eCode, uint16
wParam, uint32 dwParam)
{
    switch (eCode)
    {
        case EVT_APP_START:

            // Add your code here .....

            return(TRUE);
        case EVT_APP_STOP:

            // Add your code here .....

            return TRUE;
        default:
            break;
    }
    return FALSE;
}
```

Creating resources for MyFirstApp

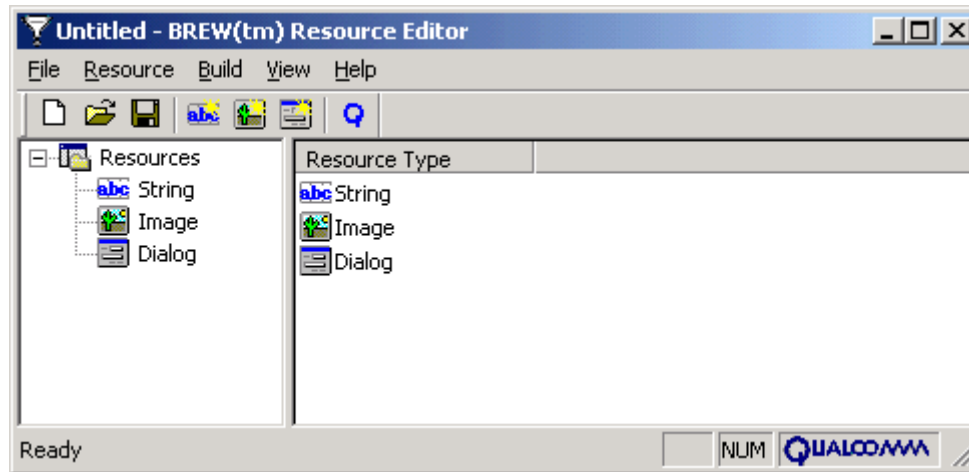
One of the objectives for the creation of MyFirstApp (see page 15) is to create a resource file that contains two text strings and a cursor image. You won't always know up front all of the resources that a given application requires. In general, you will define the resources you know about in the BRI file and, using the Resource Editor Compiler, compile them into the BAR file that is used at runtime by the application DLL whenever resources are required. Then, as you define more controls for the application, you can add them to the BRI and compile when necessary.

NOTE: If you have already defined resources and compiled them into the BAR file, and then modify those resources, it is not necessary to recompile your application source code to use those resources. If, however, you add new controls to the resource file (that is, if you change the -res.h file), you must recompile the application source code before they can be used by the application.

To define the string resources for MyFirstApp

1. Choose **Start > Programs > BREW > BREW Resource Editor**.

The BREW Resource Editor window opens.



2. Choose **Resource > New String**.

NOTE: You can also right-click on String in either of the window panes, and choose **New String**.

The String Resource dialog box opens.



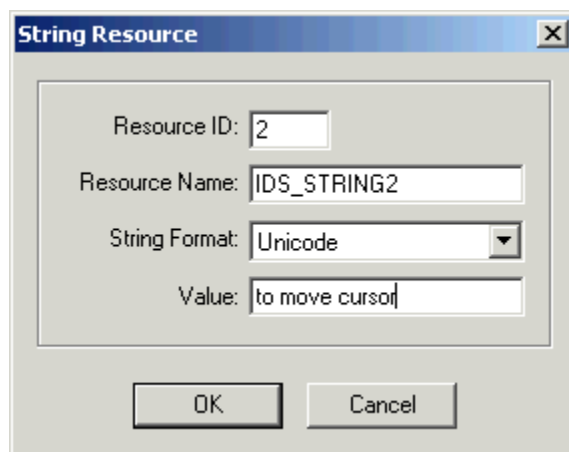
3. Make the entries shown above.

A Resource ID number between 1 and 5000 is assigned automatically; however, you can change it if you want to assign a specific ID. The important thing to remember is that the ID must be a unique integer among string resources within the same resource file (BRI). So you can assign Resource ID 1 to two string controls as long as you do not use them in the same BRI file.

The Value field contains the actual string content for the resource.

NOTE: Depending on the fonts supported by your target handset, you may need to change the entry in the String Format field.

- Repeat steps 2 and 3 to add your second string, making the entries shown below.



- Click **OK**.

The String Resource dialog box closes, and the string resources you created appear in the right pane of the BREW Resource Editor window.

ID	Name	Type	Value
1	IDS_STRING1	Unicode	Use arrow keys
2	IDS_STRING2	Unicode	to move cursor

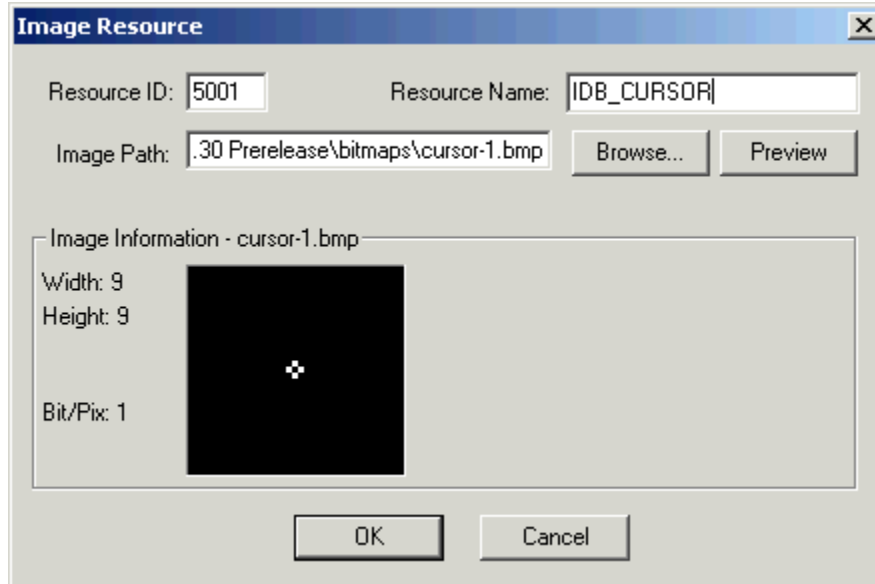
- Now you're ready to create your image resource.

To define the image resource for MyFirstApp

- With the Resource Editor window open, choose **Resource > New Image**.

NOTE: You can also right-click on Image, and choose **New Image**.

The Image Resource dialog box opens.



2. Make the entries shown above.

NOTE: For the Image Path, click **Browse**, navigate to <BREW\Bitmaps> and load cursor-1.bmp.

3. Click **OK**.

The Image Resource dialog box closes, and the image resource you created appears in the right pane of the BREW Resource Editor window.

ID	Name	Image Type	Path To Image
1	IDB_CURSOR	image/bmp	C:\Program Files\BREW SDK v2.0.0.30 Prerelease\bitmaps\cursor-1.bmp

4. Click **File > Save As**.
5. In the Save As dialog box, navigate to <BREW\Examples\MyFirstApp> and save the file as myfirstapp.bri.
6. You are now ready to compile your BAR file.

To compile the BREW resource file

1. In the BREW Resource Editor window, choose **Build > Build QUALCOMM .BAR/.H files**.

A message box opens informing you that the resource (BAR) file and header (res_h) file have been created. They are stored in the same directory as the BRI file.

2. If you had not previously saved the intermediate files, the Save As dialog box would have opened, where you would have indicated where you wanted the BRI stored and so on.
3. You are now ready to create functions in the MyFirstApp source code.

Writing the source code for MyFirstApp

Now you have created your base project, created a MIF, and defined resources in a BAR file, you are ready to start writing the actual source code for MyFirstApp. As stated in the Introduction to this guide, it is assumed you are already proficient in C/C++ programming. So this section will not dwell on definitions of operators, expressions, and variables. Instead, it examines each of the functions you need to write to meet the objectives of this application, and points out ways that using BREW APIs and functions may differ from typical C/C++ usage.

Given the skeletal source code provided by the BREW Application Wizard (shown beginning on page 23) as a starting point, you will make the following additions and modifications to the base code:

- #include lines must be added in the header to accommodate additional interface definitions needed by MyFirstApp.
- Application-specific data structure must be defined.
- New function prototypes must be defined for the InitAppData, FreeAppData, and Move functions.
- AEECIsCreateInstance must be modified. This function is created by the BREW Application Wizard; however, you need to make some minor modifications to it.
- A function for initializing the application-specific data must be defined.
- A function for freeing application-specific data must be defined.
- An event handler must be modified for MyFirstApp.
- A helper function must be defined to facilitate movement of the cursor on the device screen.

When you have successfully added or modified these components in your source code, you will compile and test MyFirstApp.

NOTE: Throughout this tutorial, you are asked to make specific additions to your source code in Visual Studio. You can do this by typing the necessary lines in your code or, if you are viewing this document online, you can select the specified code using the Text Select Tool in the Adobe Acrobat Reader, and then copy and paste it into your code in Visual Studio. The latter method will save you time; however, please be sure to delete any of the callout numbers (like ❶) after you have pasted the lines into your code.

Adding #include statements

When you ran the BREW Application Wizard, it automatically generated the following #include statements in your source code:

```
/*=====
INCLUDES AND VARIABLE DEFINITIONS
===== */
#include "AEEModGen.h"           // Module interface definitions
#include "AEEAppGen.h"          // Applet interface definitions
#include "AEEShell.h"           // Shell interface definitions
```

As you can tell by looking at the comment lines, these three elements provide definitions for the IModule, IApplet, and IShell BREW interfaces. These three interfaces are used in all BREW applications.

When you are creating a BREW application, however, you will need to add other #include statements to handle the other BREW interfaces required by the application. You also need to add definitions for any global constants needed by the application.

To add #include lines

1. Start Microsoft Visual Studio and load the MyFirstApp workspace (myfirstapp.dsw).
2. Click the **FileView** tab and open myfirstapp.c in the workspace.
3. Scroll down to the INCLUDES AND VARIABLE DEFINITIONS section shown above, and add the following #include lines after the AEEShell.h line.

```
#include "myfirstapp_res.h"    // Resource ID definitions
#include "myfirstapp.bid"      // Applet ClassID
```

NOTE: The header file you generated using the Resource Editor, `myfirstapp_res.h`, has been added to get the resource IDs for the applet. The `myfirstapp.bid` file, you created using the MIF Editor, contains the definitions of the unique ClassIDs for the applet.

4. Save your work and proceed to the next procedure.

Adding the application-specific data structure

The application-specific data structure defines the structure used to store data that needs to be remembered throughout the life of `MyFirstApp`. The most important thing to remember when setting up this structure for BREW applications is that the first data member must always be an instance of type `AEEApplet`. This structure exposes the most commonly-used BREW interface pointers, such as `Shell`, `Module`, and `Display`.

To add the application-specific data structure

1. Scroll down in your source code, and insert a blank line between the Includes and Variable Definitions and the Function Prototypes section.
2. Type the following lines:

```
typedef struct _CMyFirstApp
{
    AEEApplet  a;      // Mandatory first AEEApplet data member
    int  m_cxWidth;   // Stores the device screen width
    int  m_cyHeight;  // Stores the device screen height
    int  m_nCursorX;  // Stores the cursor bitmap x coordinate
    int  m_nCursorY;  // Stores the cursor bitmap y coordinate
    IImage * m_pIImage; // IImage interface pointer
}CMyFirstApp;
```

NOTE: As you can see, this typedef meets the requirement that the `AEEApplet` data member is first. This requirement is related to an applet being dynamically downloaded.

3. Save your work and proceed to the next procedure.

Adding function prototypes

When you ran the BREW Application Wizard, it automatically generated the following function prototype for the event handler:

```
static boolean MyFirstApp_HandleEvent(IApplet * pi, AEEEvent eCode, uint16
    wParam, uint32 dwParam);
```

For MyFirstApp, you must add three new functional prototypes to support functions you will be adding to the code.

To add function prototypes to MyFirstApp

1. Scroll down in your source code, and type the following after the event handler functional prototype:

```
static boolean MyFirstApp_InitAppData(IApplet* pMe);
static void MyFirstApp_FreeAppData(IApplet* pMe);
static void CMyFirstApp_Move(CMyFirstApp * pMe, int xc, int yc);
```

2. Save your work and proceed to the next procedure.

Modifying the AEEClsCreateInstance function

The purpose of the AEEClsCreateInstance() function is to create an instance of the AEEApplet data structure for each applet or class supported by the module. The AEEApplet data structure is created by invoking the AEEApplet_New() function with the address of the applet specific handle event function.

The BREW Application Wizard provided you with the underlying structure of the AEEClsCreateInstance() function, with commented placeholders for you to type your application-specific code. The code is shown below.

```
int AEEClsCreateInstance(AEECLSID ClsId, IShell * pIShell, IModule * po, void
    ** ppObj)
{
    *ppObj = NULL;

    if(ClsId == AEECLSID_MYFIRSTAPP){
        ❶ if(AEEApplet_New(sizeof(AEEApplet), ClsId, pIShell, po, (IApplet**)
            ppObj, (AEEHANDLER)MyFirstApp_HandleEvent, NULL)
            == TRUE)
        {
            ❷ // Add your code here .....

            return (AEE_SUCCESS);
        }
    }
}
```

```
    }  
  }  
  return (EFAILED);  
}
```

To modify AEEClsCreateInstance for MyFirstApp

1. At **1**, make the following two changes:
 - a. Change the sizeof parameter in the AEEApplet_New function from AEEApplet to CMyFirstApp.
 - b. In the nested IF statement, replace NULL with the following:

```
((PFNFREEAPPDATA)MyFirstApp_FreeAppData)
```

This function frees the application data. It is registered with the applet framework when the applet is created (inside the AEEClsCreateInstance function), and is called by the application framework when the reference count of the applet reaches zero. This function must free all the application's allocated data. For example, if the application stored data when it was suspended and resumed, that data must be freed.

2. Replace the comment line **2** with the following line:

```
if(MyFirstApp_InitAppData((IApplet*)*ppObj) == TRUE)
```

This statement calls the MyFirstApp_InitAppData function which initializes all applet-specific data after the applet has loaded successfully.

NOTE: The BREW Application Wizard automatically generated this line: `if(ClsId == AEECLSID_MYFIRSTAPP)` based on the name you assigned to the ClassID in the MIF Editor. If you assigned a name to the ClassID that is different than the name of your BREW Application Wizard project, you will receive an error during compilation. In that case, you would need to edit this line of code so that it matched the BID file. Also, it is important to note that all characters in the ClassID name are automatically capitalized and spaces are replaced with underscores.

3. Save your work and proceed to the next procedure.

Adding the MyFirstApp_InitAppData function

Now that you have successfully set up AEEClsCreateInstance, it's time to set up the InitAppData function which is used to initialize applet-specific data.

To add the MyFirstApp_InitAppData function

1. Scroll down to the end of the code for the AEEClCreateInstance function and insert a blank line.
2. Type the following code:

```
static boolean MyFirstApp_InitAppData(IApplet* pi)
{
    AEEDeviceInfo di;

    CMyFirstApp * pMe = (CMyFirstApp*)pi;

    pMe->m_pIImage = NULL;

    ❶ if ((pMe->m_pIImage = ISHELL_LoadResImage(pMe->a.m_pIShell,
        MYFIRSTAPP_RES_FILE, IDB_CURSOR)) == NULL)

        return FALSE;

    ❷ ISHELL_GetDeviceInfo(pMe->a.m_pIShell, &di);

    ❸ pMe->m_cxWidth = di.cxScreen;
    pMe->m_cyHeight = di.cyScreen;

    ❹ pMe->m_nCursorX = pMe->m_cxWidth/2;
    pMe->m_nCursorY = pMe->m_cyHeight*2/3;

    return TRUE;
}
```

3. Take note of the following information:
 - ❶ This statement loads the image from the resource file and stores a pointer to the image in the m_pIImage data member.
 - ❷ This statement uses the ISHELL_GetDeviceInfo function to determine the width and height of the device screen, and stores those values in the applet data structure.
 - ❸ These two lines cache the width and height of the device screen.
 - ❹ These two lines initialize the coordinates of the initial cursor position on the screen. Based on the device screen dimensions, the cursor image is positioned at the halfway point way along the x-axis and 2/3 of the way along the y-axis.
4. Save your work and proceed to the next procedure.

Adding the MyFirstApp_FreeAppData function

Now you are ready to define the FreeAppData function, which frees application data stored in the applet data structure. This function is registered with the applet framework when the applet is created using the AEEClCreateInstance function. It is called by the application framework when the reference count of the applet reaches zero. At that point, FreeAppData must free all of the data allocated by the application.

NOTE: Do not call the FreeAppData function directly in your application.

To add the MyFirstApp_FreeAppData function

1. Scroll down to the end of the code for the MyFirstApp_InitAppData function and insert a blank line.
2. Type the following code:

```
static void MyFirstApp_FreeAppData(IApplet* pi)
{
    CMyFirstApp * pMe = (CMyFirstApp*)pi;

    if (pMe->m_pIImage != NULL)
    {
        IIMAGE_Release (pMe->m_pIImage);
        pMe->m_pIImage = NULL;
    }
}
```

3. Save your work and proceed to the next procedure.

Modifying the MyFirstApp_HandleEvent function

Like AEEClCreateInstance, the BREW Application Wizard provided you with the underlying structure of the IHandleEvent function, with commented placeholders for you to type your application-specific code. All BREW applications must have an event handler section. The code provided by the BREW Application Wizard is shown below.

```
static boolean MyFirstApp_HandleEvent(IApplet * pi, AEEEvent eCode, uint16
wParam, uint32 dwParam)
{
    1 switch (eCode)
    {
        2 case EVT_APP_START:           //Applet start event
          // Add your code here .....

        return(TRUE);
    }
```

```

3   case EVT_APP_STOP:
4       // Add your code here .....

       return TRUE;

5   default:
       break;
}
return FALSE;
}

```

To modify MyFirstApp_HandleEvent

- At ❶, insert the following lines:

```

AEERect rc;
AECHAR szBuf[30] = {0};

CMyFirstApp * pMe = (CMyFirstApp*)pi;

```

- At ❷, replace the comment placeholder with the following lines:

```

case EVT_APP_RESUME:
IDISPLAY_ClearScreen (pMe->a.m_pIDisplay); // Erase whole screen
// Load string "Use arrow keys" from resource file.
ISHELL_LoadResString(pMe->a.m_pIShell, MYFIRSTAPP_RES_FILE,
    IDS_STRING1, szBuf, sizeof (szBuf));
IDISPLAY_DrawText(pMe->a.m_pIDisplay, AEE_FONT_NORMAL, szBuf,
    -1, pMe->m_cxWidth/5, pMe->m_cyHeight/8, 0, 0);

// Load string "to move cursor" from resource file.
ISHELL_LoadResString(pMe->a.m_pIShell, MYFIRSTAPP_RES_FILE,
    IDS_STRING2, szBuf, sizeof (szBuf));
IDISPLAY_DrawText(pMe->a.m_pIDisplay, AEE_FONT_NORMAL, szBuf,
    -1, pMe->m_cxWidth/5, pMe->m_cyHeight/5, 0, 0);

// Initialize rectangle
SETAEEERECT (&rc, 0, pMe->m_cyHeight/2 - 2, pMe->m_cxWidth, 2);
IDISPLAY_DrawRect (pMe->a.m_pIDisplay, &rc, 0, 1,
    IDF_RECT_FILL);
IIMAGE_Draw (pMe->m_pIImage, pMe->m_nCursorX, pMe->m_nCursorY);

IDISPLAY_Update(pMe->a.m_pIDisplay);

```

- Take note of the following information:

This function	Does this
IDISPLAY_ClearScreen	This function erases the device screen.

This function	Does this
ISHELL_LoadResString	This statement occurs twice, each followed by an IDISPLAY_DrawText function. The LoadResString functions load the two resource strings from the application resource file using the IDS_STRING1 and ID_STRING2 string IDs. The strings are copied to a local AECHAR type string buffer, szBuf.
IDISPLAY_DrawText	The two DrawText functions, each following a LoadResString function, display the strings stored in the szBuf buffer at these locations on the screen: <ul style="list-style-type: none"> • IDS_STRING1 is displayed at the x and y coordinates of screen-width/5 and screen-height/8. This corresponds to approximately the middle of the top half of the screen. • IDS_STRING2 is displayed at the x and y coordinates of screen-width/5 and screen-height/5. This corresponds to the location on the screen directly beneath the first string <p>NOTE: These locations will vary depending on the screen dimensions of various devices.</p>
SETAEERECT	This function initializes a 2-pixel-wide rectangle object of type AEERect used to draw a thin horizontal line that divides the screen in half. The second and third parameters are the x- and y-coordinates of the rectangle's upper left-hand corner. The fourth and fifth parameters are the height and width of the rectangle.
IIMAGE_Draw	This function displays the cursor in the lower half of the device screen, using the coordinates initialized in the InitAppData function.
IDISPLAY_Update	This function flushes the text buffers onto the device screen, allowing the images to be displayed.

4. At ❸, insert the following lines:

```

case EVT_KEY:           // Process key-down event.
    switch (wParam)
    {
        case AVK_LEFT:
        case AVK_RIGHT:
            CMyFirstApp_Move(pMe, (wParam == AVK_RIGHT ? 1 : -1),
                0); break;
        case AVK_UP:
        case AVK_DOWN:
            CMyFirstApp_Move(pMe, 0, (wParam == AVK_UP ? -1 : 1));
            break;
    }

```

```
        default:  
            return(FALSE);  
    }
```

This code switches on the AVK_LEFT, AVK_RIGHT, AVK_UP, and AVK_DOWN key press subevents, and the CMyFirstApp_Move function is called to move the cursor in the direction corresponding to the key press.

5. At 4, delete the comment placeholder.
6. At 5, insert the following two lines:

```
case EVT_APP_SUSPEND  
    return(FALSE);
```

7. Save your work and proceed to the next procedure.

Handling SUSPEND and RESUME events

It is important to include the EVT_APP_SUSPEND and EVT_APP_RESUME events within your application's HandleEvent function so that your application knows how to handle events that might interrupt its operation. For example, when an SMS message or an incoming phone call is received, your application needs to know whether it should resume where it left off at the end of the call or if it should start over.

When BREW suspends an applet, it sends the EVT_APP_SUSPEND event. Depending on the type of application you are developing, you have a couple of options on how BREW should handle the event, as follows:

- You can set the applet to return TRUE to the event, indicating that the applet has processed the Suspend and has not been unloaded from memory. This is how MyFirstApp works.
- You can also have the applet return FALSE to the event, indicating that you do not want the applet to process the Suspend. When this occurs, BREW sends an EVT_APP_STOP event and the applet is removed from memory.

When you have finished creating MyFirstApp, you can test EVT_APP_SUSPEND and EVT_APP_RESUME on the Emulator by clicking **Tools > Settings** to open the Settings dialog box. When the dialog box opens, MyFirstApp returns TRUE to Suspend operation. When you click **Cancel** to dismiss the dialog box, MyFirstApp issues the EVT_APP_RESUME event and the cursor in the application remains in the same position you left it.

Adding the CMyFirstApp_Move function

CMyFirstApp_Move is a helper function that allows the user to move the cursor on the device screen. The function uses an applet data structure pointer and two integer variables corresponding to incremental movements of the cursor in the x and y directions. For more information on helper functions, see the *BREW API Reference*.

To add the CMyFirstApp_Move function

1. Scroll down to the end of the code for the MyFirstApp_IHandleEvent function and insert a blank line.
2. Type the following code:

```
static void CMyFirstApp_Move(CMyFirstApp * pMe, int xc, int yc)
{
    AEEImageInfo iInfo;
    int min, max;
    ❶ int x = pMe->m_nCursorX;
    int y = pMe->m_nCursorY;

    ❷ IIMAGE_GetInfo (pMe->m_pIImage, &iInfo);

    // Erase the previously displayed bitmap from the screen.
    ❸ IDISPLAY_EraseRgn (pMe->a.m_pIDisplay, x, y, iInfo.cx,
        iInfo.cy);

    x += xc;
    y += yc;

    // Delimit the x & y coordinated to lower half of the screen
    ❹ min = 0;
    max = pMe->m_cxWidth - iInfo.cx;
    x = ((x < min) ? (min) : (x > max) ? max : (x));

    min = pMe->m_cyHeight/2;
    max = pMe->m_cyHeight - iInfo.cy;
    y = ((y < min) ? (min) : (y > max) ? max : (y));

    // Draw cursor at new position.
    IIMAGE_Draw (pMe->m_pIImage, x, y);

    pMe->m_nCursorX = x; // Store new x coordinate
    pMe->m_nCursorY = y; // Store new y coordinate

    // Update display.
    ❺ IDISPLAY_Update(pMe->a.m_pIDisplay);
}
```

3. Take note of the following information:

- ❶ In these statements, two local variables (x and y) are initialized with the coordinates of the current cursor position.
- ❷ The IIMAGE_GetInfo function gets the image information.
- ❸ The IDISPLAY_EraseRgn function erases the region on the screen where the cursor was previously displayed.
- ❹ This series of lines use algorithms to evaluate if the new cursor coordinates are within the boundaries of the bottom half of the screen. The IIMAGE_Draw function displays the cursor at the new coordinates.
- ❺ This updates the local x and y variables with the new cursor position.

4. Save and your work, and compile the DLL.

NOTE: All you need to know about compiling the DLL is that it must be stored in the <BREW\Examples\MyFirstApp> directory.

You are now ready to test MyFirstApp on the Emulator.

Testing MyFirstApp on the Emulator

When you have completed all of the steps outlined in this section, it will FINALLY be time for you to see how your new application works on the Emulator. .

To test MyFirstApp on the Emulator

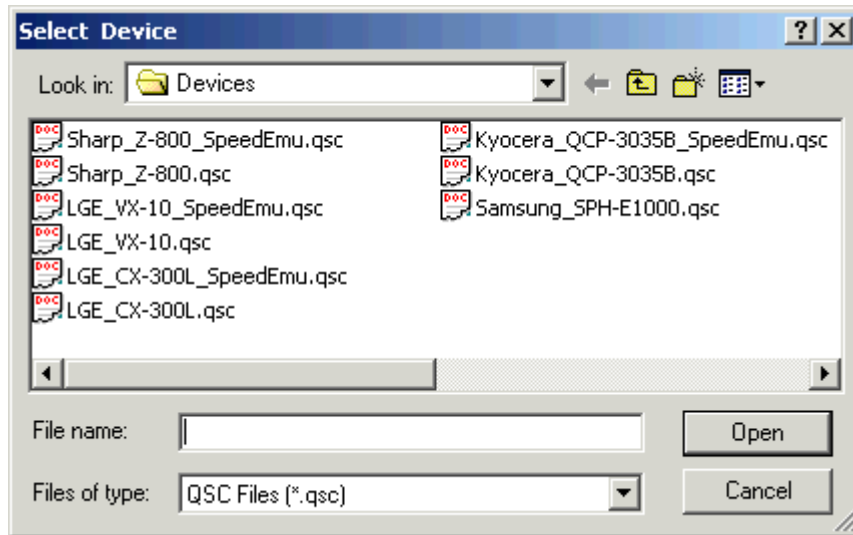
1. Choose **Start > Programs > BREW > BREW Emulator**.

The BREW Emulator runs.

2. Make sure that the Applet directory is set to <BREW\Examples>. If needed, choose **File > Change Applet Dir** to change the Applet directory.

3. If you want to view MyFirstApp on a different device image than the one currently displaying, choose **File > Load Device**.

The Select Device dialog box opens.



The Look in directory defaults to <BREWDDevices>.

4. Select the device you want to view in the Emulator and click **Open**.

The Select Device dialog box closes and the selected device image appears in the Emulator window.

5. Click the right or left arrow key until MyFirstApp is shown.

NOTE: The main image displayed in the screen is missing because you did not select an image when you created the MIF.

6. Click **Select**.

The Emulator displays a MyFirstApp application screen similar to the following:



7. Click the up, down, left, or right arrow keys to move the cursor around the bottom half of the screen.

Congratulations, you have successfully created a BREW application!

NOTE: Proceed to [What's Next?](#) on page 44 for information on ways that you can expand the capabilities of a BREW application and for methods you can use to learn more about BREW and the BREW SDK.

What's Next?

At this point, it is assumed you have successfully written MyFirstApp and tested it on the Emulator. With that goal accomplished, what's next? There are any number of ways that you can continue your education as a BREW application developer. The remainder of this section describes one suggested path, and offers a number of other resources you can use to learn about the BREW APIs and how to develop in the BREW development environment.

Learning BREW by example

You can learn a lot by examining the programming challenges faced by other BREW developers and observing how, within the boundaries of the BREW AEE, they overcame those challenges.

The <BREWExamples> directory in the BREW SDK contains several sample applications written by in-house QUALCOMM developers. The sample applications demonstrate the use of many interfaces and controls commonly used in BREW application development. Each sample application includes the source code, as well as the files you'll need to run them on the Emulator.

NOTE: A quick way to access the <BREWExamples> directory is to choose **Start > Programs > BREW > BREW Examples**.

The *BREW Sample Applications Guide* (**New in BREW 2.0**), accessed by clicking **Start > Programs > BREW > BREW SDK Docs > BREW Sample Applications Guide**, examines the following five sample BREW applications.

Sample application	Lets the user do this
Expense Tracker—a travel expense application	<ul style="list-style-type: none"> • Add new expenses on a transaction-by-transaction basis. • View expenses recorded for a specific date range, for a particular payment type or for all payment types. • Save expenses recorded for a particular time period for later viewing. • Load saved reports for viewing expenses recorded for a particular period. • View online instructions for using Expense Tracker.
MediaPlayer—a multimedia application	<ul style="list-style-type: none"> • Store up to 32 multimedia files. • Use standard multimedia controls, such as play, fast forward, rewind, stop, pause, and record. • Play a wide variety of audio, video, and still graphic formats including QCP, MP3, MIDI, BREW Compressed Image (BCI), BMP, and PMD, which contains both audio and video. • Record a QCP audio file using a microphone to emulate the device mouthpiece. • View online instructions for using MediaPlayer.
NetDiagnostics—a network diagnostics tool	<ul style="list-style-type: none"> • Specify parameters for performing echo tests on a device. • Test HTTP connection responses for a user-specified URL. • View online instructions for using NetDiagnostics.
Road Warrior—a web-access application	<ul style="list-style-type: none"> • Road Warrior performs only two simple tasks: pointing to the California Department of Transportation web server and reporting traffic speeds. It is important to point out, however, that Road Warrior demonstrates a real-life application that has passed TRUE BREW Testing and is being used in commercial BREW environments.
WhiteBoard—a drawing application	<ul style="list-style-type: none"> • Draw a wide variety of shapes, including lines, circles, rectangles, triangles, and polygons. • Draw filled or unfilled shapes with user-selected border and fill colors. • Erase a drawing at any time. • Clip a rectangular portion of a drawing. • Undo (or erase) the shape you are currently drawing. • View online instructions for using WhiteBoard.

In the *BREW Sample Application Guide*, read the descriptions of what these applications allow the handset user to do, try them out on the Emulator to see how they really work, and look at the source code for each. The source code is heavily commented to aid in your exploration.

NOTE: The *BREW Sample Application Guide* is new to BREW 2.0. BREW 1.1, however, provided a wide variety of sample and usage applications that you can explore in much the same way. For more information, see the BREW Sample Applications section of the *BREW SDK User's Guide* (in BREW SDK version 1.1).

Learning BREW using other resources

In addition to the valuable information you can find by exploring the sample applications included with the BREW SDK, other resources are available for learning more about BREW and developing BREW applications.

BREW SDK documentation

The documentation set of online (PDF) manuals provide a wealth of information about the BREW SDK and BREW. After you have completed the tutorial in this book and devoured the sample applications discussed in the *BREW Sample Applications Guide*, the next logical place to go is the *BREW SDK User's Guide* for an in-depth look at the concepts and inner workings of BREW. And once you have a firm grasp of coding in BREW, you'll find the *BREW API Reference* an invaluable tool. See [BREW documentation](#) on page 6 for a complete list of the documentation available with the BREW SDK.

BREW Developer Extranet

The BREW Developer Extranet provides online access to the BREW developer community. Depending on your level of membership in the BREW Developer Alliance Program, you can download tools and applications, get technical and operational support, and submit your own applications for testing via the BREW Developer Extranet.

BREW training

QUALCOMM offers a variety of training packages designed to benefit developers, managers, and trainers. For a complete list of the courses available and price lists, see <http://www.qualcomm.com/brew/developer/training/brewtraining.html>.

BREW Developer Conference

QUALCOMM hosts an annual BREW Developer Conference which attracts developers, carriers, OEMs, industry analysts, and media from around the world to San Diego to see and hear the latest developments of BREW. For BREW developers, the Conference is a great forum for learning about BREW directly from members of the QUALCOMM BREW Development Team, and by networking with other members of the BREW community.

Go to this link to see highlights of the 2002 BREW Developer Conference:

http://www.qualcomm.com/brew/news/events/devconf_2002/devconf_2002.html.

Symbols

#include statements, adding 31

A

AEAppGen source file 10

AEApplet data structure 33

AEEClsCreateInstance, modifying 33

AEEModGen source file 10

Applet

directory, changing on BREW Emulator 41

hidden 21

types, assigning to MIF 21

Application Wizard add-in

files created with 23

purpose of 9

source code for MyFirstApp 23

starting 16

Applications

creating project for 10

creating resources for 12, 26

creating your first 15

generated source files 23

supporting interfaces 17

testing on Emulator 13

writing 10

Application-specific data structure, adding 32

B

BAR files

compiling 29

creating 12

BID file, compiling 11

BRI file, creating 12

C

Character strings, creating for applications 12

ClassIDs

generating 19

generating locally 20

obtaining 11

Compiling

BID file 11

header file 12

MyFirstApp 41

resource files 29

Components of BREW SDK 8

Creating

application project 10

BAR file 12

BRI file 12

character strings, images, and dialogs 12

DLL file 12

MOD file 11

MyFirstApp project files 16

resources 12

resources in BREW applications 26

your first BREW application 15

D

Data structure, adding 32

Developer

Extranet 46

Development process overview 8

Device Configurator 10

Dialogs, creating for applications 12

DLL file, creating 12

Documentation

SDK 6, 46

Downloading

BREW SDK 15

DSP file, generated by Application Wizard 23

E

Emulator

starting 41

testing BREW applications 13

testing MyFirstApp 41

Event handling 39

Examples directory 44

Extranet, BREW Developer 46

F

Features, requesting new 7

FreeAppData function, adding 36

Function prototypes, adding 33

H

HandleEvent function, modifying [36](#)
Header file, compiling [12](#)
Hidden applets [21](#)

I

IDISPLAY
 _ClearScreen function [37](#)
 _DrawText function [38](#)
 _EraseRgn function [41](#)
 _Update function [38](#)
IIMAGE
 _Draw function [38](#), [41](#)
 _GetInfo function [41](#)
Image resources, defining [28](#)
Images, creating for applications [12](#)
Include statements, adding [31](#)
InitAppData function, adding [34](#)
Interfaces, support for in applications [17](#)
ISHELL
 _GetDeviceInfo function [35](#)
 _LoadResString function [38](#)

L

Local ClassIDs [20](#)
Localizing BREW applications [12](#)

M

Microsoft Visual Studio
 creating a project [10](#)
 obtaining [15](#)
 starting [16](#)
MIF Editor, starting [18](#)
MIF file
 assigning applet type to [21](#)
 role in SDK [11](#)
MOD file, creating [11](#)
MyFirstApp
 code generated by Application Wizard [23](#)
 development objective [15](#)
 writing source code for [30](#)

P

Prerequisites to BREW development [15](#)
Project, creating with Application Wizard [16](#)

R

Requesting new BREW features [7](#)
Resource Compiler [12](#)

Resource Editor
 role in SDK [12](#)
 starting [27](#)
Resources
 assigning ID numbers to [12](#), [28](#)
 compiling the BAR file [29](#)
 creating for BREW applications [12](#), [26](#)
 defining images for [28](#)
 learning BREW [46](#)
 translating [12](#)
Resume events, handling [39](#)

S

Sample applications [5](#), [44](#)
SDK
 components [8](#)
 documentation [16](#), [46](#)
 downloading [15](#)
SETAEERECT function [38](#)
Source code, writing [30](#)
Strings
 creating for applications [12](#)
 defining [27](#)
Suspend events, handling [39](#)

T

Testing
 applications on the Emulator [13](#)
 MyFirstApp on the BREW Emulator [41](#)
Training, BREW [46](#)
Translating resources [12](#)
Tutorial, tips for using [5](#)

V

Visual Studio
 creating a BREW application project [10](#)
 obtaining [15](#)
 starting [16](#)

W

Web site, BREW [7](#)
Workspace file [23](#)
Writing
 BREW applications [10](#)
 source code for MyFirstApp [30](#)