QUALCOMM

Binary Runtime Environment for Wireless®

Starting with BREW™

brew™

QUALCOMM Proprietary

Starting with BREW
80-D4725-1 Rev. A
April 27, 2004

Contents

Figures

Introduction

QUALCOMM's Binary Runtime for Wireless Environment® (BREW™) provides a powerful framework for creating exciting applications on a wide variety of mobile devices. By using this guide, you will become familiar with the BREW development environment. The guide walks you through the steps necessary to install the BREW SDK™, develop BREW applications, and load those applications into a handset. Solutions are presented for the most common problems encountered during each step of the process; additional resources for solving other issues pertinent to each phase of development are also listed.

About the BREW SDK

As of this article's publication, five versions of the BREW SDK are currently available for download: 1.0, 1.1, 2.0, 2.1, and 3.0. Depending on your targeted platform and the capabilities required by your application, you may need to download and install multiple versions of the SDK. Patches for many of the SDK installations are available for download on the same page as the SDK, and you should install them as appropriate. Note that devices may not support all interfaces included with the SDK. By inspecting the appropriate Device Data Sheets (DDS), ensure that the desired device supports all APIs necessary for the application under development. To ensure compatibility with the mobile device, you should perform all development in the same or lower SDK release as the version of BREW used by the device.. BREW devices are backwards compatible; that is, any applications written in a previous version of the SDK function on later versions of BREW.

This section provides a brief overview of the features of each version of the SDK, including any major changes made to the API collection. For more information on the specific API additions, consult the Release Notes for the desired SDK, which are available on the installation page. Many APIs are enhanced with additional functionality with each new release; the BREW SDK APIs by Version guide gives a complete listing of the methods available under a given version of an interface, including the general SDK helper functions provided. Detailed method usage information for each API is available through the BREW API Reference, included in the SDK download package.

After you identify the SDK versions necessary to develop the desired platforms, you install them from the SDK download page along with the corresponding patches. Obtain a CD-ROM containing the SDK installation packages free of charge by contacting brew-support@qualcomm.com be sure to include full shipping information in your request.

BREW SDK 1.0

No BREW 1.0 devices are currently available commercially. Previously available handsets implementing BREW 1.0 include the Motorola V731 and Sharp Z800.

API additions in BREW SDK 1.0

IApplet	Provides event-handling routine. All BREW applications must implement this interface
IAStream	Provides an abstract interface to read data from an asynchronous stream
IBase	Provides the base interface for all other BREW AEE interfaces,

	provides object reference counting mechanism
IControl	Provides an abstract interface implemented by all BREW control interfaces
IDatabase	Allows access and modification of database records from database files opened or created by IDBMgr
IDateCtl	Provides a control interface for choosing and displaying dates in several formats
IDBMgr	Opens or creates database files for modification through IDatabase
IDBRecord	Accesses and modifies the fields of database records
IDialog	Manipulates dialogs created through IShell
IDisplay	Modifies the device display with text, bitmaps, and simple geometric shapes
IFile	Allows access and modification of files opened or created by IFileMgr
IFileMgr	Opens or creates files for modification though IFile
IGraphics	Adds the ability to draw more complex lines and shapes than those supported through IDisplay
IHeap	Manages device memory and provides allocation/deallocation routines and memory usage information
IImage	Implements support for drawing various image formats to the screen
IMemAStream	Creates an asynchronous stream resource from a section of memory
IMenuCtl	Provides access to a variety of customizable generic menu controls
IModule	Supplies a mechanism for controlling access to a group of associated applets or components
INetMgr	Configures parameters associated with the network subsystem of a mobile device
INotifier	Used by applets to register for notification of events occurring in other classes
IShell	Permits access to a wide variety of lower-level services provided by the phone
ISocket	Grants control to manage and use TCP and UDP sockets opened through INetMgr
ISound	Allows access to basic sound services, such as beeps and vibrations
ISoundPlayer	Supports playback of advanced audio formats
IStatic	Used to create and display static text controls with unmodifiable text
ITAPI	Interfaces with the device telephony layer for access to voice and SMS services
ITextCtl	Provides text entry and editing capabilities
ITimeCtl	Control for several formats of time value input

IViewer	Provides functionality identical to the IImage interface

BREW SDK 1.1

BREW 1.1 has a widely established user base, with the continued commercial availability devices of BREW 1.1 devices. Some of the more prominent examples in this category are the Motorola T-720 and LGE VX4400.

API additions

IAddrBook	Provides an interface for interacting with the OEM address book capabilities of the handset
IAddrRec	Facilitates the access and modification of fields within the address records retrieved through IAddrBook
ICipher	Enables applications to easily encrypt and decrypt information
IGetLine	Supplies methods allowing line parsing of data from sources
IHash	Provides access to data hashing routines
IHtmlViewer	Supports rendering a subset of HTML 3.2
ILicense	Allows applications to query for their respective license information, and modify usage-based license counts
IPeek	Implements a Peek/Advance functionality for accumulating data in a buffer
IPosDet	Grants access to position-determination services using sector or GPS information
IQueryInterface	Base level object class for deriving extensible API object classes
IRingerMgr	Manipulates the handset ringer folder, allowing creation and deletion of ringer files
IRSA	Supports encryption of data using the RSA algorithm
ISource	An abstract data source providing a Read/Readable interface
ISourceUtil	Used to construct an ISource from commonly-used sources
IWeb	Facilitates conducting web transactions using a variety of protocols
IWebOpts	Allows for configuration options affecting IWeb transactions to be specified

SDK utility additions

Purevoice Converter	This application allows conversion of audio files from WAV to Qualcomm's Purevoice (QCP) format. For usage specifics, consult the BREW Utilities Guide, included as part of the *BREW SDK User* document in BREW SDK 3.0 and greater, and as a standalone document in previous SDK versions.

BREW SDK 2.0

Example devices in this category are the LGE VX6000, Kyocera SE47 (Slider), and Samsung SCH-A670.

API additions

IBitmap	Manipulates in-memory bitmaps, facilitating simple draw operations and bit blitting
IBTAG	Used to open, close, and manage Bluetooth Audio Gateway with a device (removed in SDK 3.0)
IBTSDP	Allows discovering Bluetooth devices and querying for device information (removed in SDK 3.0)
IBTSIOPORT	Provides Bluetooth serial interface connection management and data transfer services (removed in SDK 3.0)
IClipboard	Provides access to standard clipboard cut and paste features
IDIB	Implements a device-independent bitmap structure, inheriting from IBitmap (removed in SDK 3.0)
IDNS	Permits DNS queries
IFont	Adds functions for drawing and measuring text
IHashCTX	Enables greater functionality in hashing data
IImageCtl	Permits displaying an image within a scrollable view frame
IMedia	Abstract base class for all BREW multimedia objects
IRamCache	Provides an LRU caching with TTL mechanism for heap storage
ISprite	Contains methods for rendering sprites and tile maps
ISSL	Supports SSL/TLS security for network connection
ITransform	Provides functions for performing bit blitting operations with transforms using a bitmap
IUnzipAStream	Implements support for decompressing and reading compressed IAStreams
IVocoder	Interfaces with the handset vocoder to capture and play vocoder frames
IX509Chain	Used for managing and verifying a chain of X.509 certificates

SDK utility additions

2Bit Tool	Facilitates conversion between 4-bit and 2-bit BMP formats for editing purposes. For usage specifics, consult the BREW Utilities Guide in SDK User documents in BREW SDK 3.0 and greater, and as a standalone document in previous SDK versions.
NMEA Logger	Allows retrieving recorded GPS activity from a GPS or GNSS device for use in the Emulator. For usage specifics, consult the BREW

Utilities Guide, included as part of the BREW SDK User documents in BREW SDK 3.0 and greater, and as a standalone document in previous SDK versions.

BREW SDK 2.1

The example device in this category is the Toshiba CDM-9900.

API additions in BREW SDK 2.1

I3D	Provides the definitions of a 3D graphics engine for rendering triangles (removed in SDK 3.0)
I3DModel	Supports drawing structured groups of triangles (3D models) (removed in SDK 3.0)
I3DUtil	Allows access to the transformation matrices and unit vectors used by the 3D engine (removed in SDK 3.0)
ICallHistory	Grants access and modification to the native call history records
ICamera	Enables a standardized interface for accessing the camera on a mobile device and record media in various formats
ILogger	Provides a standardized data logging interface for logging using a range of transport mechanisms
IMediaUtil	Assists in creating IMedia objects from various input sources and encoding of new media files
IRecordStore	Abstract class for accessing simple key-value stores of records

BREW SDK 3.0

No BREW 3.0 commercial devices are currently available, though they are slated for release in the second half of 2004.

API additions in BREW SDK 3.0

IAClockCtl	Allows for the creation of analog clock displays
IBitmapDev	Used by various functions dealing with device bitmaps
IModuleSupport	Provides module support services to dynamically loaded modules
IPort	Generic interface for bidirectional data stream implementing common interfaces
IRscPool	Enables memory and interface grouping for many resources with identical lifetimes
IThread	Provides the capability to implement cooperative multithreading
IWebEng	Abstract base class allowing BREW modules to extend the functionality of IWeb

API removals in BREW SDK 3.0

I3D

I3DModel

I3DUtil

IBTAG

IBTSDP

IBTSIOPORT

IDIB

Installing the BREW SDK

From the BREW SDK download page, select the desired version of the BREW SDK and click **Install**. A web-based installer guides you through the installation process.

Installation Directory

Generally, it is best to install the BREW SDK to the default directory specified by the installer. Changing the installation directory does not have an adverse impact in most cases, though in BREW 1.x installing to a directory containing a '.' (period) may cause issues when executing the Emulator.

NOTE: The Emulator was changed to Simulator in the SDK 3.0 and will henceforth be referred to as the Simulator.

BREWDIR

During the installation process, you are prompted to set/update the BREWDIR environment variable, or leave it unchanged. Microsoft Visual Studio, as well as makefiles generated by the BREW Add-Ins for Microsoft Visual Studio, uses this environment variable. For practical purposes, it should be set to the directory of the SDK intended for the majority of development work. If multiple versions of the BREW SDK are installed on an individual computer, the BREWDIR variable must be changed prior to beginning development in each version of the SDK. If the environment variable is not changed, it is necessary to modify makefiles generated by the secondary BREW SDK installations; the developer manually adds the proper versions of the BREW API libraries to new BREW applications created using the application wizard.

SDK core components

BREW Compressed Image Authoring Tool (BREW 1.1+)	Allows customization of device skin files used by the Simulator to simulate the appearance and capabilities of a mobile device. This tool is obsolete in BREW 3.0+, as the BREW Simulator incorporates all the functionalities. Documentation for this application is available in the *BREW Device Configurator Guide* (BREW 1.1, 2.x), or the "BREW Device Configurator" section of the *BREW Online Help* (BREW 2.0+).
BREW Device Configurator (BREW 1.x, 2.x)	Allows customization of device skin files used by the Simulator to simulate the appearance and capabilities of a mobile device.This tool is obsolete in BREW 3.0+, as the BREW Simulator incorporates all the functionalities. Documentation for this application is available in the *BREW Device*

	Configurator Guide (BREW 1.1, 2.x), or the “BREW Device Configurator” section of the *BREW Online Help* (BREW 2.0+).
BREW Emulator (BREW 1.x, 2.x)	The Emulator provides a device-like environment for running BREW applications on a PC, which assists in the debugging process. In BREW SDK 3.0+, the BREW Simulator has replaced the BREW Emulator. Detailed Emulator documentation is available in the “BREW Emulator” sections of the *BREW SDK User’s Guide* (BREW 1.1, 2.x) and *BREW Online Help* (BREW 2.x).
BREW MIF Editor	Used for creating and modifying an application’s Module Information File (MIF), which contains important information about the classes and applications supported by a module. Documentation for this application is available in the *BREW MIF Editor Guide* (BREW 1.1, 2.x), or the “BREW MIF Editor” section of the *BREW Online Help* (BREW 2.0+).
BREW Resource Editor	To enhance inter-platform portability, developers should place resources within BREW Applet Resource (BAR) files, which can be compiled using the Resource Editor. Documentation for this application is available in the *BREW Resource Editor Guide* (BREW 1.1, 2.x), or the “BREW Resource Editor” section of the *BREW Online Help* (BREW 2.0+).
BREW Simulator (BREW 3.0+)	Encapsulates the functionality of the BREW Emulator and Device Configurator, allowing the execution of BREW applications on a modifiable simulated device environment. For complete documentation on Simulator usage, consult the “BREW Simulator” section of the *BREW SDK User* documents (BREW 3.0+).

Documentation

BREW API Reference (BREW 1.x, 2.x)	Contains detailed information on the BREW APIs and helper methods available within the SDK, which will be an especially valuable resource during BREW development. This information is also available as an online help (CHM) file in BREW 2.0+.
BREW BCI Guide (BREW 1.1, 2.x)	Provides additional information on the BREW Compressed Image format, as well as usage of the BREW Compressed Image Authoring Tool. The information contained in this guide is also available in the “BREW Compressed Image Tool” section of the *BREW Online Help* (BREW 2.0+).
BREW Device Configurator Guide (BREW 1.x, 2.x)	Provides instruction on using the BREW Device Configurator to customize the appearance and functionality of Emulator device skins. The information contained in this guide is also available in the “BREW Device Configurator” section of the *BREW Online Help* (BREW 2.x).
BREW MIF Editor Guide (BREW 1.x, 2.x)	Assists in using the BREW MIF Editor for creating and modifying application MIF files. The information contained in this guide is also available in the “BREW MIF Editor” section of the *BREW Online Help* (BREW 2.0+).
BREW Resource Editor Guide (BREW 1.x, 2.x)	Contains instructions for using the BREW Resource Editor to create and compile BREW Applet Resources. The information contained in this guide is also available in the “BREW Resource Editor” section of the *BREW Online Help* (BREW 2.0+).
BREW Sample Application Guide	Summarizes the sample applications included with the SDK. Each version of the BREW SDK includes a collection of source code for several applications

(BREW 2.0+)	demonstrating the API functions. This provides a valuable resource for developers seeking examples of functional BREW programs.
BREW SDK User's Guide (BREW 1.x, 2.x)	This document provides an overview of fundamental aspects of BREW, including programming techniques, troubleshooting assistance for a variety of common problems, and an overview of the applications included in the SDK.
BREW Utilities Guide (BREW 1.1, 2.x)	Covers topics relevant to the BREW Utilities included in the particular SDK version. In BREW 3.0+, this information was incorporated into the BREW SDK User Docs.

Online help

BREW API Reference (BREW 2.0+)	Contains detailed information on the BREW APIs and helper methods available within the SDK, which will be an especially valuable resource during BREW development.
BREW Online Help (BREW 2.x)	This document provides an overview of fundamental aspects of BREW, including programming techniques, troubleshooting assistance for a variety of common problems, and an overview of the applications included in the SDK. In BREW 3.0+, this guide has been split into separate BREW Programming Concepts and BREW SDK User Doc components.
BREW Programming Concepts (BREW 3.0+)	Offers valuable information on programming within the BREW environment, including event-handling techniques, module design, and BREW extension guidelines. This information is also contained within the BREW Online Help guide in previous versions of BREW.
BREW SDK User Doc (BREW 3.0+)	Documents the applications comprising the SDK and provides troubleshooting advice for SDK problems. This information is also contained within the BREW Online Help guide in previous versions of BREW.

Utilities

2Bit Tool (BREW 2.0+)	Facilitates conversion between 4-bit and 2-bit BMP formats for editing purposes. For usage specifics on this and other BREW Utilties, consult the *BREW Utilities Guide*, included as part of the *BREW SDK User Docs* document in BREW SDK 3.0 and greater, and as a standalone document in previous SDK versions.
NMEA Logger Tool (BREW 2.0+)	Allows retrieving recorded GPS activity from a GPS or GNSS device for simulating GPS behavior within the Simulator.
Purevoice Converter (BREW 1.1+)	This application allows conversion of audio files from WAV to Qualcomm's Purevoice (QCP) format, which offers compression benefits at very little compromise of audio quality.

Common issues

System requirements

If your computer fails to meet the minimum system requirements, the BREW SDK installation fails.

- Internet Explorer version 5.5 SP2 or higher with 128-bit encryption
- Windows NT 4.0, Windows 2000, or Windows XP
- Administrator privileges for your computer

The BREW SDK is currently available for installation on Windows platforms only; due to the requirement for Unicode text encoding capabilities, Windows 9x and ME are not supported.

Additional resources

Please address any problems with accessing the SDK download page to brew-support@qualcomm.com. Be sure to include a detailed description of the problem encountered as well as any pertinent error messages.

Writing BREW Applications

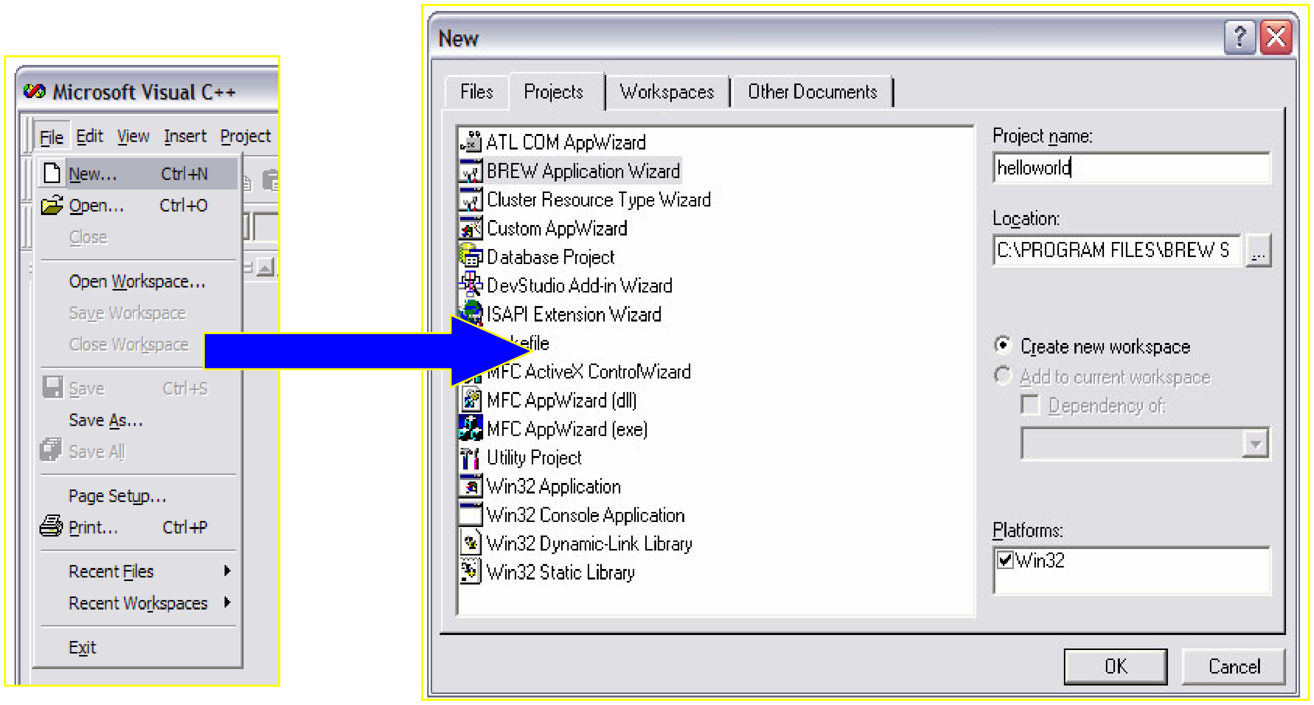
This guide assumes that development is performed within the Microsoft Visual Studio 6.0 environment, which is the officially supported development studio. Authoring BREW applications using later versions of Microsoft Visual Studio (Visual Studio .NET) is also possible. Consult the Creating BREW™ Applications Using Visual Studio .NET guide for specifics on importing BREW projects into the Visual Studio .NET environment.

This section presents an overview of the BREW development process, including the basic steps necessary to write a functional BREW application. For a more in-depth tutorial on writing your first BREW application refer to the Creating a BREW™ Application from Scratch document, which details the creation of an application with key handling and resource files.

Creating a new application with the BREW Application Wizard

The BREW Application Wizard facilitates the creation of new BREW projects by providing a generic code framework for your application. The wizard is automatically installed with BREW SDK 1.1 and above. To use the BREW Application Wizard, launch Microsoft Visual Studio and click **File>New** to create a new project (File → New). The BREW Application Wizard appears within the available project type menu (see Figure 1 Invoking the BREW Application Wizard). Enter the desired project name and location and click **OK** to start the wizard.

NOTE: To function properly on all versions of BREW, filenames should not contain uppercase letters. Filenames containing uppercase characters are not allowed on the mobile device; additionally, they cause errors during emulation in BREW SDK 3.0+. Your application's name must begin with alphabetical characters; applications downloaded over the air have MIF and directory names containing only numbers, and BREW treats these files differently. For example, an application directory named *helloworld* is renamed as something similar to *1234* when downloaded over the air. Failing to follow this convention may lead to fatal errors in your application. Additionally, note that it is not safe to assume that you will know the name of your MIF file or application directory when the application has been downloaded to a device over the air.



Project name: helloworld

Location: C:\PROGRAM FILES\BREW S

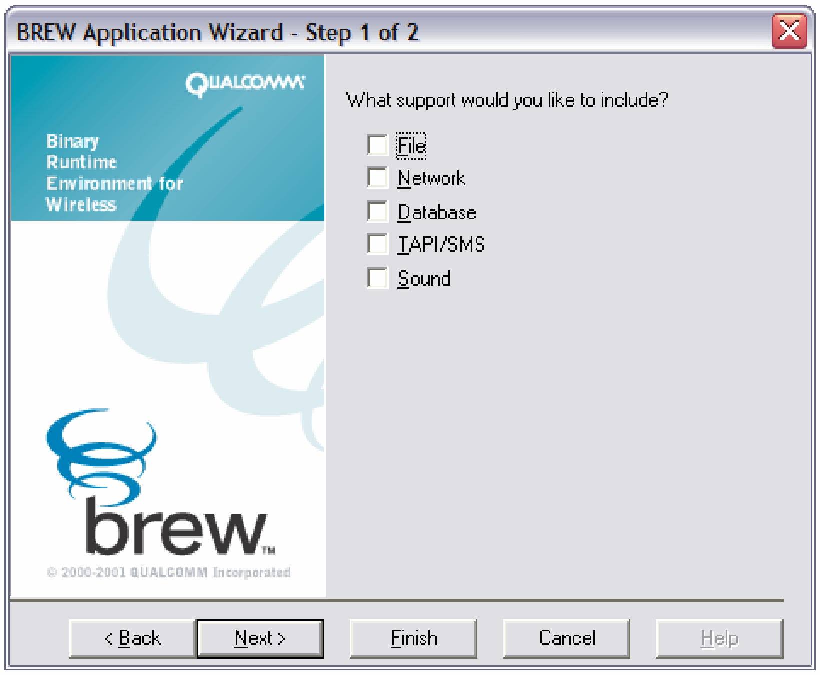
(●) Create new workspace

Platforms: ☑ Win32

Figure 1 Invoking the BREW Application Wizard

The first step in the Application Wizard is to specify the interfaces your application uses (see Figure 2. BREW Application Wizard step 1). Checking boxes within this window inserts a `#include` preprocessor directive within your source code, providing access to the API methods supporting the specified functionality. You need to provide additional `#include` directives for any other header files required by the APIs used in the application (see the *BREW API Reference* for this information).

NOTE: Selecting support for interfaces within the Application Wizard is not equivalent to specifying privilege levels within the application MIF. If your application requires privileges, you must specify the appropriate access rights using the MIF Editor. See Setting privileges for details. Find information on the privilege levels required for various APIs within the corresponding section of the *BREW API Reference*.

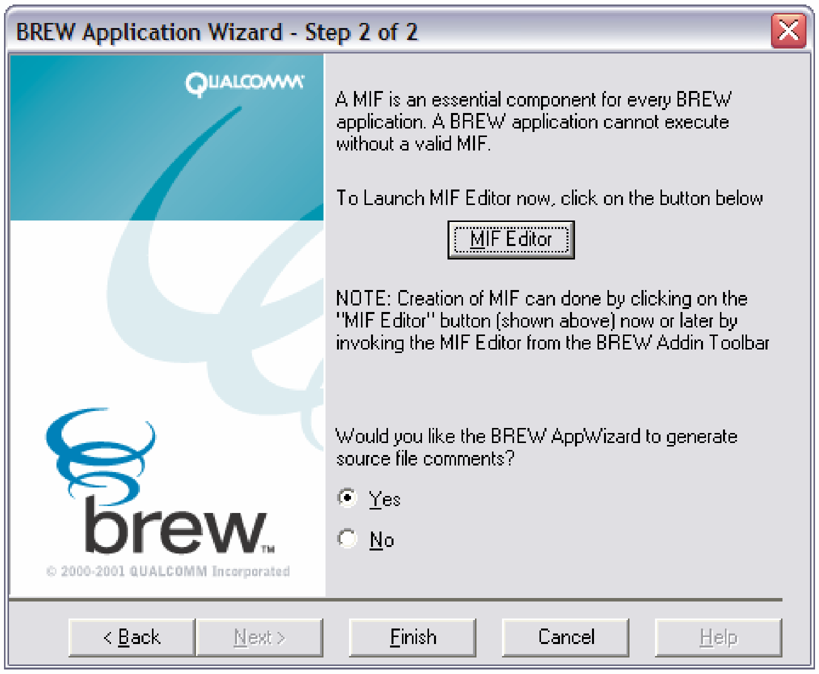

BREW Application Wizard - Step 1 of 2
QUALCOMM
Binary Runtime Environment for Wireless
brew
© 2000-2001 QUALCOMM Incorporated
< Back
Next >
Finish
Cancel
Help

What support would you like to include?

- [ ] File
- [ ] Network
- [ ] Database
- [ ] TAPI/SMS
- [ ] Sound

Figure 2. BREW Application Wizard step 1

After adding support for the desired interfaces, click **Next** to advance the wizard to the second and final step (see Figure 3. BREW Application Wizard step 2).


BREW Application Wizard - Step 2 of 2
QUALCOMM
brew
© 2000-2001 QUALCOMM Incorporated
MIF Editor
< Back
Next >
Finish
Cancel
Help

A MIF is an essential component for every BREW application. A BREW application cannot execute without a valid MIF.

To Launch MIF Editor now, click on the button below

NOTE: Creation of MIF can done by clicking on the "MIF Editor" button (shown above) now or later by invoking the MIF Editor from the BREW Addin Toolbar

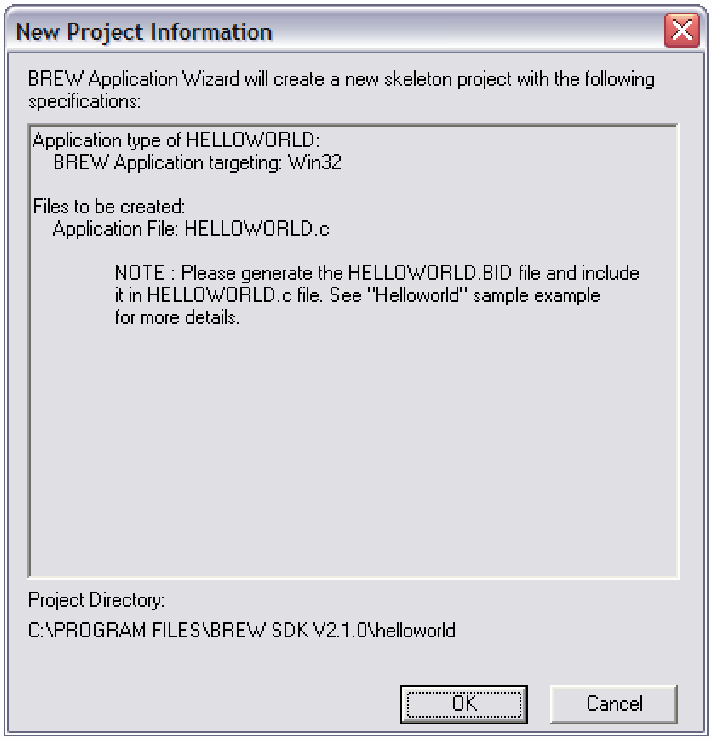
Would you like the BREW AppWizard to generate source file comments?

[✓ Yes] No

Figure 3. BREW Application Wizard step 2

Alternatively, you can click **Finish** to create your project immediately, without disabling pre-generated source code comments. In this step, the wizard prompts you to launch the BREW

MIF Editor to create a MIF for your module. The MIF contains important information about the classes and applications supported by a module, such as the applet name and icon. While you may start application development before you create a MIF, you must do so before testing the application within the Simulator or loading to a mobile device; see Using the BREW MIF Editor for details. You may also opt to disable the automatic generation of source code comments, though you may find the comments helpful in developing your first BREW applications. After you click **Finish**, the wizard completes its processes, and a window appears, summarizing the actions taken (see Figure 4. New Project Information window).



BREW Application Wizard will create a new skeleton project with the following specifications:

```
Application type of HELLOWORLD:
   BREW Application targeting: Win32

Files to be created:
   Application File: HELLOWORLD.c

            NOTE : Please generate the HELLOWORLD.BID file and include
            it in HELLOWORLD.c file. See "Helloworld" sample example
            for more details.
```

Project Directory:

C:\PROGRAM FILES\BREW SDK V2.1.0\helloworld

Figure 4. New Project Information window

To view the newly created code, close the New Project Information window, and open the helloworld.c file (the name of this file will depend on the name of your project) from the project workspace FileView (). The wizard-generated code is presented below, with a description of the important sections.

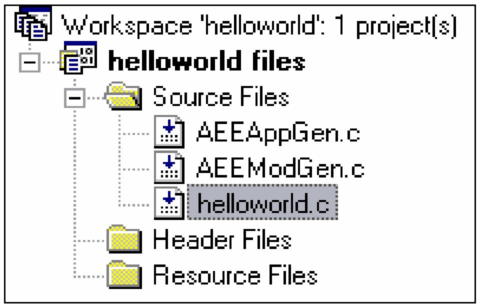


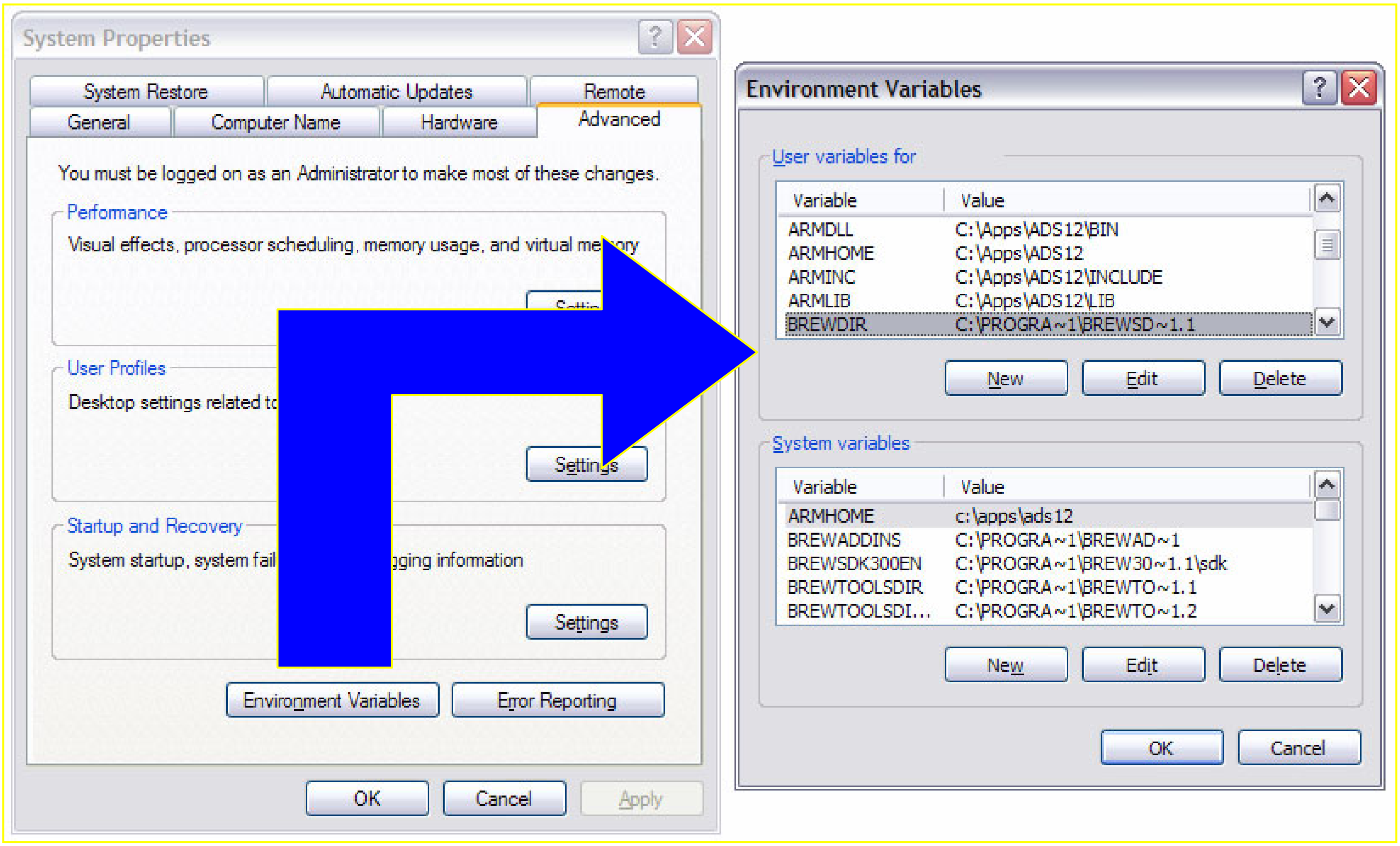
Figure 5. New project workspace

NOTE: If you are developing your application for a version of the BREW SDK located in a directory other than the location specified by the BREWDIR environment variable, the AEEAppGen.c, AEEModGen.c, and libraries linked with your application will be incorrect. To remedy this error, you must manually replace the files in your project with the correct version (a tedious and error-prone process), or change the value of the BREWDIR variable. Changing the value of the BREWDIR is the preferred solution, as the project may later be safely compiled under other versions of the SDK by simply updating the environment variable.

Changing the BREWDIR environment variable

The BREWDIR variable is modified through the Environment Variables section of your computer's System Properties. Right-click on **My Computer > Properties > Advanced > Environment Variables** to access the window. Scroll down through the list of user variables until you find the entry for BREWDIR. This should be set to the root directory of the target SDK installation (see Figure 6. Modifying environment variables).

NOTE: If you change the BREWDIR environment variable while an application using the variable is running (for example, an MS-DOS command prompt or Visual Studio), you must close and reopen that application for the changes to take effect.



You must be logged on as an Administrator to make most of these changes.

Visual effects, processor scheduling, memory usage, and virtual memory

Desktop settings related to

System startup, system fail ... gging information

User variables for

Variable	Value
ARMDLL	C:\Apps\ADS12\BIN
ARMHOME	C:\Apps\ADS12
ARMINC	C:\Apps\ADS12\INCLUDE
ARMLIB	C:\Apps\ADS12\LIB
BREWDIR	C:\PROGRA~1\BREWSD~1.1

System variables

Variable	Value
ARMHOME	c:\apps\ads12
BREWADDINS	C:\PROGRA~1\BREWAD~1
BREWSDK300EN	C:\PROGRA~1\BREW30~1.1\sdk
BREWTOOLSDIR	C:\PROGRA~1\BREWTO~1.1
BREWTOOLSDI...	C:\PROGRA~1\BREWTO~1.2

Figure 6. Modifying environment variables

Skeletal code generated by the BREW Application Wizard

The wizard generated code is included below. Important sections of the code (numbered) are discussed afterwards

```
/*===========================================================================

FILE: helloworld.c
===========================================================================*/


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


#include "AEEModGen.h"          // Module interface definitions     ❶
#include "AEEAppGen.h"          // Applet interface definitions
#include "AEEShell.h"           // Shell interface definitions


#include "helloworld.bid"     ❷
```

```
/*-----------------------------------------------------------------
Applet structure. All variables in here are reference via "pMe->"
-----------------------------------------------------------------*/

typedef struct _helloworld {    ❸

  AEEApplet      a ;          // First element of this structure must be
AEEApplet
  AEEDeviceInfo  DeviceInfo; // always have access to the hardware device
information
  IDisplay      *pIDisplay;  // give a standard way to access the Display
interface
  IShell        *pIShell;    // give a standard way to access the Shell
interface

    // add your own variables here...


} helloworld;

/*-----------------------------------------------------------------
Function Prototypes
-----------------------------------------------------------------*/
static  boolean helloworld_HandleEvent(helloworld* pMe, AEEEvent eCode,
                                             uint16 wParam, uint32
dwParam);

boolean helloworld_InitAppData(helloworld* pMe);
void    helloworld_FreeAppData(helloworld* pMe);

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

/*===========================================================================
FUNCTION: AEEClsCreateInstance
===========================================================================*/

❹

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

  if( ClsId == AEECLSID_HELLOWORLD )
  {
    // Create the applet and make room for the applet structure
    if( AEEApplet_New(sizeof(helloworld),
          ClsId,
          pIShell,
          po,
            (IApplet**)ppObj,
            (AEEHANDLER)helloworld_HandleEvent,
            (PFNFREEAPPDATA)helloworld_FreeAppData) )
      // the FreeAppData function is called after sending EVT_APP_STOP to
the HandleEvent
      // function

    {
      //Initialize applet data, this is called before sending EVT_APP_START
      // to the HandleEvent function
      if(helloworld_InitAppData((helloworld*)*ppObj))
      {
        //Data initialized successfully
        return(AEE_SUCCESS);
      }
      else
      {
        //Release the applet. This will free the memory allocated for the
applet when
        // AEEApplet_New was called.
        IAPPLET_Release((IApplet*)*ppObj);
        return EFAILED;
      }

    } // end AEEApplet_New



  return(EFAILED);
}


/*===========================================================================
FUNCTION SampleAppWizard_HandleEvent
===========================================================================*/
```

❺

```
static boolean helloworld_HandleEvent(helloworld* pMe, AEEEvent eCode,
uint16 wParam, uint32 dwParam)
{

  switch (eCode)
  {
    // App is told it is starting up
    case EVT_APP_START:
      // Add your code here...

      return(TRUE);


    // App is told it is exiting
    case EVT_APP_STOP:
      // Add your code here...

      return(TRUE);


    // App is being suspended
    case EVT_APP_SUSPEND:
      // Add your code here...

      return(TRUE);


    // App is being resumed
    case EVT_APP_RESUME:
      // Add your code here...

      return(TRUE);


    // An SMS message has arrived for this app. Message is in the dwParam
above as
    // (char *)
    // sender simply uses this format "//BREW:ClassId:Message", example
    //BREW:0x00000001:Hello World
    case EVT_APP_MESSAGE:
      // Add your code here...

      return(TRUE);

    // A key was pressed. Look at the wParam above to see which key was
pressed. The key
    // codes are in AEEVCodes.h. Example "AVK_1" means that the "1" key was
pressed.
    case EVT_KEY:
      // Add your code here...

      return(TRUE);


    // If nothing fits up to this point then we'll just break out
    default:
        break;
   }

   return FALSE;
}
```

❻

```
// this function is called when your application is starting up
boolean helloworld_InitAppData(helloworld* pMe)
{
  // Get the device information for this handset.
  // Reference all the data by looking at the pMe->DeviceInfo structure
  // Check the API reference guide for all the handy device info you can
get
  pMe->DeviceInfo.wStructSize = sizeof(pMe->DeviceInfo);
  ISHELL_GetDeviceInfo(pMe->a.m_pIShell,&pMe->DeviceInfo);

  // The display and shell interfaces are always created by
  // default, so we'll asign them so that you can access
  // them via the standard "pMe->" without the "a."
  pMe->pIDisplay = pMe->a.m_pIDisplay;
  pMe->pIShell   = pMe->a.m_pIShell;

  // Insert your code here for initializing or allocating resources...



  // if there have been no failures up to this point then return success
  return TRUE;
}
```

❼

```
// this function is called when your application is exiting
void helloworld_FreeAppData(helloworld* pMe)
{
  // insert your code here for freeing any resources you have allocated...

  // example to use for releasing each interface:
  // if ( pMe->pIMenuCtl != NULL )         // check for NULL first
  // {
  //    IMENUCTL_Release(pMe->pIMenuCtl)   // release the interface
  //    pMe->pIMenuCtl = NULL;         // set to NULL so no problems trying
to free later
  // }
  //

}
```

API library includes ❶

To use the methods in a particular BREW API, the corresponding header file must be included in your application. Information on the necessary header files for each API is contained within the BREW API Reference.

NOTE: To prevent the introduction of static and global data, do not link standard C libraries to your BREW application. As such, standard C library routines (for instance, strcat(), malloc(), sprintf()) cannot be used within BREW. Helper functions are provided for the most commonly used C functions; see the *BREW API Reference* for more details. Additionally, floating-point operations may only be performed through the floating-point helper functions. These limitations are the result of the ARM platform, not BREW itself.

Class ID file ❷

Before compiling your application,, you must generate a class ID file. This file contains a unique 32-bit identification code (8-digit hexadecimal value), which is used by the BREW interface creation mechanism. For testing on a local system, it is sufficient to generate a local class ID through the MIF Editor, though QUALCOMM must issue a unique ID to applications intended for distribution. For more information on obtaining class ID files, see the Using the BREW MIF Editor.

Applet structure ❸

Any objects used during the lifetime of your application should be placed within the applet structure. As static and global variables are not permitted within BREW, the applet structure provides a mechanism for achieving similar behavior. The applet structure is received as a parameter in most BREW methods, which allows access to the structure members. Typically, the elements of the applet are all initialized within the application’s data initialization method, and deallocated in the application’s data free method. Consolidating all allocations and deallocations decreases the chance of memory access errors and memory leaks.

NOTE: The first element of your applet structure must be an AEEApplet object so the AEEApplet_New() method can properly initialize your applet.

AEEClsCreateInstance() method ❹

This is a required method for all applications. BREW automatically invokes this routine to create an instance of your applet in memory prior to sending the EVT_APP_START. Within this method AEEAppletNew() is called, registering the event handler, data initialization (optional), and data deallocation functions. All initialization activities specific to your application are performed in your data initialization function; it should not be necessary to modify the AEEClsCreateInstance() method.

Event handler ❺

All BREW applications must contain an event handling function, which is registered in the call to AEEAppletNew() within the AEEClsCreateInstance() method. Events are passed to this function by the BREW layer, where the application performs the appropriate actions. In the BREW event-handling environment, events must be handled within a limited timeframe, and all processing must be completed before the next event may be received. After receiving an event, the handler code should return TRUE to indicate successful handling, or FALSE if the event is not handled. Event codes supported by BREW (AEE Events) are listed within the Data Types section of the *BREW API Reference*. Consult the *BREW Programming Concepts Guide* for more information on event-handling rules and principles in BREW.

Applet data init method ❻

This method is used to perform any initialization code required by the applet. Typically, this function is used to instantiate the members of your applet structure or allocate memory for buffers, returning a Boolean value to indicate whether the process was successful. By placing all your allocation code in a single method, you reduce the potential for neglecting to properly instantiate a member of the applet structure.

Applet data free method ❼

The applet data free method is used for deallocation of all resources and any additional procedures required for the application to safely terminate. In general, this is accomplished by calling the corresponding XXX_Release() method for objects that were instantiated through ISHELL_CreateInstance(), using FREE() on memory allocated through MALLOC(), and canceling all pending timers. All resource allocations performed in your applet’s data initialization method should be deallocated within this method.

Common issues

If the BREW Applications Wizard is not available within Visual Studio's New Project window, you may need to install a newer version of the BREW Add-Ins for Microsoft Visual Studio.

Additional resources

As previously mentioned, the *Creating a BREW™ Application from Scratch* document provides an excellent tutorial for writing a BREW application with simple event handling and a resource file. The example programs included with the SDK installation (documented in the *BREW Sample Applications Guide*) are also a valuable resource in learning BREW; these enable you to analyze functional BREW code. For any specific usage details on the various BREW APIs, consult the *BREW API Reference*. *The BREW Programming Concepts Guide* contains a general overview of the BREW environment and BREW development principles.

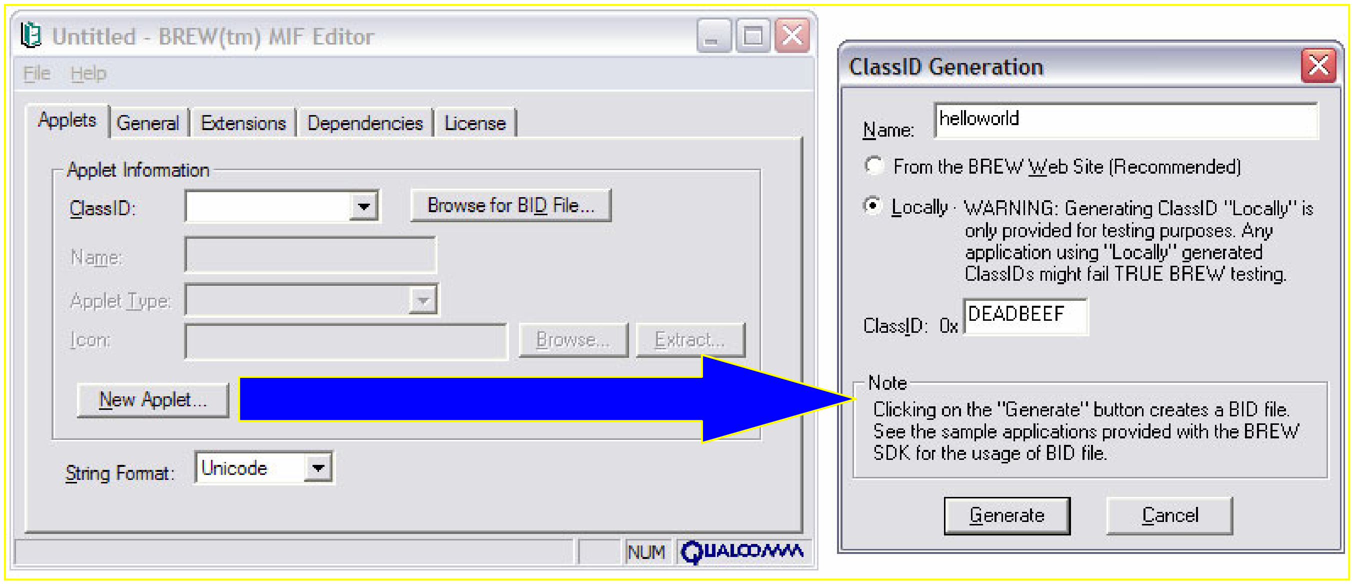
Using the BREW MIF Editor

You must create a MIF before you place your application on a BREW device or within the BREW Simulator. The MIF contains important information about your application, such as the name, icons, class ID, and privilege levels. The BREW MIF Editor, described in this section, provides an easy tool for creating this file.

Generating a class ID

To ensure unique class ID numbers across all developers, applications intended for distribution must generate a class ID through the BREW Generators page, which requires authentication. For testing purposes, generating a local class ID file using the MIF Editor is sufficient.

To generate a local class ID file (see Figure 7. Local Class ID generation), click **New Applet**, then enter the applet name in the subsequent window and click **Locally**. Provide a class ID value (in hexadecimal form) that you know is not already in use, and click **Generate**. Save the file within your project directory so it will be available for #include directives.



String Format: Unicode

Name: helloworld

( ) From the BREW Web Site (Recommended)

(●) Locally - WARNING: Generating ClassID "Locally" is only provided for testing purposes. Any application using "Locally" generated ClassIDs might fail TRUE BREW testing.

ClassID: 0x DEADBEEF

Note

Clicking on the "Generate" button creates a BID file. See the sample applications provided with the BREW SDK for the usage of BID file.

Figure 7. Local Class ID generation

If you open the resulting BID file in a text editor (see Figure 8. Sample Class ID file), note that the class ID is specified as a #define macro using the convention AEECLSID_<PROGRAM NAME>, which allows you to refer to this value within your application by using a #include directive. In the future, you create new local class ID files by editing this constant's name and value.

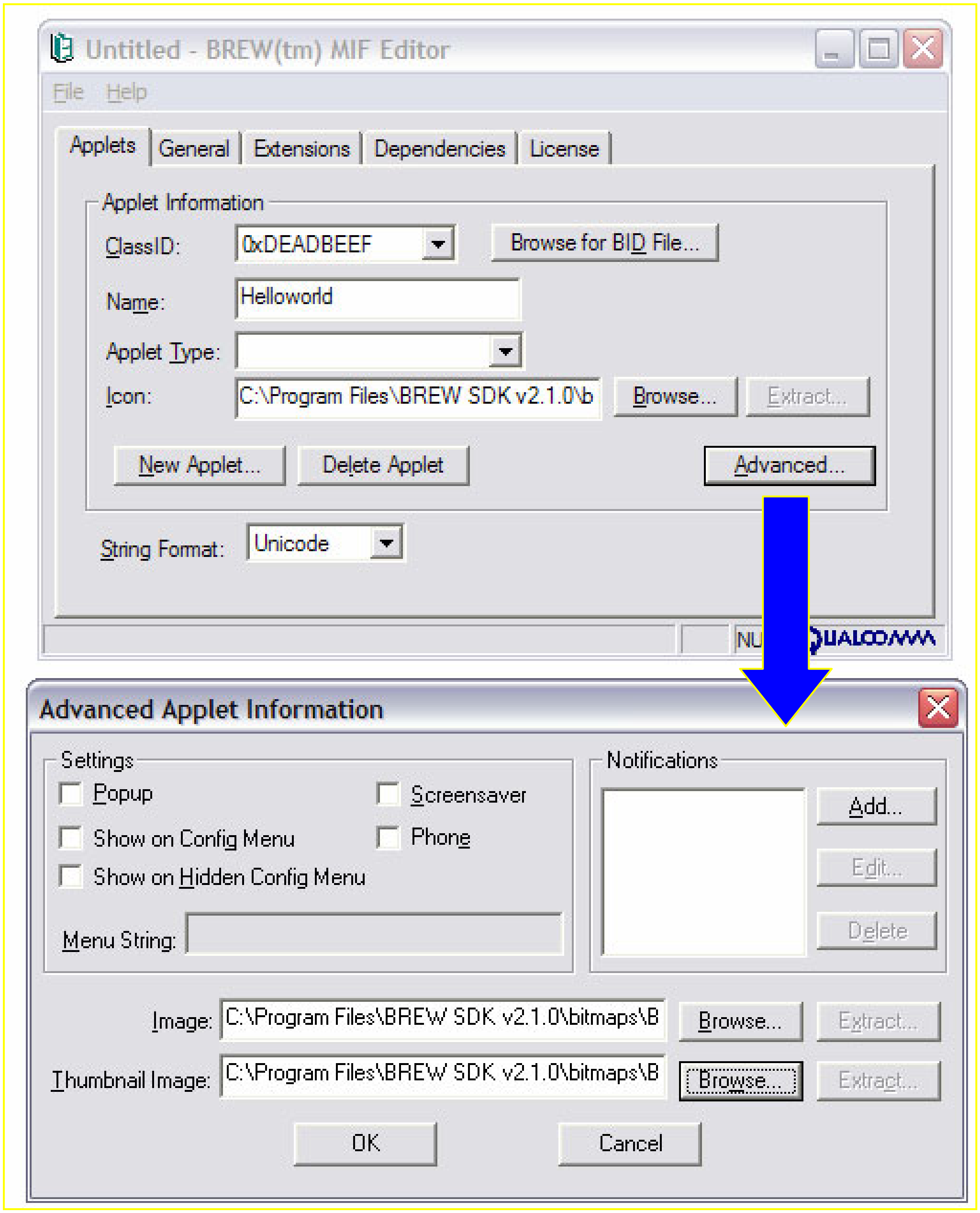
```
#ifndef HELLOWORLD_BID
#define HELLOWORLD_BID

#define AEECLSID_HELLOWORLD    0xDEADBEEF

#endif //HELLOWORLD_BID
```

Figure 8. Sample Class ID file

NOTE: If the class ID specified in your MIF differs from the class ID used in your code, your application will not run properly. Be sure that the two values are identical.

Application name and icons

After creating the BID file, the class ID field of the MIF is automatically updated with the newly generated value. The next step is to specify your application's name and icons, which are used to display your application within the BREW Application Manager. Enter the title of your application in the Name field, and then select a valid image for the Icon. Additional images may be added by clicking **Advanced**, and modifying the Image and Thumbnail fields (see Figure 9. Specifying the application title and icons).



ClassID: 0xDEADBEEF
Name: Helloworld
Applet Type:
Icon: C:\Program Files\BREW SDK v2.1.0\b
String Format: Unicode

Popup
Screensaver
Show on Config Menu
Phone
Show on Hidden Config Menu
Menu String:
Image: C:\Program Files\BREW SDK v2.1.0\bitmaps\B
Thumbnail Image: C:\Program Files\BREW SDK v2.1.0\bitmaps\B

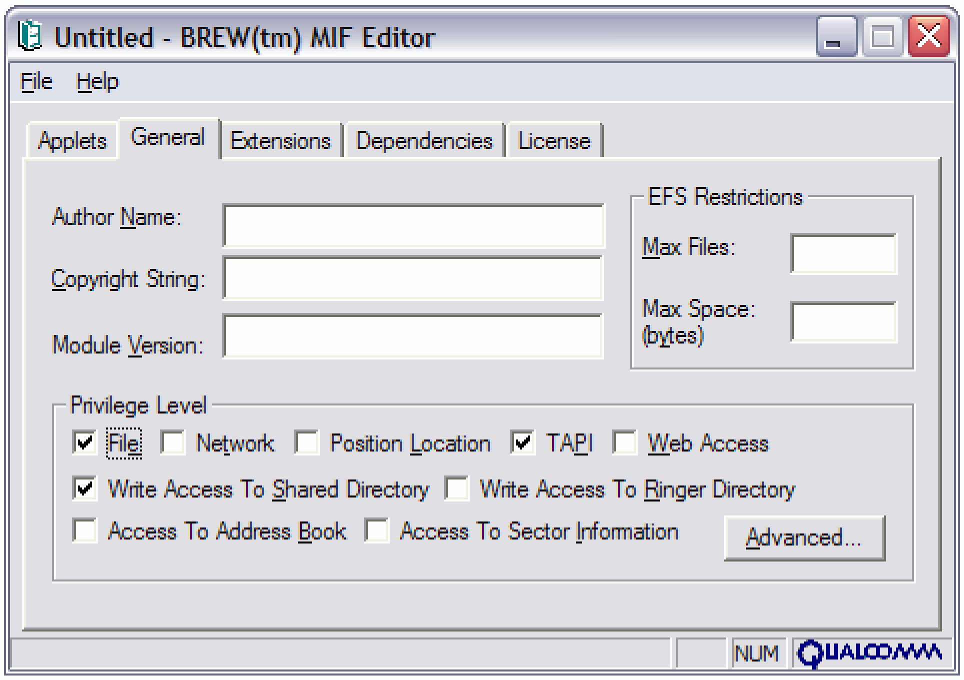
Figure 9. Specifying the application title and icons

NOTE: To ensure compatibility between various BREW devices, the maximum dimensions of your images should be:

Icon:	26*26
Thumbnail:	16*16
Image:	65*42

Setting privileges

If your application uses APIs that require application privilege levels, you must modify the MIF privilege levels accordingly, or your application will fail. To specify privilege levels, click **General**, and click the boxes corresponding to the required privilege levels (see Figure 10. Specifying privileges). Information on the privilege levels required for various APIs is found in the *API Reference Guide*.



Author Name:
Copyright String:
Module Version:

EFS Restrictions
Max Files:
Max Space: (bytes)

Privilege Level
☑ File ☐ Network ☐ Position Location ☑ TAPI ☐ Web Access
☑ Write Access To Shared Directory ☐ Write Access To Ringer Directory
☐ Access To Address Book ☐ Access To Sector Information

Figure 10. Specifying privileges

After making these modifications to the MIF file, click **File > Save** using the name of your application. The preferred directory structure for your application directories and MIFs is shown in Figure 11. BREW directory structure. Placing the MIFs and application directories in the same folder simplifies the process of specifying directory settings within the Simulator, and models the actual file system arrangement of the BREW device.

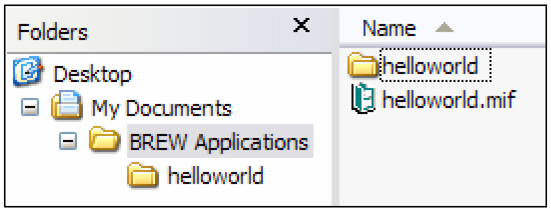


Figure 11. BREW directory structure

Additional resources

The MIF settings covered in this section are vital to the BREW environment and must be specified prior to starting your application. As you become more experienced in BREW, you may modify more of the MIF's attributes. Consult the *BREW MIF Editor Guide* for more detailed assistance on using the BREW MIF Editor.

Using the BREW Resource Editor

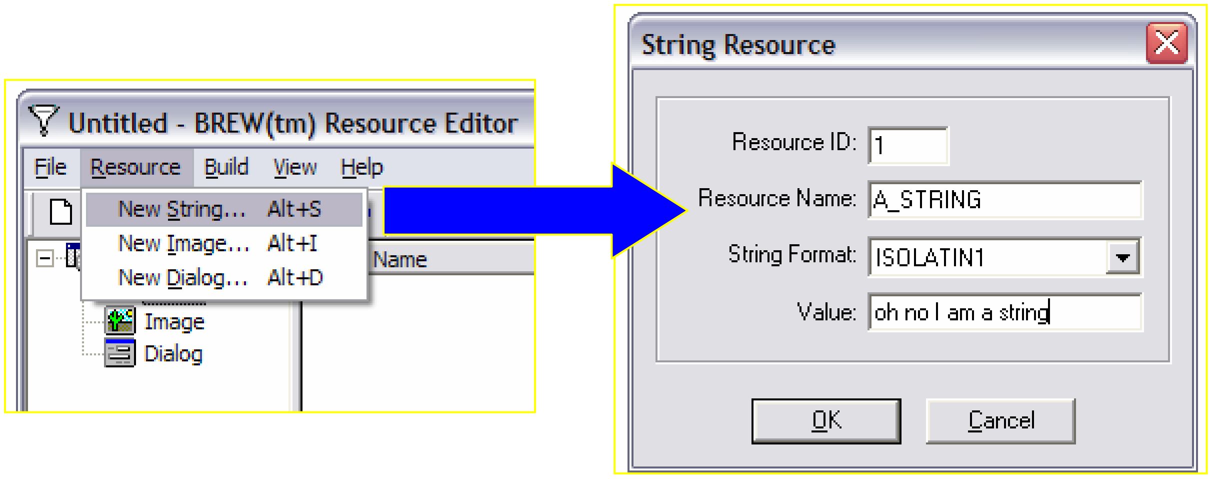
Resource files are an important part of ensuring code portability across various mobile devices, each with its own limitations and capabilities. For instance, by storing all strings used by the application within a resource file, you allow portability of the application to another language by simply creating a new resource file with the translated strings. You may code the application to check the language settings used on the device, and load the appropriate resource file at runtime.

Another example of a situation where resources aid in code portability is the case in which an application is developed for devices with widely varying screen dimensions. By creating device-specific resource files containing images properly sized for the target device, porting the application becomes a matter of switching resource files. Some carriers also place a restriction on the number of files a BREW application may contain; resource files provide a convenient solution to reducing an application's file count.

Creating string resources

To add a new string to your resource file, click **Resource > New String**. This action opens the Image Resource creation window (see Figure 12. Adding string resources). Within this window, you specify the Resource Name (used to access the resource), the String Format (be sure that your target device supports the selected character encoding), and Value (actual string). Optionally, you may modify the assigned Resource ID. Click **OK** after completing the fields in this window.

NOTE: Resource IDs must be unique across resource types within the resource file; error messages appear if duplicate IDs are entered.

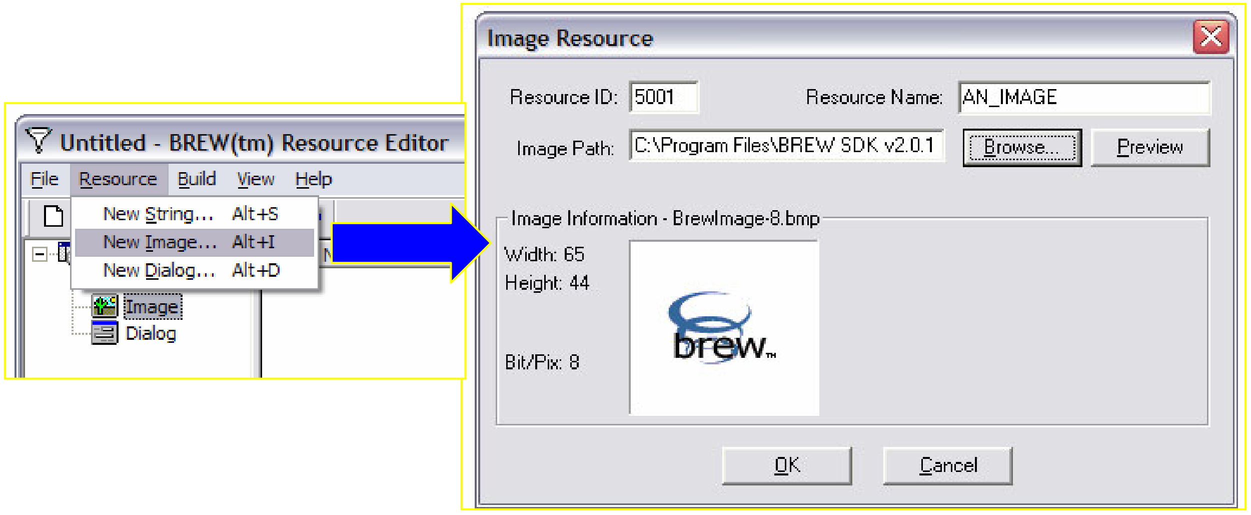


Resource ID: 1
Resource Name: A_STRING
String Format: ISOLATIN1
Value: oh no I am a string

Figure 12. Adding string resources

Creating image resources

Adding an image resource to the resource file is accomplished through a similar procedure. From the Resource menu, select **New Image**, and the Image Resource window opens (see Figure 13. Adding image resources). Enter the Resource ID and Resource Name, click **Browse,** and select a valid image in a supported file format. A preview of the image appears within the Image Resource window; click **OK** if everything looks correct.



Resource ID: 5001
Resource Name: AN_IMAGE
Image Path: C:\Program Files\BREW SDK v2.0.1
Image Information - BrewImage-8.bmp
Width: 65
Height: 44
Bit/Pix: 8

Figure 13. Adding image resources

Creating sound resources

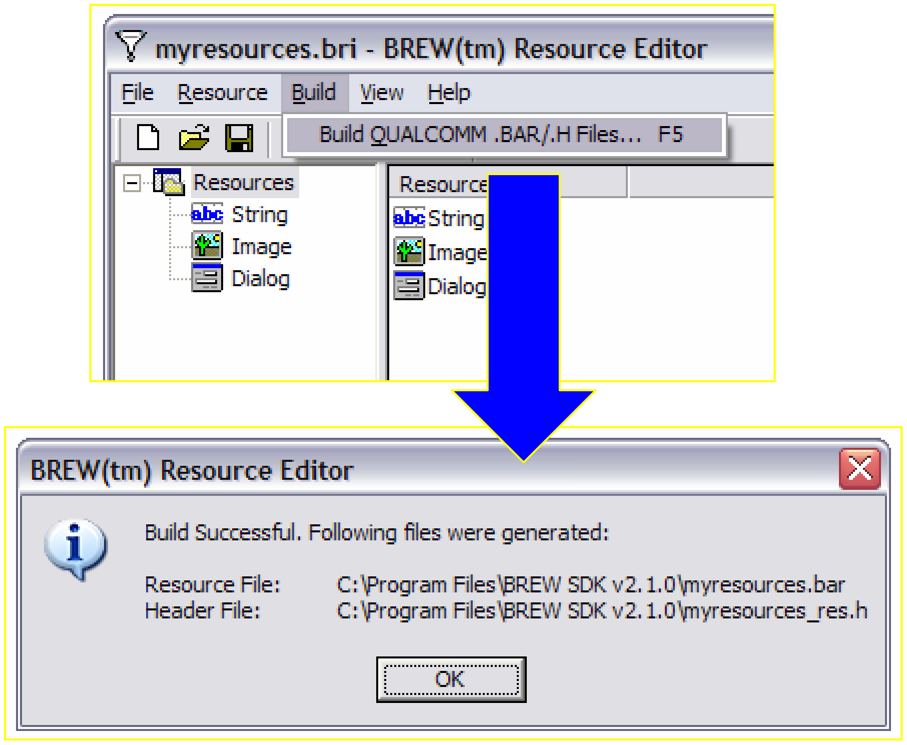
As of this writing, the BREW Resource Editor does not provide native support for sound files. However, sound files may be added as images to work around this issue. To add a sound resource, follow the above steps for adding image resources, but enter your sound file into the Image Path rather than an image.

Within your application, use the ISHELL_LoadResData() or ISHELL_LoadResDataEx() methods to retrieve the information stored in the resource with ResType parameter RESTYPE_IMAGE. This returns a pointer to a memory block containing a MIME header and the raw data. The first byte in the MIME header indicates the offset where the actual raw data begins. To access the raw sound data, you must do something similar to the following example:

```
void * ptrMIME, ptrData;

  // get a pointer to the beginning of the (header + raw data) memory block
ptrMIME = ISHELL_LoadResData(pIShell, MYRESOURCEFILE, MYRESOURCEID,
RESTYPE_IMAGE);

  // add the data offset to find where data begins
ptrData = (byte *)ptrMIME + *(byte *)ptrMIME;
```

Compiling resource files for use in BREW

After adding all resources to the resource file, you must compile it. Resource files are stored in the BREW Resource Intermediate (BRI) format until compilation, at which point the BREW Applet Resource (BAR) file is created, along with a resource header file used to access the resources within your code. To compile the BRI file, select **Build Qualcomm .BAR/.h Files** from the Build menu. If the compilation operation is successful, a confirmation window opens, listing the files generated as a result (see Figure 14. Compiling a BRI file).


myresources.bri - BREW(tm) Resource Editor
File Resource Build View Help
Build QUALCOMM .BAR/.H Files... F5
Resources
String
Image
Dialog
BREW(tm) Resource Editor
OK

Build Successful. Following files were generated:

Resource File: C:\Program Files\BREW SDK v2.1.0\myresources.bar
Header File: C:\Program Files\BREW SDK v2.1.0\myresources_res.h

Figure 14. Compiling a BRI file

This resulting header file (see Figure 15. Sample resource header file) is included within your application, where the constants are used to access the resources within the file.

```
#ifndef MYRESOURCES_RES_H
#define MYRESOURCES_RES_H

// WARNING: DO NOT MODIFY THIS FILE
// AUTO-GENERATED BY BREW Resource Editor


#define MYRESOURCES_RES_FILE "myresources.bar"

#define A_STRING        1
#define AN_IMAGE        5001


#endif  // MYRESOURCES_RES_H
```

Figure 15. Sample resource header file

Additional resources

The BREW Resource Editor is also used to create a wide variety of dialog control types, but that is a topic beyond the scope of this guide. For additional usage information on this application, read the *BREW Resource Editor Guide*.

Using the BCI Authoring Tool

The BCI Authoring tool provides a simple interface for converting images into the BREW Compressed Image (BCI) format and creating BCI animations. Compressing image files reduces the size of the image and increases the display speed, both of which provide desirable benefits.

Creating BCI images

This process is as simple as opening a supported graphics file and then resaving it as a BCI. To open an image, click the File menu and select **Open Image File**. Browse to an image file in a supported graphics type and open it; the image appears in the BCI Authoring Tool window (see Figure 16. Opening image files).

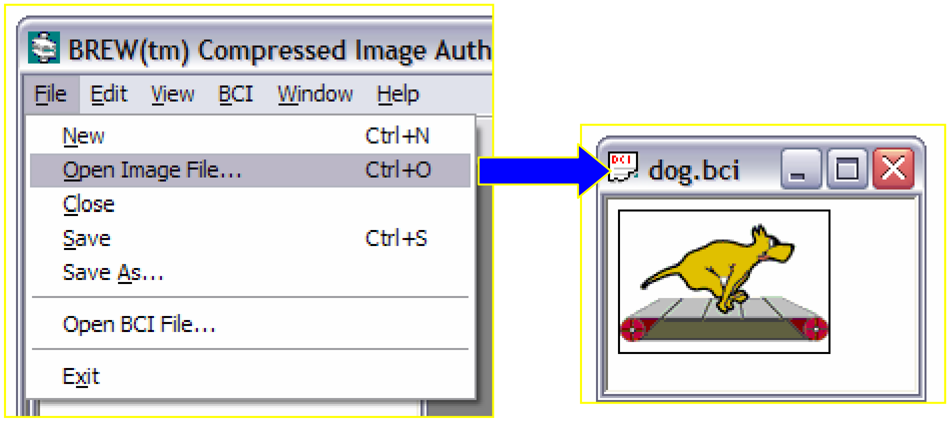


Figure 16. Opening image files

If everything looks as expected, select **Save As** from the File menu, and save the new image file.

Animating BCI images

Adding animation to a BCI image is also a very simple procedure. Follow the steps above to open all the frames in your animation; the images are appended to your animation in the order they are opened (see Figure 17. BCI animation frames).

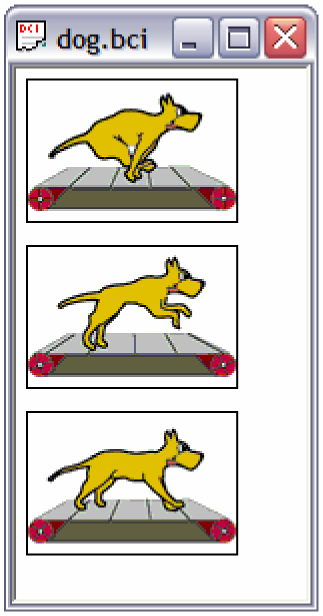


Figure 17. BCI animation frames

The default frame duration for each frame in your animation is 150 ms.To modify this value, click the desired frame and select **Frame Duration** from the Edit menu. A dialog box opens, allowing you to specify a new duration for the frame (see Figure 18. Modifying frame durations).

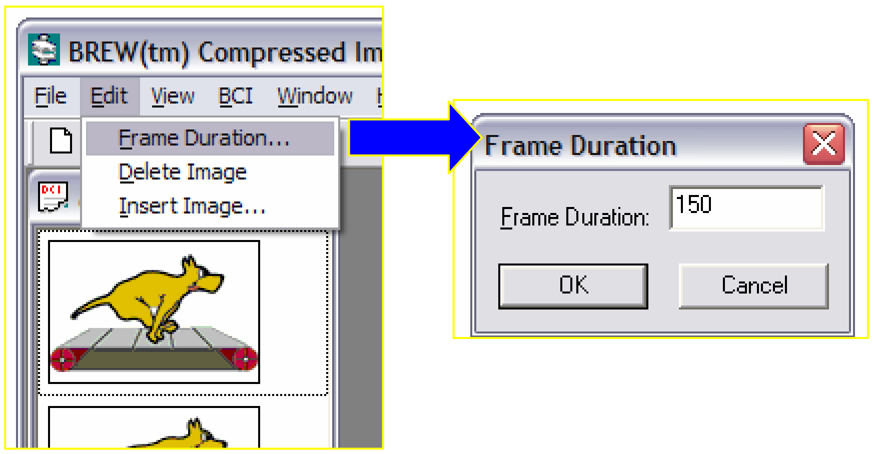


Figure 18. Modifying frame durations

After saving your BCI animation, you can preview the result by selecting **Animate** from the BCI menu (see Figure 19. Previewing animation). The resulting animation plays, allowing you to judge if the frame durations have been appropriately specified to achieve the proper animation effect.

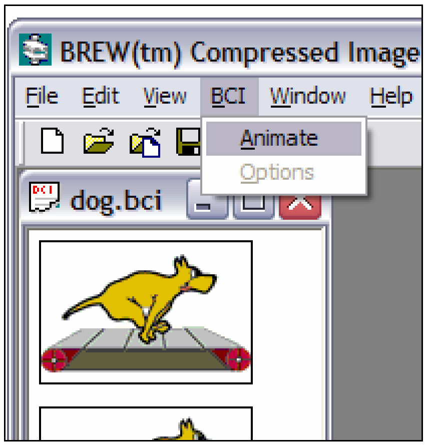


Figure 19. Previewing animation

Additional resources

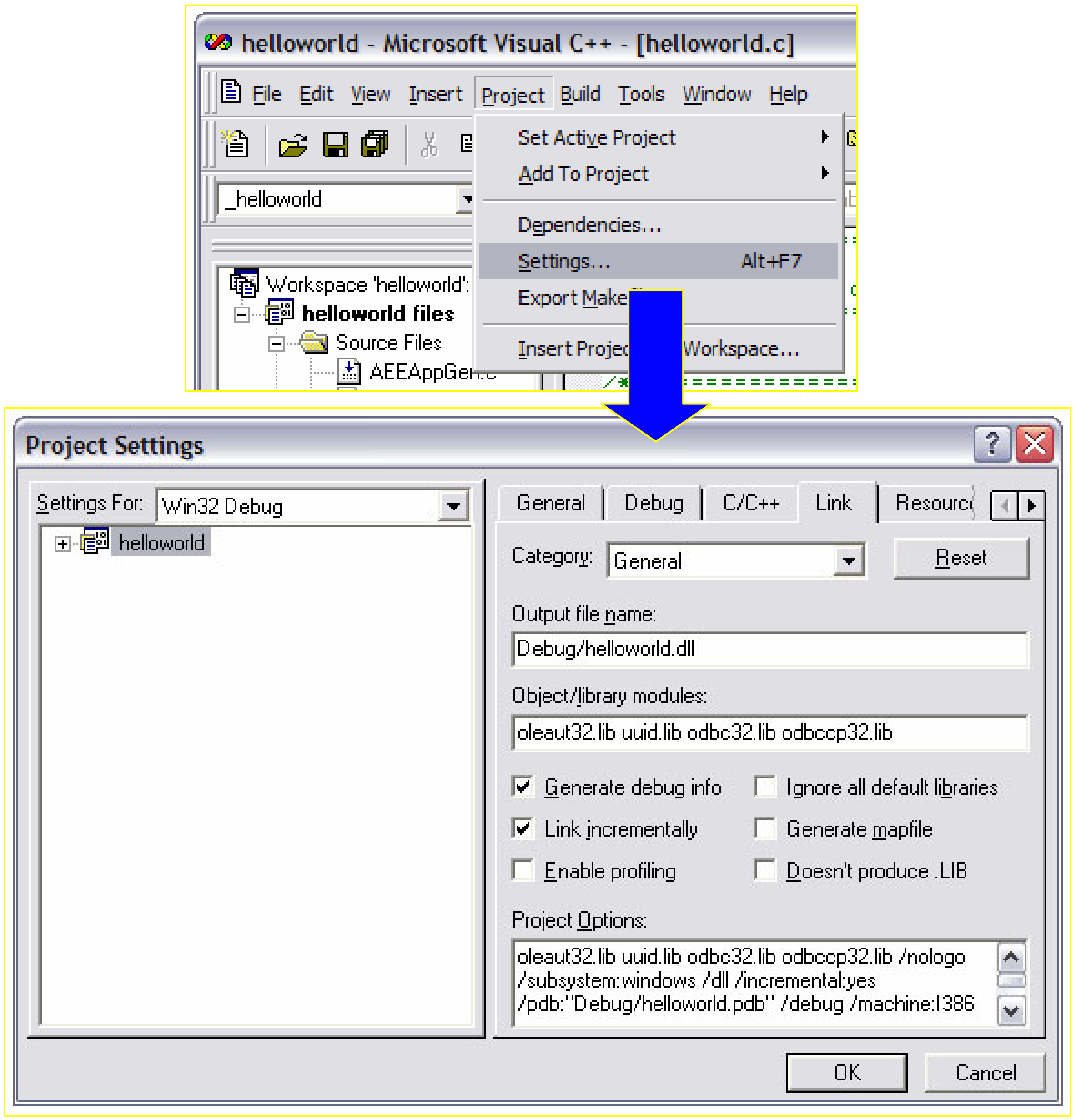
Consult the *BREW BCI Guide* for more help on using the BCI Authoring Tool.

Compiling BREW Applications for Simulation

After writing your BREW application, you can build a DLL for use by the Simulator. First, ensure that you created a MIF for your application and generated a valid class ID file (see the Using the BREW MIF Editor for more details).

Specifying DLL destination

The default location for your DLL file is a subdirectory created within your application directory named Debug. Unfortunately, the Simulator is not able to execute your application properly unless the application DLL resides within the application directory. To specify the proper destination for your project DLL, change your project's linking options by selecting **Project > Settings > Link**. The Output file name is listed as Debug/<projectname>.dll; remove the Debug/ prefix (see Figure 20. Changing DLL output).



Output file name:
Debug/helloworld.dll

Object/library modules:
oleaut32.lib uuid.lib odbc32.lib odbccp32.lib

- [✓] Generate debug info
- [ ] Ignore all default libraries
- [✓] Link incrementally
- [ ] Generate mapfile
- [ ] Enable profiling
- [ ] Doesn't produce .LIB

Project Options:

```
oleaut32.lib uuid.lib odbc32.lib odbccp32.lib /nologo
/subsystem:windows /dll /incremental:yes
/pdb:"Debug/helloworld.pdb" /debug /machine:I386
```

Figure 20. Changing DLL output

Building an application DLL

After changing the default DLL output directory, your application is ready to compile into a DLL. This process is accomplished by selecting **Build <projectname>.dll** from the Build menu (see Figure 21. Building the DLL).

NOTE: The DLL file is only used by the Simulator. Compiling your application into a MOD for the BREW device requires a separate procedure, detailed in the Loading BREW Applications to the Mobile Device section.

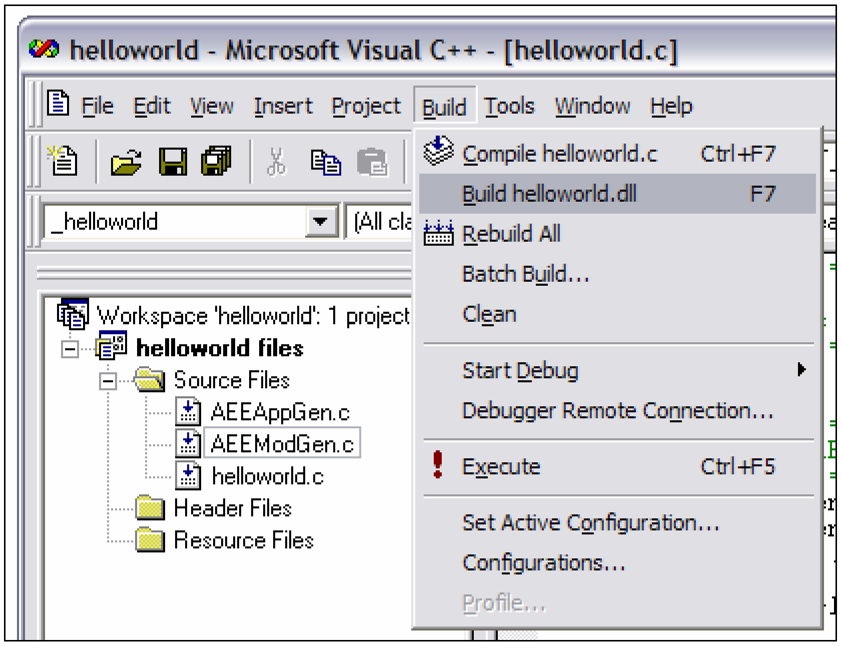

helloworld - Microsoft Visual C++ - [helloworld.c]
File Edit View Insert Project Build Tools Window Help
_helloworld
Workspace 'helloworld': 1 project
helloworld files
Source Files
AEEAppGen.c
AEEModGen.c
helloworld.c
Header Files
Resource Files
Compile helloworld.c Ctrl+F7
Build helloworld.dll F7
Rebuild All
Batch Build...
Clean
Start Debug
Debugger Remote Connection...
Execute Ctrl+F5
Set Active Configuration...
Configurations...
Profile...

Figure 21. Building the DLL

If the build process does not complete successfully, you must debug your application. For more information on solving common BREW code errors, see the Common Issues section.

Running your application

After successfully building your application DLL, you can choose to run it within the Simulator. To do so, select **Execute** from the Visual Studio Build menu (see Figure 22. Running the DLL).

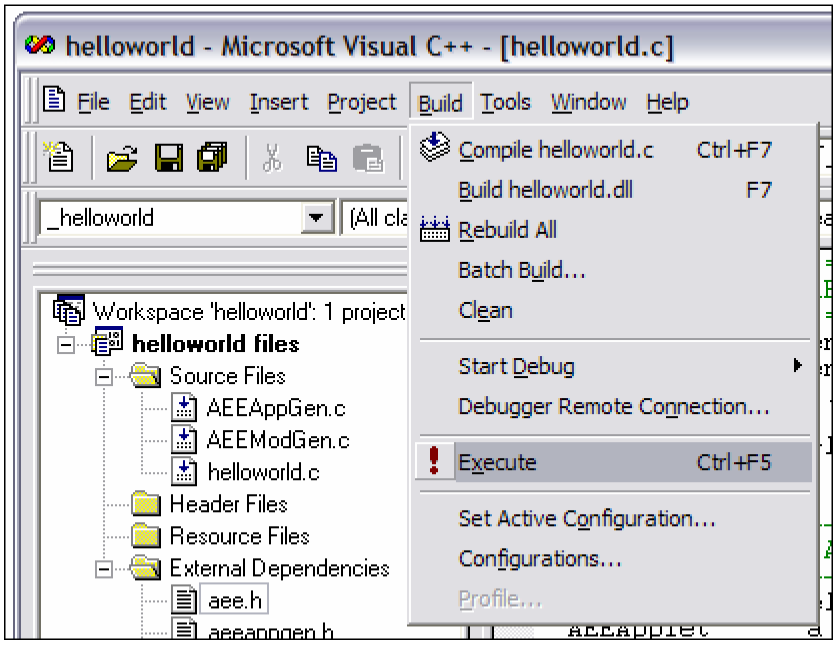


Figure 22. Running the DLL

The first time you run your application, you are prompted to enter the name of an executable; you must select **Browse** and specify the appropriate version of the Simulator from the Bin directory of your BREW SDK installation (see Figure 23. Specifying the executable).

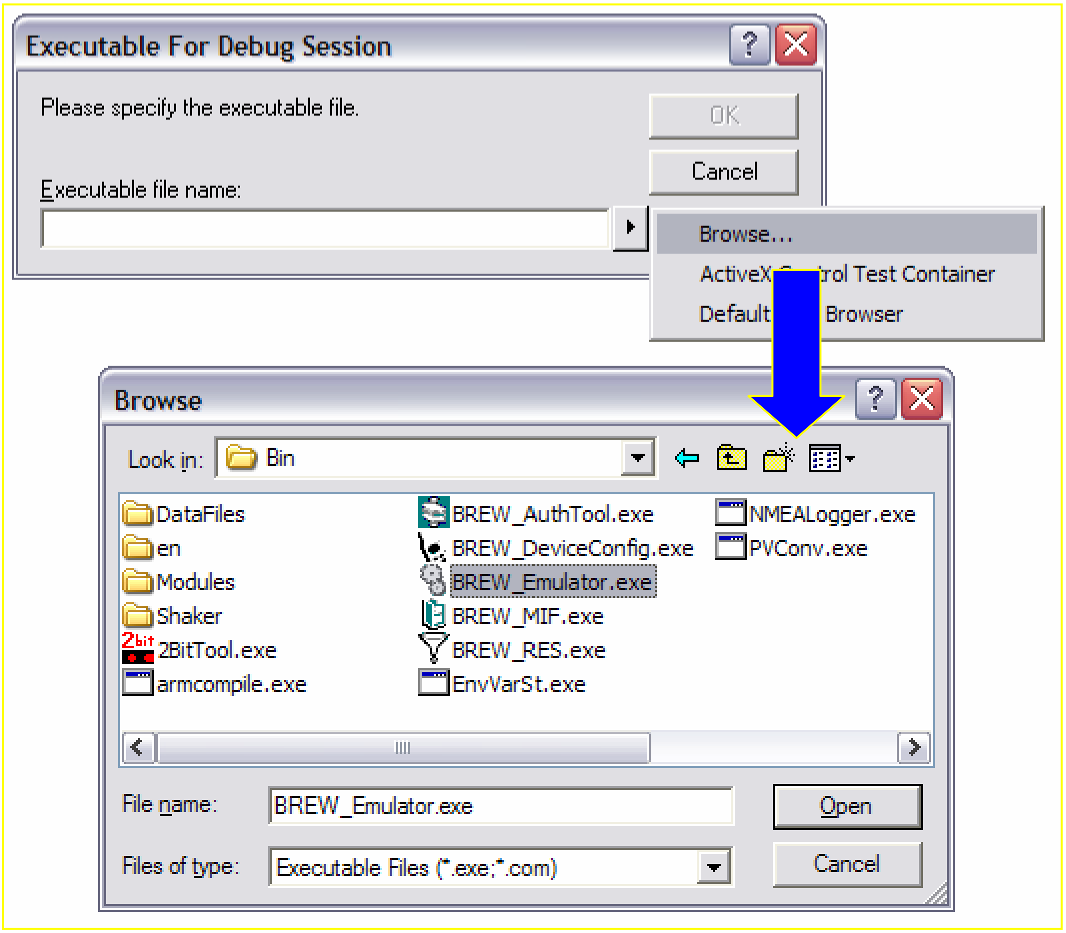


Figure 23. Specifying the executable

Common issues

Many issues encountered during the compilation process are the result of linking errors. The following steps are designed to assist in resolving most linking errors you may encounter while programming in BREW.

Read the BREW API Reference

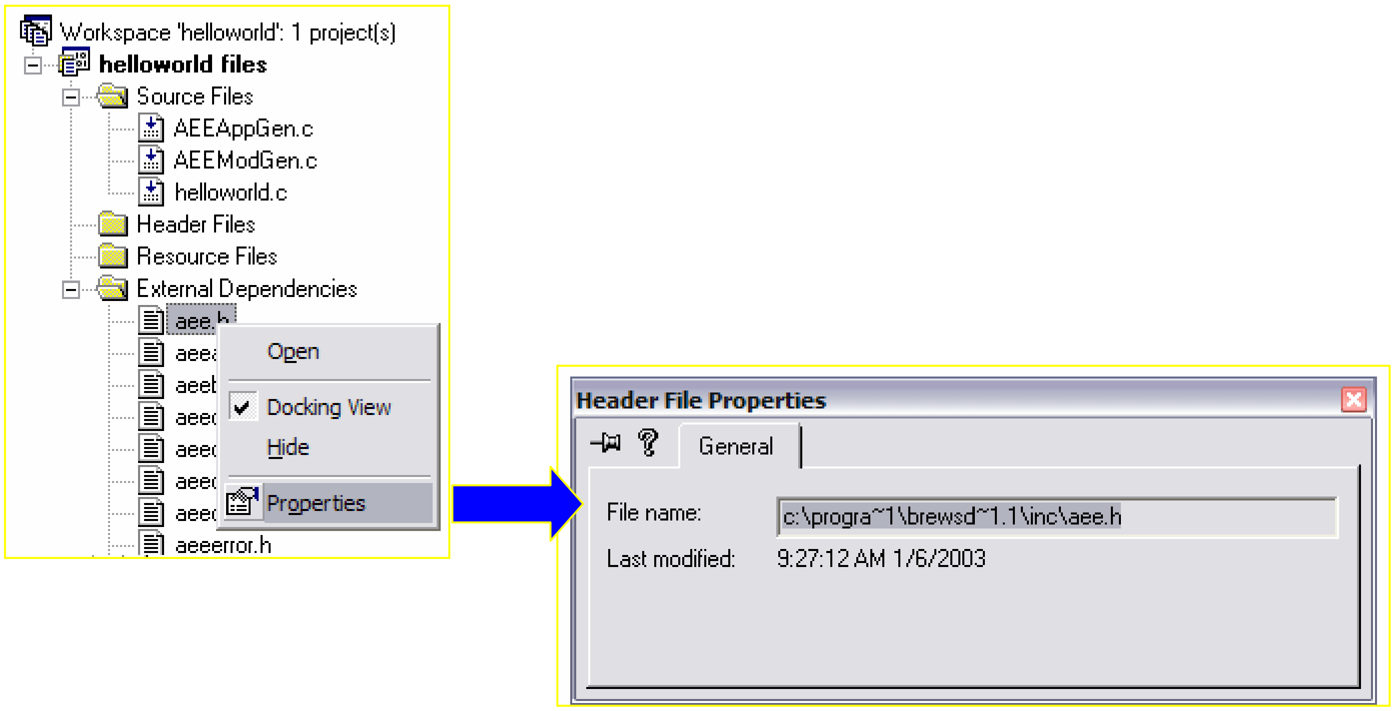
Verify that you are using the correct names for BREW constants, methods, interfaces, and structure members.

Include all necessary header files

If header files are not included, the compiler cannot recognize the symbols specified in your code. The *BREW API Reference* contains a listing of the required header files for each interface.

Ensure linking is performed against the proper library versions

If your BREWDIR environment variable was not properly specified for the targeted version of BREW, applications created using the BREW Application Wizard will link projects to the incorrect version of libraries. To check that the proper libraries are used, right-click on the files within your project workspace window, and select **Properties**. In the resulting Header File Properties window, verify that the File name is correct for the intended BREW version (see Figure 24. Checking library versions). For information on changing the BREWDIR variable, see Changing the BREWDIR environment variable.



File name: c:\progra~1\brewsd~1.1\inc\aee.h

Last modified: 9:27:12 AM 1/6/2003

Figure 24. Checking library versions

Close any applications using your DLL

If another application is currently using your DLL (for example, the Simulator), compilation fails with an error message of

```
LINK : fatal error LNK1168: cannot open helloworld.dll for writing
Error executing link.exe.
```

To solve this problem, Visual Studio must have complete access to the DLL.

Additional resources

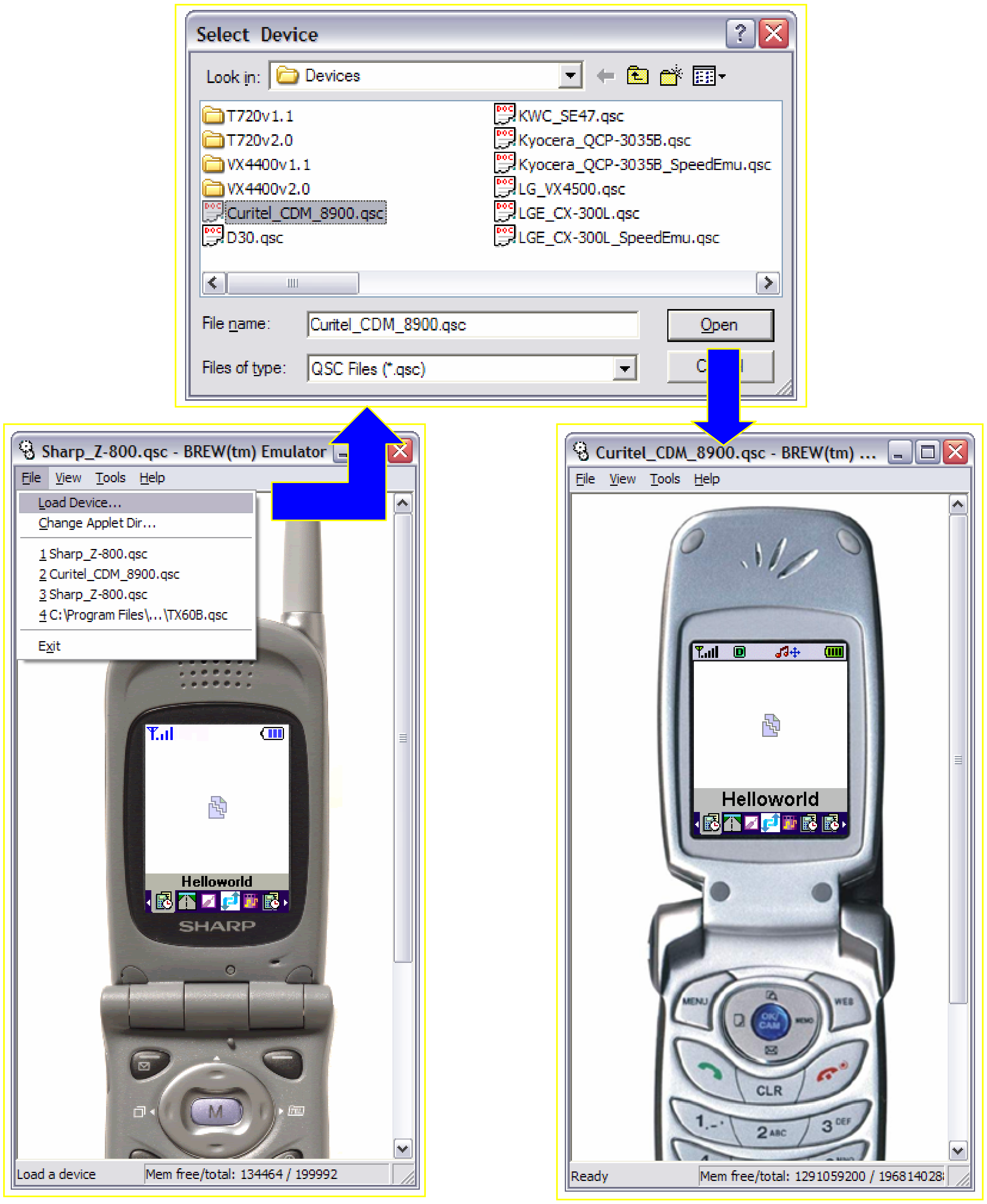
The *BREW API Reference* guide proves valuable in diagnosing most errors encountered during the BREW compilation process.

Running BREW Applications on the Simulator

The BREW Simulator provides a convenient environment for running and debugging your applications prior to transferring them to a mobile device.

Device skins

The Simulator is installed along with a default set of device skins (QSC files), located in the Devices folder of your BREW installation. Additional device skins are available for download by authenticated developers from the Phone Details page. To change the device skin used by the Simulator, select **Load Device** from the File menu. Select a new device skin and click **Open**; the Simulator's appearance is modified to reflect the specified skin (see Figure 25. Changing device skins).



File name: Curitel_CDM_8900.qsc

Files of type: QSC Files (*.qsc)

Load a device | Mem free/total: 134464 / 199992

Ready | Mem free/total: 1291059200 / 196814028

Figure 25. Changing device skins

NOTE: Device skins allow you to test your application using the appearance and limitations of the intended device, but they do not completely replicate the specific device environment.

The Simulator and its device skins should not replace testing your application directly on the device. Many phones have known issues are not reproduced in the Simulator.

Modifying device skins

Prior to BREW 3.0, the Device Configurator handles the modifications of attributes of device skins. This functionality permits developers to specify the attributes of the device used for emulation, though the OEM typically provides the appropriate values. Developers who have not been authenticated and cannot download new device skins may also want to modify the device skin to more closely reflect the specifications of the targeted mobile platform. For more information on using the Device Configurator, see the BREW Device Configurator Guide.

As of BREW 3.0, the BREW Simulator incorporates the functionality of the Device Configurator with the use of Device Packs (DPK files). Device Packs are modifiable directly within the Simulator; see the BREW Simulator section of the BREW SDK User documents for more details.

Specifying Simulator directories

To locate and load your application, the Simulator must be configured with the proper directories for loading MIFs and applets. Failing to configure the Simulator with the correct directories causes errors when trying to run your application.

The MIF directory is the directory that contains your MIF files, while the application directory is the parent directory containing all your applet directories. For instance, in Figure 26. Sample directory structure, the BREW applications directory is both the MIF directory and the application Directory.

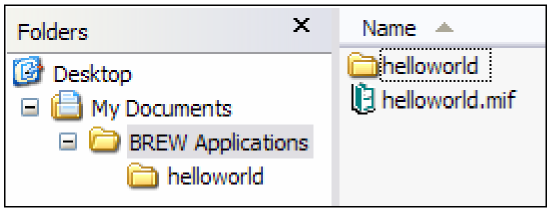


Figure 26. Sample directory structure

In the Simulator, the application directory is specified by selecting **Change Applet Dir** from the File menu. In the following window, you should select the directory containing your applet folders (see Figure 27. Changing the applet directory).

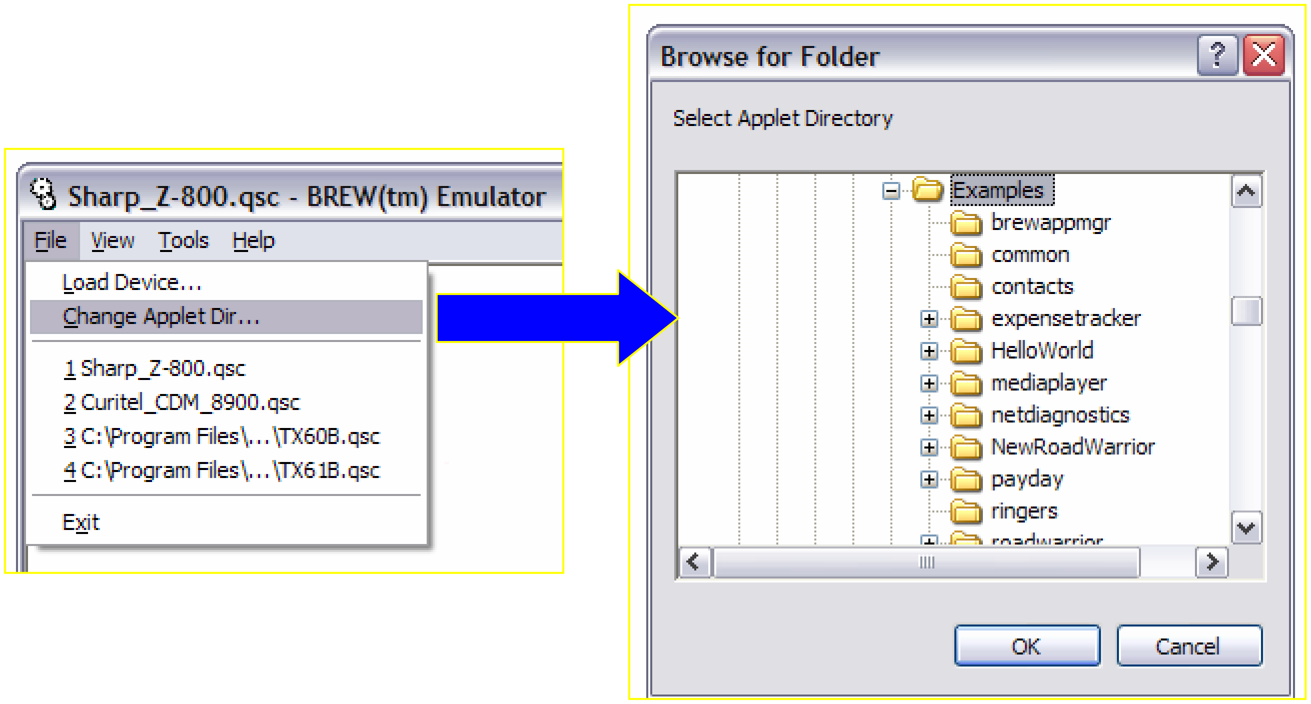
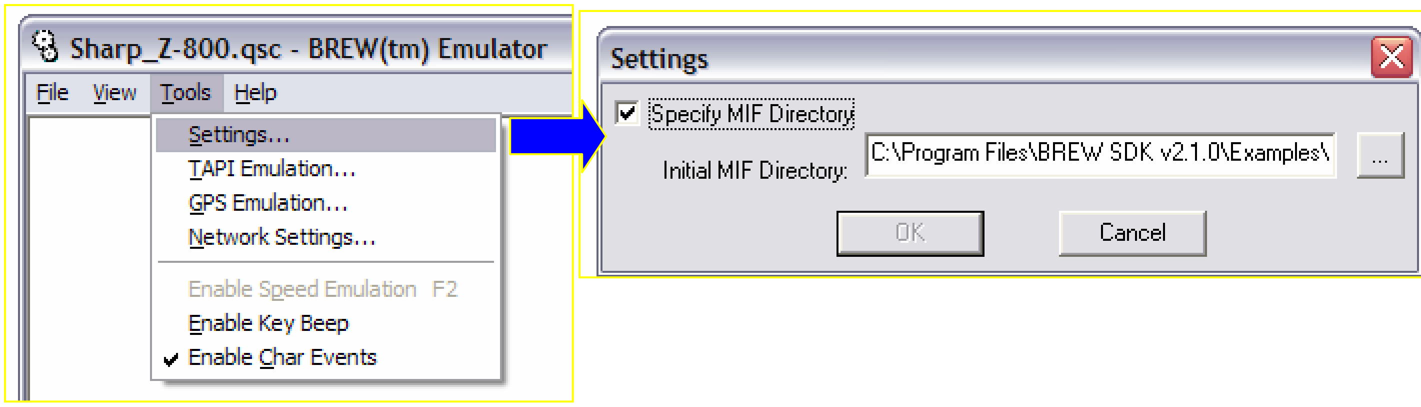


Figure 27. Changing the applet directory

The MIF Directory may be changed by selecting **Settings** from the Tools menu (see Figure 28. Changing the MIF directory). If you use the same directory to hold both your application folders and your MIFs, clear the **Specify MIF Directory** check box; otherwise you will need to select the appropriate MIF directory and click **OK**.

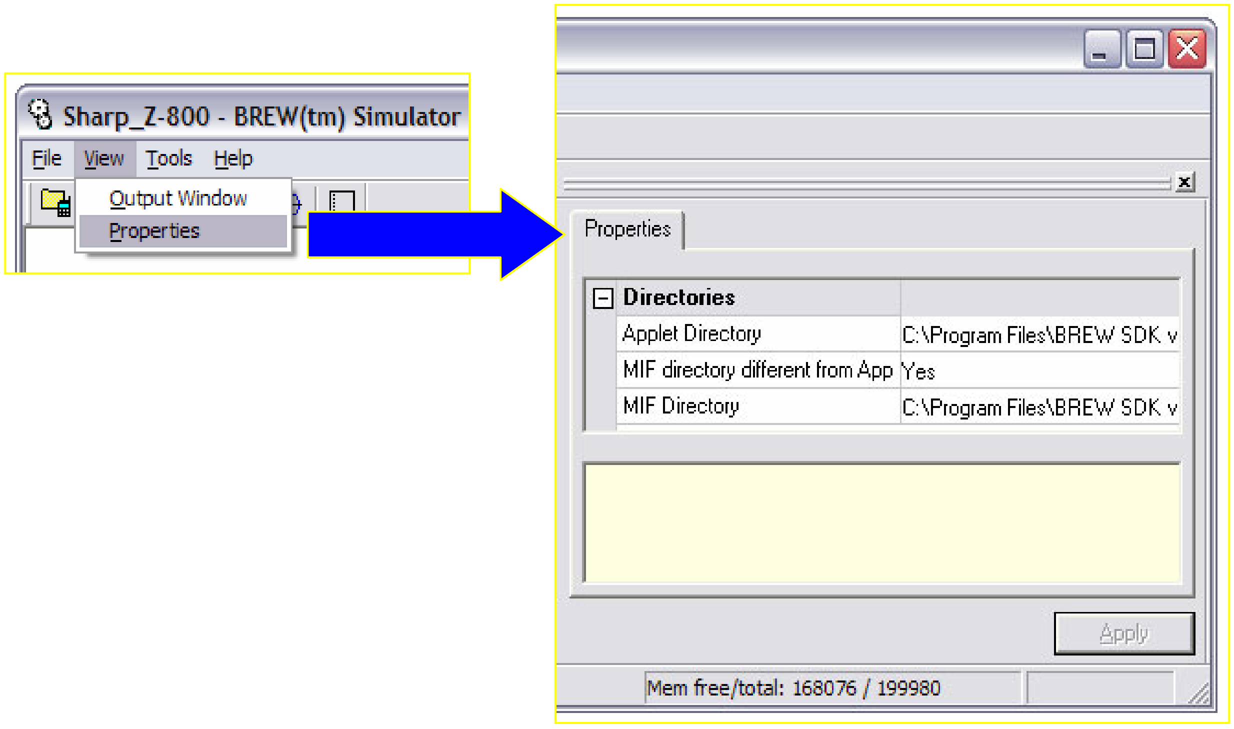


[✓] Specify MIF Directory

Initial MIF Directory: C:\Program Files\BREW SDK v2.1.0\Examples\

Figure 28. Changing the MIF directory

In the BREW Simulator, the application and MIF directories may be changed through the Properties tab. If the Properties tab is not enabled, show it by selecting **Properties** in the View menu (see Figure 29. Changing the MIF and application directories in the Simulator). Within this tab, change the applet and MIF directories by modifying the corresponding fields, and click **Apply** to confirm the changes.



Directories	
Applet Directory	C:\Program Files\BREW SDK v
MIF directory different from App	Yes
MIF Directory	C:\Program Files\BREW SDK v

Mem free/total: 168076 / 199980

Figure 29. Changing the MIF and application directories in the Simulator

Running your application

After the Simulator has been properly configured with the correct MIF and application directories, run your application by simply navigating to the proper icon within the application menu (see Figure 30. Application menu), using the mouse to interact with the device buttons or using the keyboard for mapped keys. Click **Select** on the device skin or press **Enter** on your keyboard to start the highlighted application.

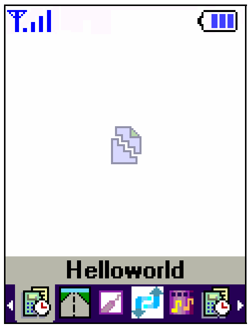


Figure 30. Application menu

Advanced Simulator capabilities

The Simulator has the capability of mimicking external stimulus to your application, such as GPS readings or incoming voice calls and SMS messages. For more information on harnessing these capabilities, see the BREW Emulator section of the *BREW SDK User Guide*, or the BREW Simulator portion of the BREW SDK User documents in BREW 3.0+.

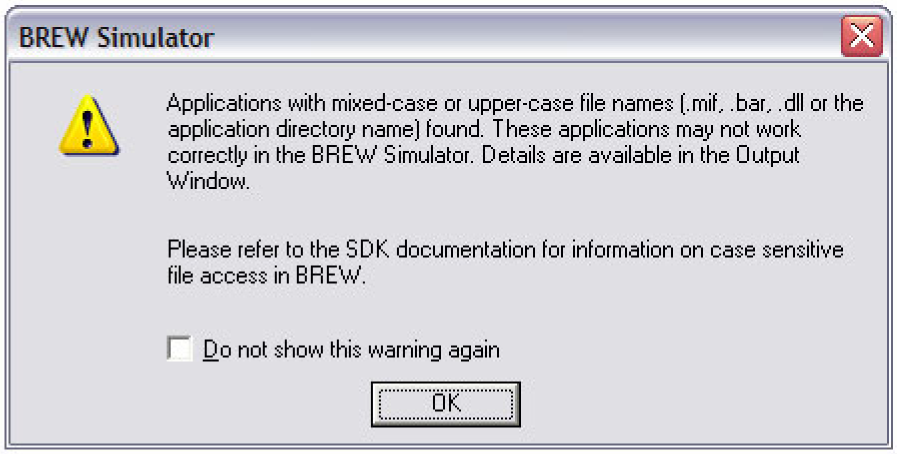
Common issues

ISHELL_CreateInstance() or API methods unexpectedly failing

Not all APIs and API methods are supported by the Simulator. For example, instances of the IVocoder and ICamera interfaces cannot be created on the Simulator. Other API capabilities may not be completely supported, such as sending outgoing SMS messages with ITAPI_SendSMS(). In these cases, you may find that ISHELL_CreateInstance() fails with return value ECLASSNOTSUPPORT. These errors also occur if your application has not specified the necessary privilege levels within your MIF.

Mixed-case name warning displayed

BREW applications may not have uppercase letters in their filenames and application directories. Mixed-case filenames in the Emulator do not generate error messages. The Simulator was modified to imitate the behavior of the BREW environment regarding file/directory naming conventions, and it a warning message (see Figure 31. Mixed case naming warning) appears when it encounters a violation. Failure to follow naming rules may result in you're the malfunction of your application.

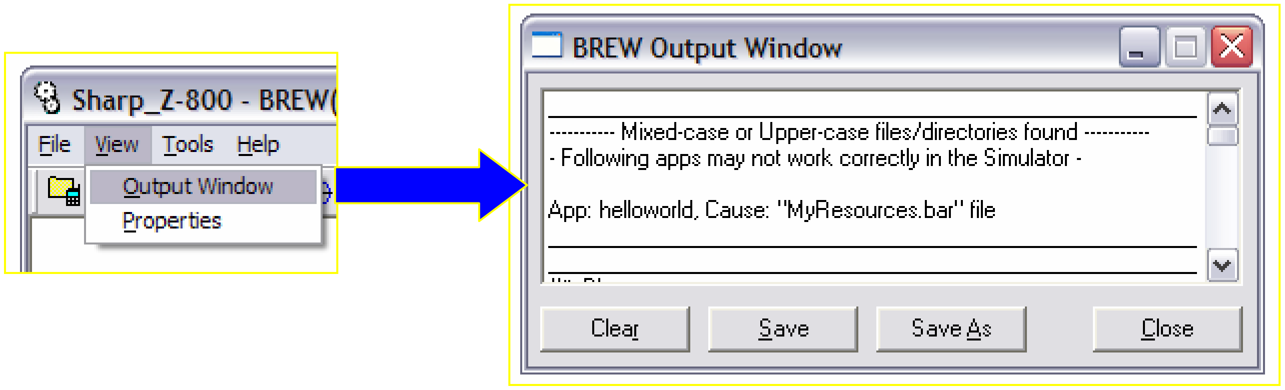

BREW Simulator

Applications with mixed-case or upper-case file names (.mif, .bar, .dll or the application directory name) found. These applications may not work correctly in the BREW Simulator. Details are available in the Output Window.

Please refer to the SDK documentation for information on case sensitive file access in BREW.

Figure 31. Mixed case naming warning

To determine which file and directory names are causing the violation, click **Output Window** and inspect the log; the offending file or directory names are shown (see Figure 32. Resolving naming violations). Rename these files and modify your application code as appropriate.



```
---------- Mixed-case or Upper-case files/directories found ----------
- Following apps may not work correctly in the Simulator -

App: helloworld, Cause: "MyResources.bar" file
```

Figure 32. Resolving naming violations

The application’s icon is not displayed

If you are unable to see your application’s icon within the Application Manager screen, the Simulator is unable to find the MIF file for your application. Make sure that you have created the MIF file using the BREW MIF Editor, and you properly specified the Emulator’s MIF directory (see Specifying Simulator directories).

NOTE: Some device skins may not display your application if you have not provided a valid icon within the MIF. Try switching to another device skin or providing a valid application icon within your MIF.

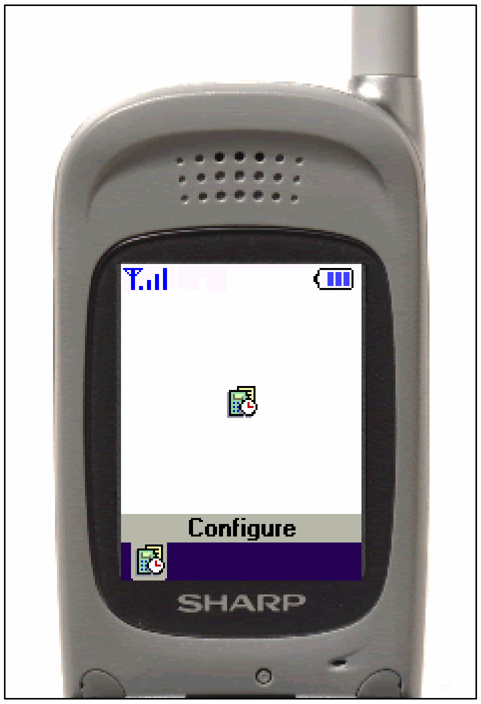
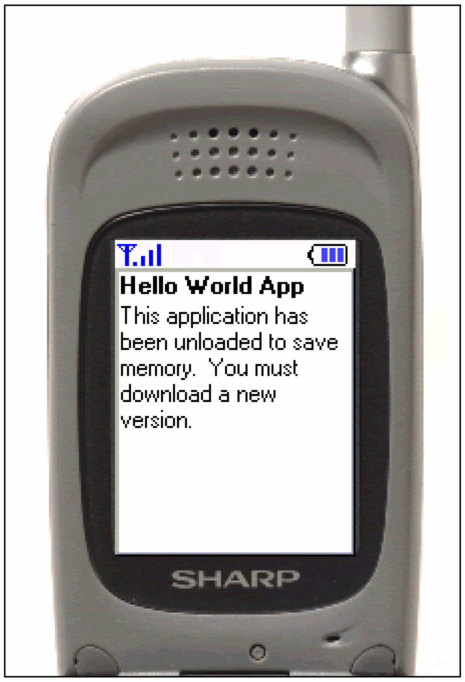


Figure 33. Emulator with incorrect MIF directory

Application has been unloaded error message displayed

If the Simulator shows the error message in Figure 34. Emulator unable to find DLL when you start your application, it was unable to find the proper DLL file. This is frequently caused by an erroneous application directory specification (see Specifying Simulator directories). Verify that your application's directory contains a valid DLL; if not, build your application's DLL in Visual Studio (see Building an application DLL). Also verify that you have configured Visual Studio to write the DLL file within the application directory and not the Debug folder (see Specifying DLL destination).



Hello World App
This application has been unloaded to save memory. You must download a new version.

Figure 34. Emulator unable to find DLL

Unknown error (1) message displayed

This error message (see Figure 35. Application initialization code failure) is displayed when your application returns EFAILED within the AEEClsCreateInstance() method. Typically EFAILED is returned when your class data init function fails, which depends on the implementation of the function. See the above section on ISHELL_CreateInstance() or API methods unexpectedly failing; verify that all your memory allocation statements are completing correctly. You may need to insert debug statements or breakpoints into your code so you can isolate the cause of these error conditions; consult the Debugging BREW applications on the Simulator section below for additional help on this topic. The AEEClsCreateInstance() method also fails if there is a mismatch in the class ID specified in the MIF and the ID included in your source code.

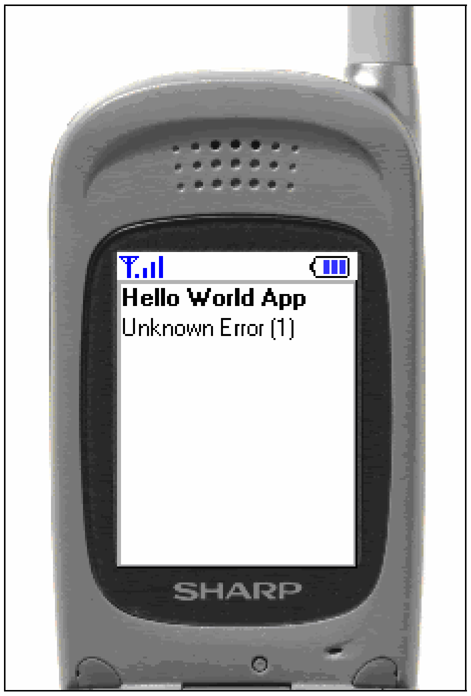
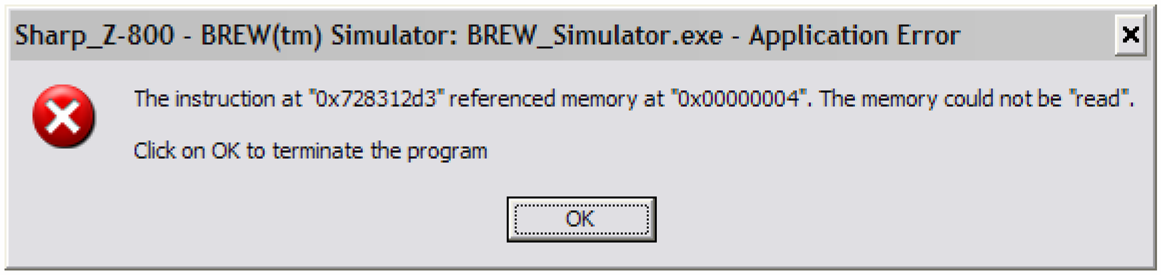

Hello World App
Unknown Error (1)

Figure 35. Application initialization code failure

BREW Simulator fails

If the BREW Simulator fails while starting your application, it is probably a memory access violation, such as accessing a null pointer or attempting to release the same pointer twice. When this occurs, the Simulator fails, and a message similar to the one in Figure 36. Memory access violation, appears. Verify all your memory accesses to verify that you are not performing any illegal operations. For more information on debugging this type of error, refer to Debugging BREW applications on the Simulator below.


Sharp_Z-800 - BREW(tm) Simulator: BREW_Simulator.exe - Application Error

The instruction at "0x728312d3" referenced memory at "0x00000004". The memory could not be "read".

Click on OK to terminate the program

Figure 36. Memory access violation

Additional resources

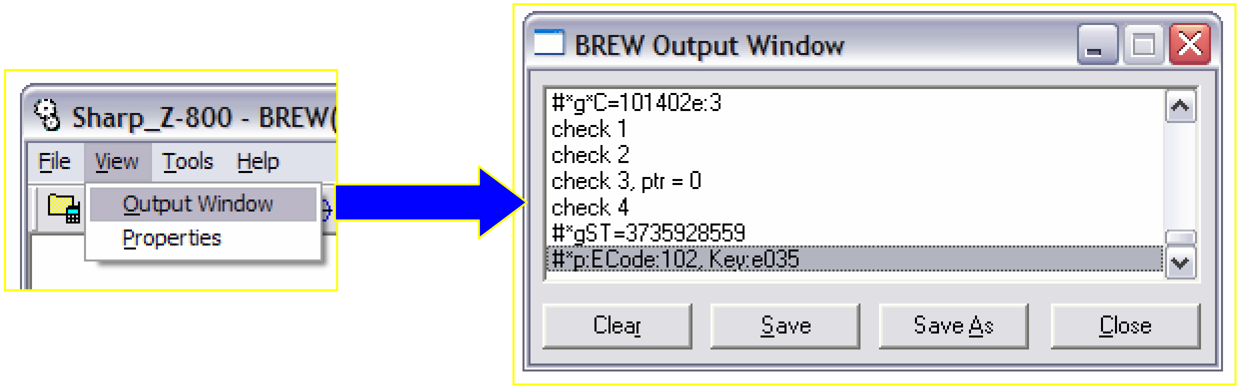
See the section on Troubleshooting BREW within the *BREW SDK User Guide* and BREW SDK User documents. This chapter contains a listing of common error messages encountered while running the Simulator and solutions to these problems.

Debugging BREW applications on the Simulator

There are two primary strategies available for debugging program behavior in BREW. The first is to use DBGPRINTF() for simple error-checking print statements; the other is to debug the application with Visual Studio while running in the Simulator.

DBGPRINTF()

Any information printed using the DBGPRINTF() helper function is displayed in the output window when the application is run in the Simulator, or on the BREW Logger when the application is run on the mobile device. This is helpful for inspecting the value of variables and debugging the application flow of control; however, it is not as robust as taking advantage of the Visual Studio debugging functionality. Inset calls to DBGPRINTF() as necessary in your code, open the output window in the Simulator by selecting **View > Output Window** while your application is running, and observe the results (see Figure 37. DBGPRINTF() output). For more information on the DBGPRINTF() helper function, consult the *BREW API Reference*.



```
#*g*C=101402e:3
check 1
check 2
check 3, ptr = 0
check 4
#*gST=3735928559
#*p:ECode:102, Key:e035
```

Figure 37. DBGPRINTF() output

Debugging in Visual Studio

To launch a debug session in Visual Studio, select **Start Debug > Go** in the Build menu (see Figure 38. Starting a debug session). A warning window (see Figure 39. Debugging information warning) opens the first time you debug your application; this warning window may be safely ignored.

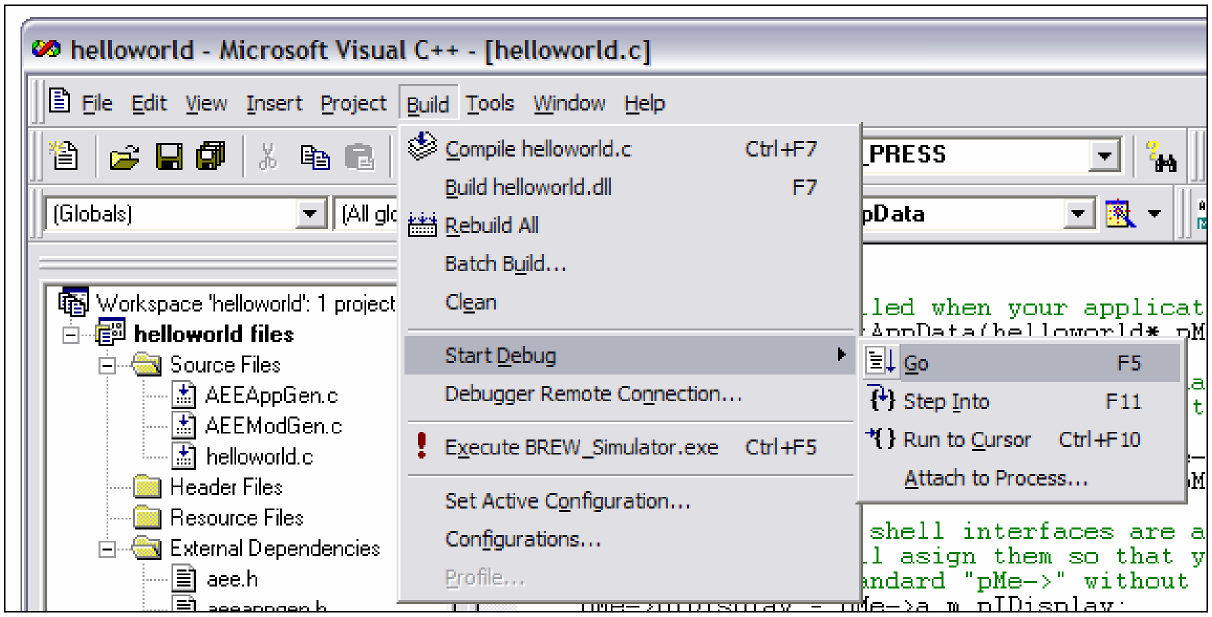
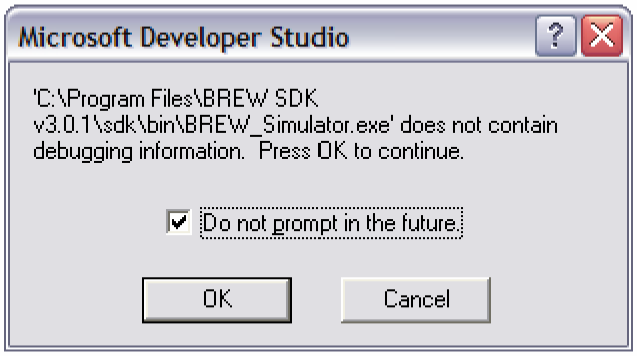


Figure 38. Starting a debug session

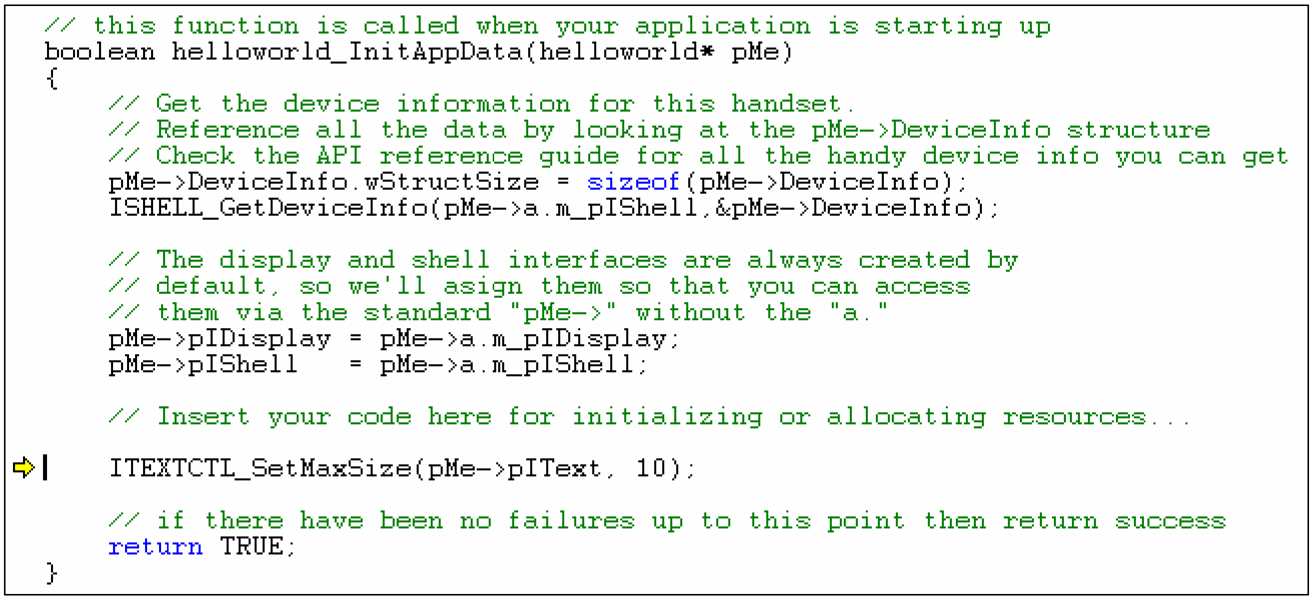


'C:\Program Files\BREW SDK v3.0.1\sdk\bin\BREW_Simulator.exe' does not contain debugging information. Press OK to continue.

[✓] Do not prompt in the future.

Figure 39. Debugging information warning

When you start debugging, the Simulator is invoked. Start your application as normal within the Simulator; control returns to Visual Studio when the Simulator fails, or a breakpoint is reached. Note that this tool greatly facilitates debugging memory access issues, as Visual Studio indicates exactly which line of code is responsible for the failure (see Figure 40. Memory access violation).



```
// this function is called when your application is starting up
boolean helloworld_InitAppData(helloworld* pMe)
{
    // Get the device information for this handset.
    // Reference all the data by looking at the pMe->DeviceInfo structure
    // Check the API reference guide for all the handy device info you can get
    pMe->DeviceInfo.wStructSize = sizeof(pMe->DeviceInfo);
    ISHELL_GetDeviceInfo(pMe->a.m_pIShell,&pMe->DeviceInfo);

    // The display and shell interfaces are always created by
    // default, so we'll asign them so that you can access
    // them via the standard "pMe->" without the "a."
    pMe->pIDisplay = pMe->a.m_pIDisplay;
    pMe->pIShell   = pMe->a.m_pIShell;

    // Insert your code here for initializing or allocating resources...

    ITEXTCTL_SetMaxSize(pMe->pIText, 10);

    // if there have been no failures up to this point then return success
    return TRUE;
}
```

Figure 40. Memory access violation

If breakpoints were specified, control returns to Visual Studio as soon as failure reaches the line of code containing the breakpoint. This is most useful for debugging non-failing code logic issues. To specify a breakpoint, right-click on the desired line of code and select **Insert/Remove Breakpoint** from the resulting pop-up menu (see Figure 41. Adding breakpoints).

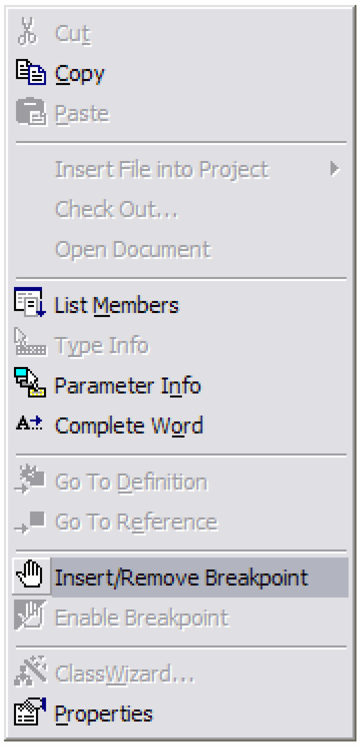
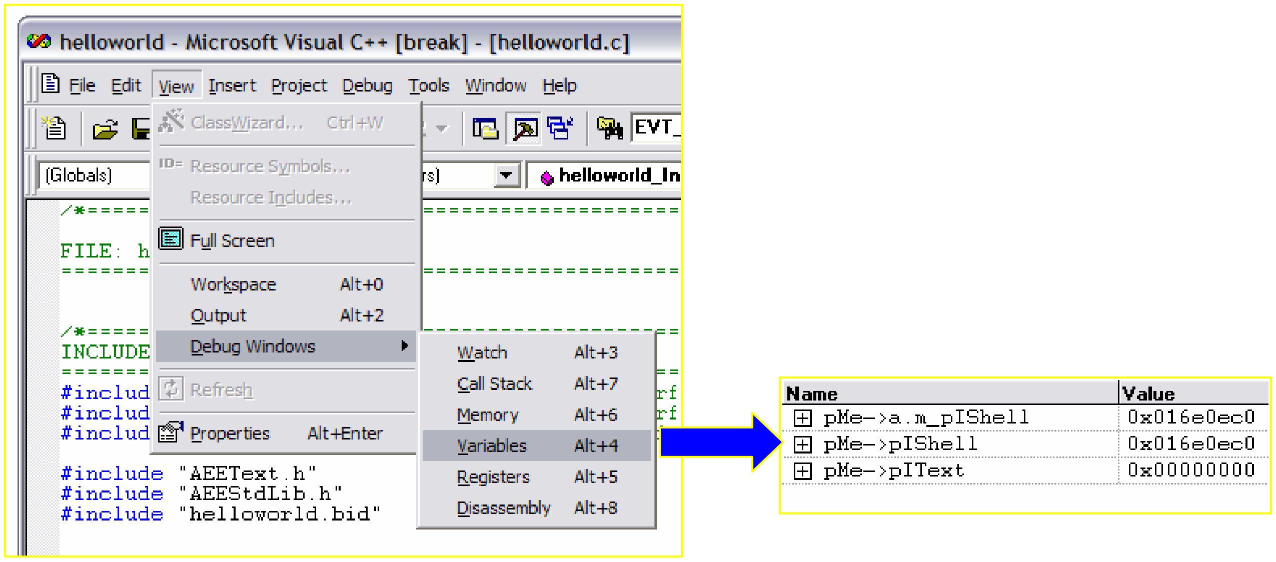


Figure 41. Adding breakpoints

After the breakpoint is reached, use Visual Studio's debugging tools to inspect relevant variables and function call parameters and review your code to diagnose the causes of your program's undesired behavior (see Figure 42. Displaying debug windows). Make sure the necessary debugging windows are open by selecting **View > Debug Windows > Variables, Memory, or something.**).

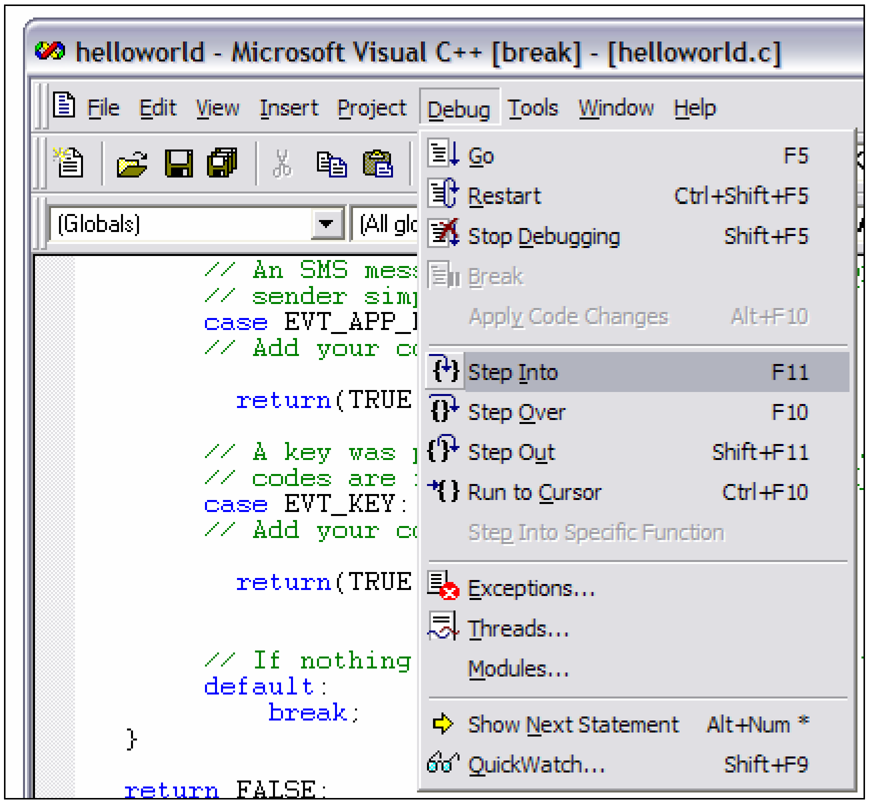


```
#include "AEEText.h"
#include "AEEStdLib.h"
#include "helloworld.bid"
```

Name	Value
⊞ pMe->a.m_pIShell	0x016e0ec0
⊞ pMe->pIShell	0x016e0ec0
⊞ pMe->pIText	0x00000000

Figure 42. Displaying debug windows

After all the necessary information appears, use the Debug menu to review your code and note the behavior of these watched variables (see Figure 43. Advancing program execution).


helloworld - Microsoft Visual C++ [break] - [helloworld.c]
File Edit View Insert Project Debug Tools Window Help
(Globals)
(All glo
Go F5
Restart Ctrl+Shift+F5
Stop Debugging Shift+F5
Break
Apply Code Changes Alt+F10
Step Into F11
Step Over F10
Step Out Shift+F11
Run to Cursor Ctrl+F10
Step Into Specific Function
Exceptions...
Threads...
Modules...
Show Next Statement Alt+Num *
QuickWatch... Shift+F9

```
        // An SMS mess
        // sender simp
        case EVT_APP_I
        // Add your co

          return(TRUE

        // A key was p
        // codes are i
        case EVT_KEY:
        // Add your co

          return(TRUE


        // If nothing
        default:
            break;
    }

    return FALSE;
```

Figure 43. Advancing program execution

Additional resources

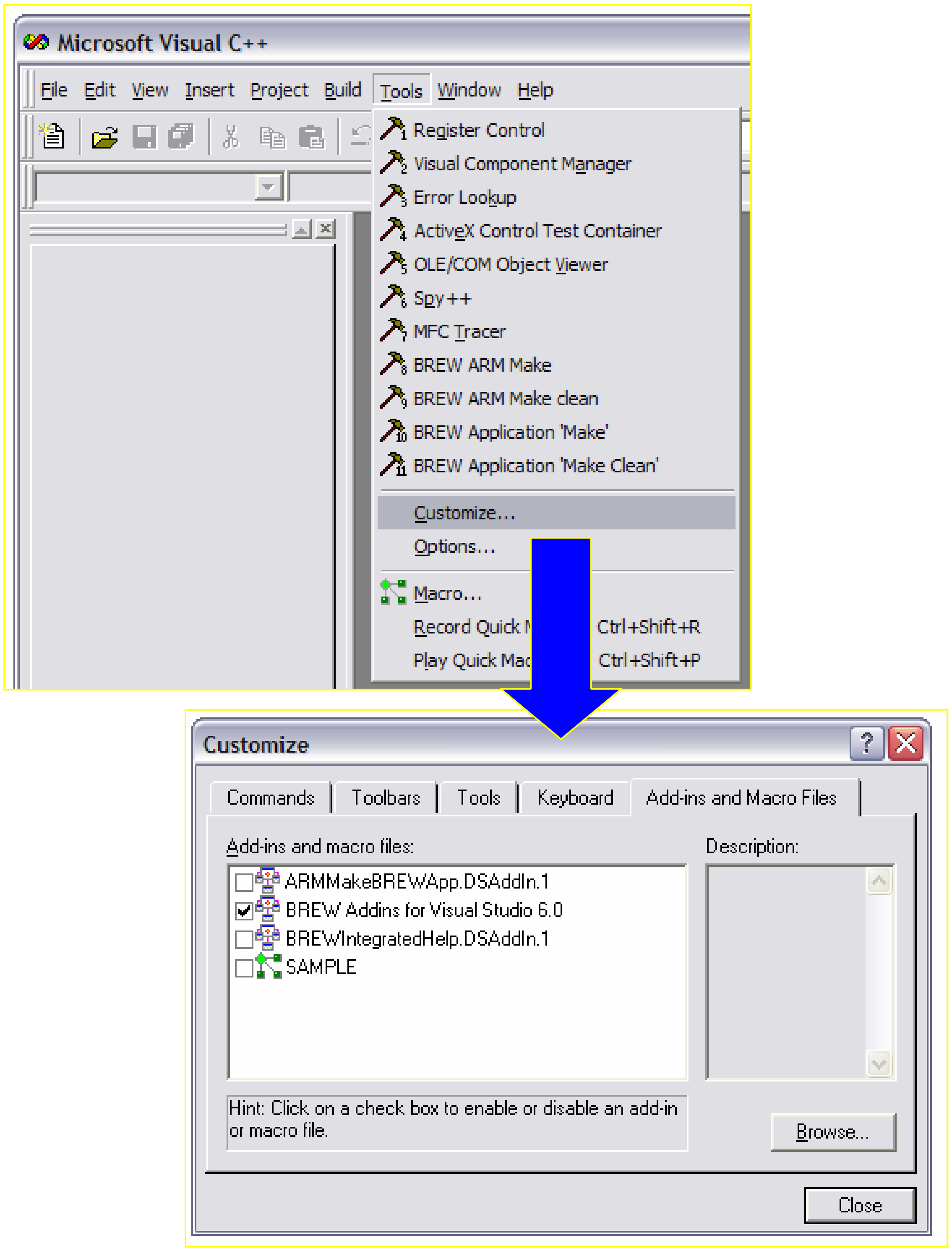
For more information on Microsoft Visual Studio's debugging capabilities, consult the Visual Studio references.

Loading BREW Applications to the Mobile Device

Before you load your application to a mobile device, you must compile it into a MOD file for the ARM processor. The officially supported compilation solution is the ARM Realview Cross Compiler, part of the ARM Developer Suite. For more information on the ARM Realview Cross Compiler and information on obtaining a free 45-day demo version, follow the appropriate link from the BREW Compilation Tools page.

Generating ARM makefiles

Compilation with the Realview Cross Compiler requires an appropriate makefile; the simplest way to generate a suitable makefile is through the automated makefile wizard in the BREW Add-ins for Microsoft Visual Studio. After installing the add-ins, you must enable the BREW Add-Ins toolbar within Visual Studio. To do so, select **Customize** from the Tools window, and in the Customize dialog check the box next to the BREW Addins listing on the Add-ins and Macro Files tab (see Figure 44. Enabling the BREW Add-ins toolbar).



Add-ins and macro files:

- [ ] ARMMakeBREWApp.DSAddIn.1
- [✓] BREW Addins for Visual Studio 6.0
- [ ] BREWIntegratedHelp.DSAddIn.1
- [ ] SAMPLE

Hint: Click on a check box to enable or disable an add-in or macro file.

Figure 44. Enabling the BREW Add-ins toolbar

The toolbar (see Figure 45. BREW Add-ins toolbar) has buttons for ARM and GNU makefile generation, as well as quick links to the BREW MIF Editor, BREW Resource Editor, BREW Simulator, and BREW Online Help. Clicking the appropriate makefile generation button creates a valid makefile within your application directory.

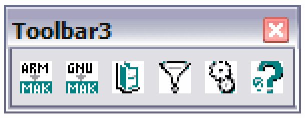


Figure 45. BREW Add-ins toolbar

Compiling for ARM

After creating a makefile for your application, open a command prompt window and navigate to your application directory. At the prompt, type:

```
nmake /f <makefile name> all
```

This invokes the nmake utility with your makefile, compiling the application for ARM.

NOTE: If you are trying to compile the sample applications provided in the SDK with the supplied makefiles, verify that you have the proper version of the ARM Developer Suite installed to meet the makefile prerequisites. Running a makefile designed for a different version of the ARM Developer Suite may result in an unusable compilation.

NOTE: Applications targeted for big-endian (Nokia) handsets will need to specify the big-endian compilation option. To do so, modify the appropriate line within the compiler code generation options section, assuming a makefile generated using BREW Add-Ins, to read:

```
END = -bigend
```

GCC compilation

For developers on a tight budget, GCC may be used to compile BREW applications. The appropriate support files are available on the BREW Compilation Tools page. GCC makefiles may be generated with the BREW Add-Ins for Visual Studio following the procedure outlined above. Note that QUALCOMM does not provide assistance for GCC-specific issues; the GCC support files are provided merely for your convenience.

Generating test signatures

A test signature is necessary to start your application on a handset. Authenticated developers generate test signatures on the BREW Generators page.

About the BREW Tools Suite

The BREW Tools Suite provides a collection of tools to facilitate loading and testing your application on a mobile device. To download the BREW Tools Suite, you must be an authenticated developer.

Contents

BREW AppLoader	The BREW AppLoader provides an interface for accessing the file system of your mobile device, which allows a developer to remove applications, modify directories, and copy applications to the handset for testing.
BREW AppSigner	This tool is used to digitally sign an application, which is necessary for submission.
BREW Logger	Logger connects to a device and displays debugging information printed with DBGPRINTF().
The Grinder	The Grinder is an enhancement to the Simulator that allows for the automatic generation of testing events (for example, key entry). Events may be randomly generated or specified through the use of Perl scripting.

Connecting to a device

When you run AppLoader or Logger, you are prompted with a connection window. Within this window, specify the DLL used to communicate with the mobile device (default is QCOMOEM.dll) and the port used to communicate with the phone (see Figure 46. BREW Tools connection window). Select the proper port setting and click **OK** to connect with the device.

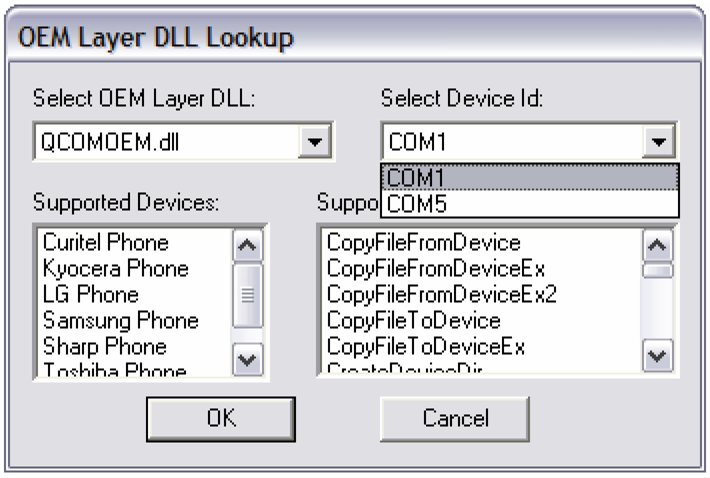


Figure 46. BREW Tools connection window

BREW AppLoader

After connecting to the handset, the AppLoader shows the deivce's BREW file system (see Figure 47. BREW file system), including information about the free memory available.

NOTE: The AppLoader cannot rename files directly on the handset or copy certain file types to your PC for security reasons.

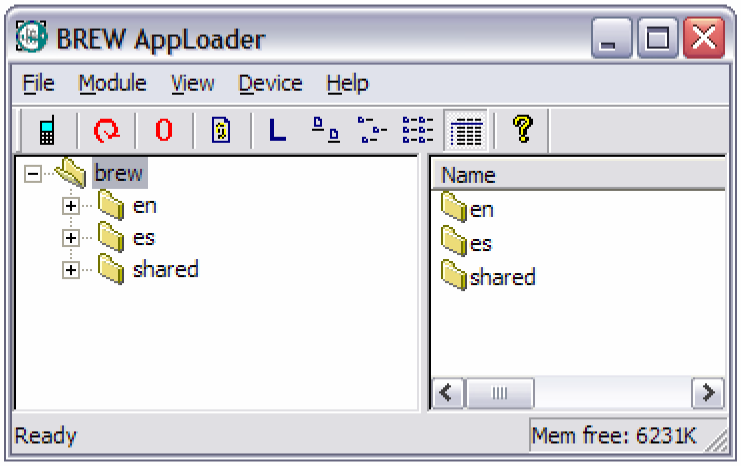
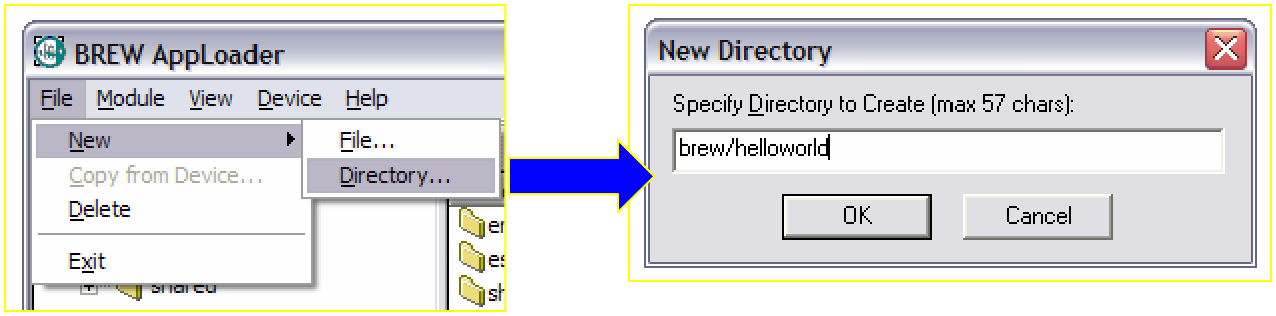


Figure 47. BREW file system

To load an application to a handset, you will need to create a directory with your application's name under the brew root folder (see Figure 48. Creating the application directory).



New Directory

Specify Directory to Create (max 57 chars): brew/helloworld

Figure 48. Creating the application directory

Within this directory, copy all files used by your application and the test signature file (files may be copied by dragging from a Windows file explorer into the AppLoader window) (see Figure 49. Application directory contents). Copy your application's MIF to the brew directory (see Figure 50. MIF location).

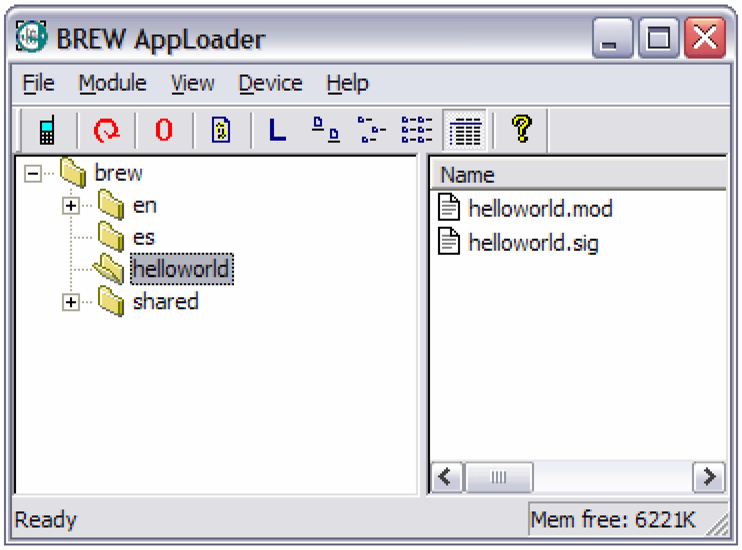


Figure 49. Application directory contents

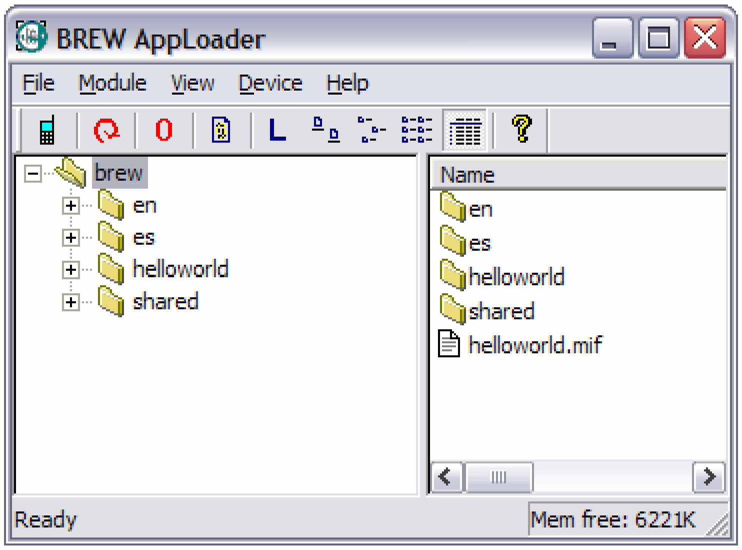


Figure 50. MIF location

The device must be power-cycled before you can run your application; select **Device > Reset** from the BREW AppLoader menu to trigger a device reset (see Figure 51. Resetting the device). After resetting your device, start your application within the BREW environment.

NOTE: You must have a valid test signature in your application directory and a test-enabled handset, or your application is deleted when you power-cycle the phone.

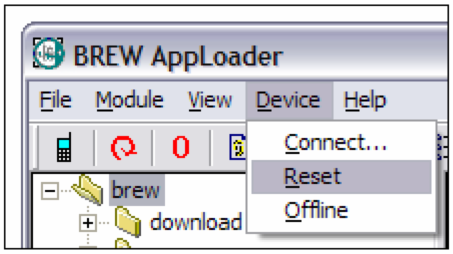


Figure 51. Resetting the device

BREW AppSigner

The BREW AppSigner is used to digitally sign application packages prior to submitting completed applications for True BREW testing; Its functionality is beyond the scope of this guide. For information on using the BREW AppSigner, consult the *BREW AppSigner Help* guide, included in the BREW Tools Suite installation.

NOTE: The SIG file created by AppSigner is not a handset signature file and should not be uploaded to the handset.

BREW Logger

After you connect to the handset through the standard BREW Tools connection window, select **Connect** from the File menu (see Figure 52. Connecting to the BREW device). To begin logging, select **Start Logging** from the Configure menu (see Figure 53. Starting logging operations); the BREW Logger Log Output window displays all log messages.This process displays information printed using DBGPRINTF() while running on the BREW device, facilitating the debugging process. Certain features of the BREW Logger are scriptable with Perl; see the *BREW Logger Guide* for more information.

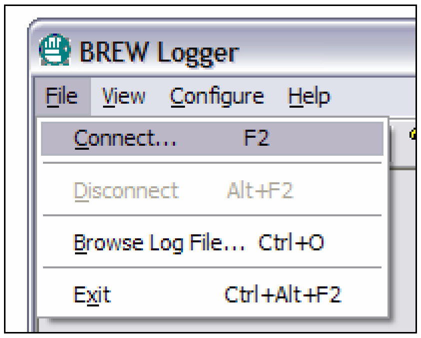


Figure 52. Connecting to the BREW device

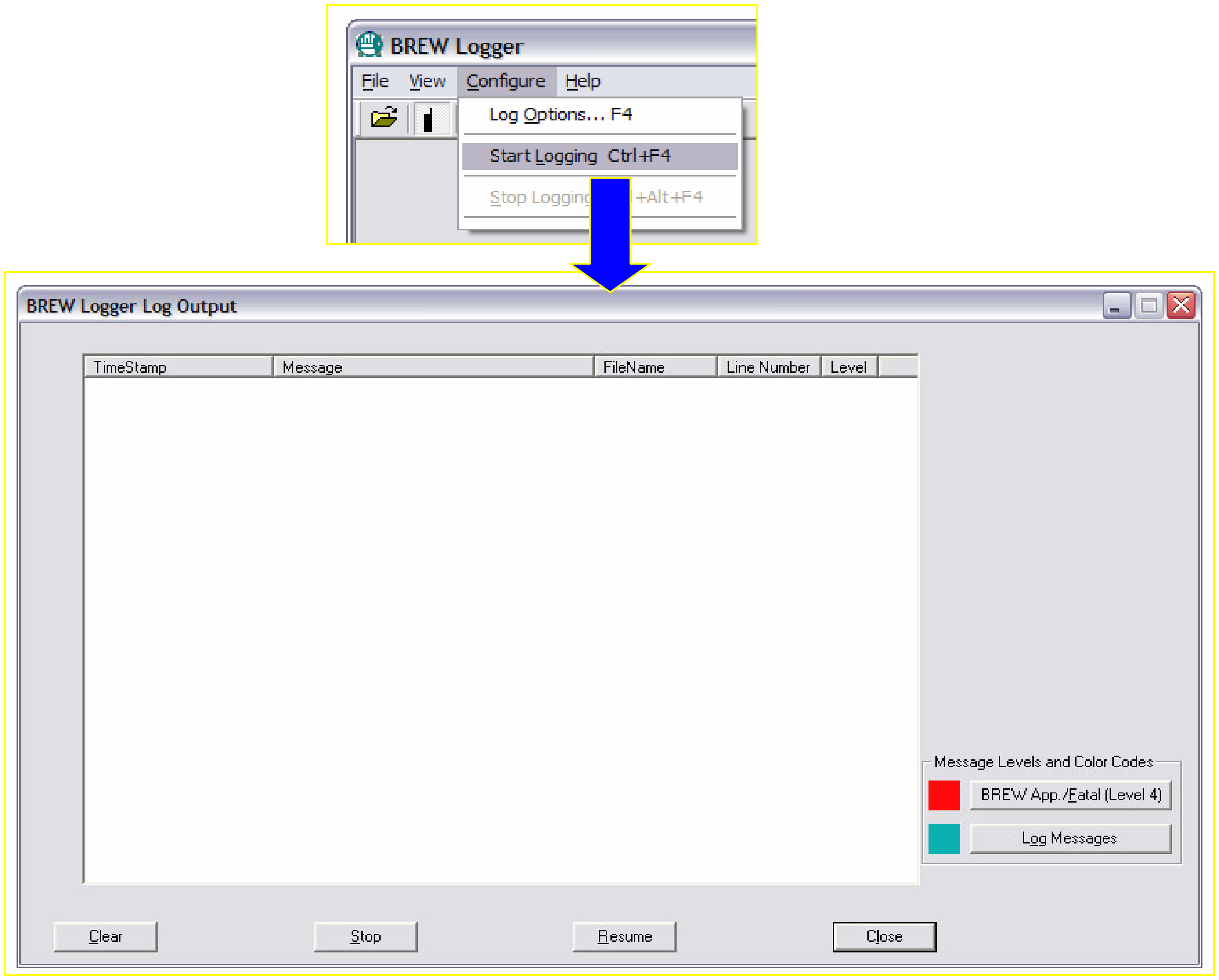


Figure 53. Starting logging operations

The Grinder

The Grinder may be used for the automated testing of your application and is beyond the scope of this guide. For information on using The Grinder, consult *The Grinder Help* guide, included in the BREW Tools Suite installation.

Common issues

The BREW Tools Suite may sometimes encounter difficulties while attempting to connect to a device. Following these steps will resolve most conflicts and connection issues:

- Check that all cables and devices are securely connected.
- Ensure that you are using a data cable that is compatible with your handset.
- Try power-cycling the mobile device.
- Terminate all applications outside of the BREW Tools Suite that have access to the device or COM port. Sometimes these applications may be running in the system tray or invisible, so you might need to uninstall the offending software.

Using BREW Tools Suite 2.1.2+ for non-3.0 BREW devices

By default, the BREW Tools Suite 2.1.2 expects a BREW 3.0 device. If you are connecting to a phone running an earlier version of BREW, you need to edit the <BREW TOOLS DIRECTORY>\OEMLayer\QCOMOEM.INI file to add support. When you open this file, you see this line:

```
[BREW Version Information]
Version=3
```

To connect to a device running an earlier version of BREW, you must modify the Version field accordingly. For example:

```
[BREW Version Information]
Version=2
```

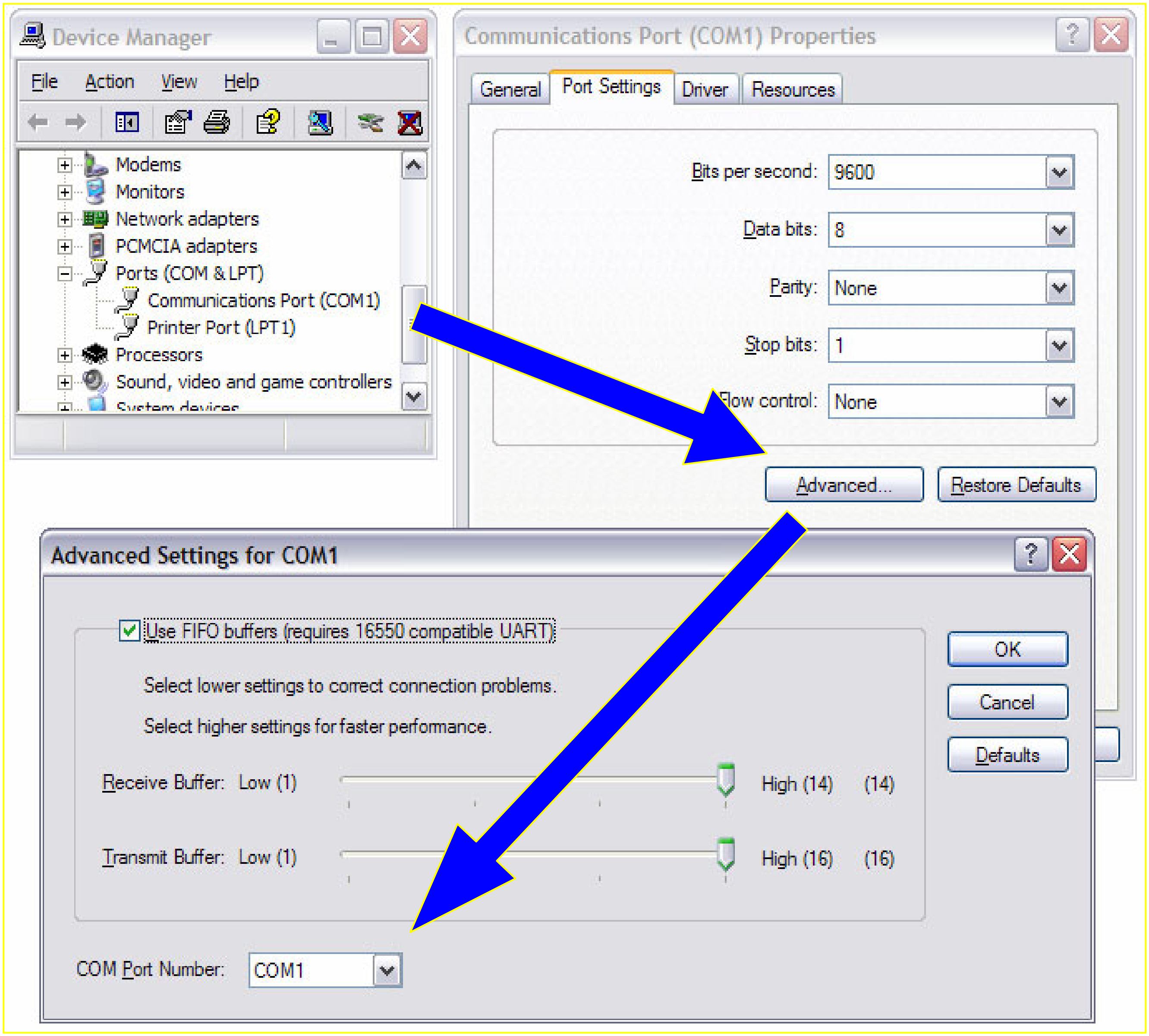
would allow BREW Tools Suite to connect to a BREW 2.x device.

Connecting with USB cables

USB data cables require special drivers, typically provided by the cable manufacturer, to mimic the behavior of a serial cable. Be sure that you have a recent set of drivers for the cable installed on your computer. Contact the cable manufacturer or vendor for more information about obtaining these drivers.

Setting COM port

If your COM port is not available for selection on the connection dialog, you may need to change the COM port assignment in Windows. To change the port assignment, open the Device Manager and right-click on **My Computer > Properties > Hardware > Device Manager**, Change the COM port number for the desired port by selecting the desired **COM port > Port Settings > Advanced > COM Port Number** (see Figure 54. Changing COM settings).


Device Manager
File Action View Help
Modems
Monitors
Network adapters
PCMCIA adapters
Ports (COM & LPT)
Communications Port (COM1)
Printer Port (LPT1)
Processors
Sound, video and game controllers
System devices
Communications Port (COM1) Properties
General Port Settings Driver Resources
Advanced... Restore Defaults
Advanced Settings for COM1
OK
Cancel
Defaults

Bits per second: 9600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

[✓] Use FIFO buffers (requires 16550 compatible UART)

Select lower settings to correct connection problems.

Select higher settings for faster performance.

Receive Buffer: Low (1) High (14) (14)

Transmit Buffer: Low (1) High (16) (16)

COM Port Number: COM1

Figure 54. Changing COM settings

Non-MSM chipsets

Connecting to a non-MSM based handset (for example, Nokia phones) requires special procedures and may require a big-endian compilation. See the Loading BREW Applications to the Mobile Device section for more information on big-endian/little-endian compilation. Nokia phones require a hardware dongle for hardware connection, which is separate from the data cable. You will also need to download a special DLL package (available under the appropriate device information page), and follow the instructions to install the files. Once the dongle has been connected, launch AppLoader and select the appropriate OEM Layer DLL (see Figure 55. Selecting an OEM layer DLL for non-MSM devices). The AppLoader should now be able to connect to the device.

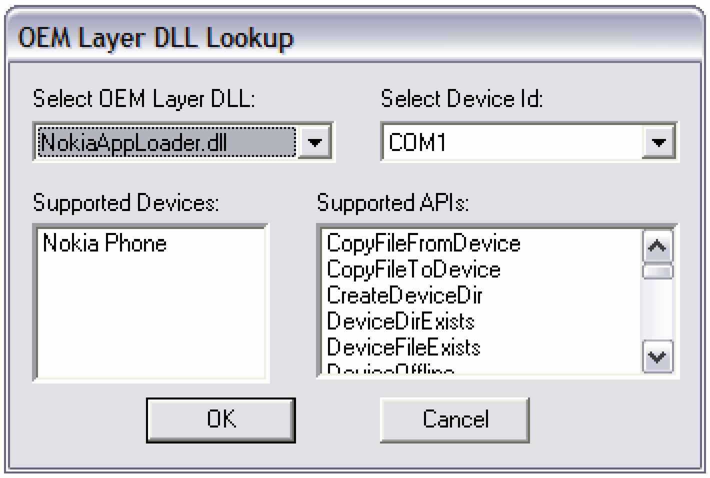


Figure 55. Selecting an OEM layer DLL for non-MSM devices

Killing OEMServerLayer.exe

If you have previously been able to connect to your device but are now unable to do so (usually occurs after abnormal exits of BREW Tools applications), you may need to manually terminate the OEMServerLayer.exe process. Open the Windows Task Manager by pressing **Ctrl+Alt+Del** and selecting **Task Manager**. Terminate the OEMServerLayer.exe process (see Figure 56. Terminating the OEM layer server process). After this process has been terminated, try connecting to your handset.

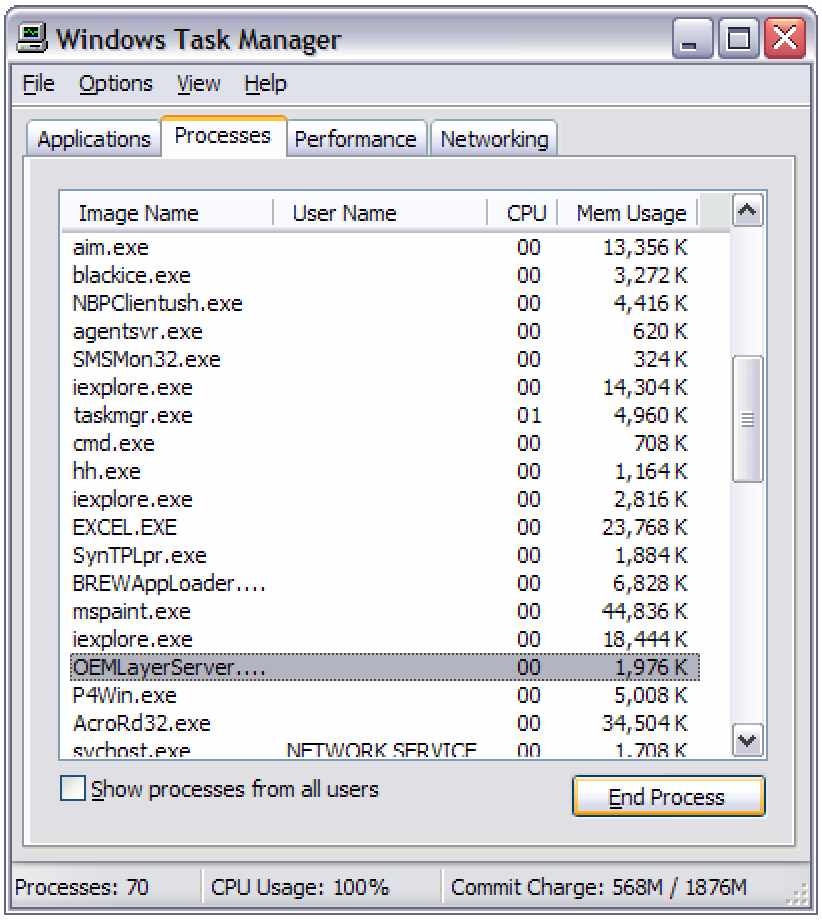

Windows Task Manager
File Options View Help
Applications Processes Performance Networking
Show processes from all users
End Process

Image Name	User Name	CPU	Mem Usage
aim.exe		00	13,356 K
blackice.exe		00	3,272 K
NBPClientush.exe		00	4,416 K
agentsvr.exe		00	620 K
SMSMon32.exe		00	324 K
iexplore.exe		00	14,304 K
taskmgr.exe		01	4,960 K
cmd.exe		00	708 K
hh.exe		00	1,164 K
iexplore.exe		00	2,816 K
EXCEL.EXE		00	23,768 K
SynTPLpr.exe		00	1,884 K
BREWAppLoader....		00	6,828 K
mspaint.exe		00	44,836 K
iexplore.exe		00	18,444 K
OEMLayerServer....		00	1,976 K
P4Win.exe		00	5,008 K
AcroRd32.exe		00	34,504 K
svchost.exe	NETWORK SERVICE	00	1,708 K

Processes: 70 CPU Usage: 100% Commit Charge: 568M / 1876M

Figure 56. Terminating the OEM layer server process

Additional resources

For more information on the BREW Tools Suite components, read the documentation included with the installation.

Running BREW Applications on the Mobile Device

After loading your application to the BREW handset (see About the BREW Tools Suite section above for information on this procedure), start the application through the BREW Application Manager interface on your handset. Your carrier may have given the BREW Application Manager a proprietary name; for example, Get It Now on Verizon, U-Magic on China Unicom, BellSouth Interactivo on BellSouth. The procedure to invoke the BREW Application Manager varies between handset models and carriers; check your handset manual if you have questions on the proper sequence. Due to the characteristics of the mobile environment, your application may not work as anticipated. This section lists the major causes of erroneous application behavior on a mobile device and provides guidance on resolving these issues.

Watchdog timer

The BREW environment limits the amount of time any process may run without yielding through a watchdog timer. If your application trips this watchdog timer, it terminates, and the handset may power-cycle. Many factors can cause your application to trigger the watchdog timer, though the most common culprit is improper looping within your application. BREW uses an event-driven application model, so a program should never implement busy-waiting or perform loops that are likely to monopolize the processor for extended periods.

Memory alignment

Improperly aligned memory accesses on the ARM processor cause your application to fail. In the ARM environment, 16-bit quantities must be at even addresses, and 32-bit quantities must be at addresses divisible by 4. Members of structures must be properly aligned as well; the ARM compiler automatically pads structures to ensure that all members are correctly aligned. If you use a PACKED structure, the structure members are arranged with single-byte memory alignment, and you must take special care in accessing the members to avoid misaligned memory access. Generally the best way to conserve space in a structure is to declare the members in order of decreasing size; this minimizes the amount of padding inserted by the compiler while retaining proper member alignment.

Heap usage

Applications running in BREW may fail if proper precautions are not taken to verify that heap memory allocations complete successfully. BREW applications should always inspect the return value of any dynamic memory operations to ensure that the result is valid, and the application should exit gracefully after detecting an error.

Stack usage

Care should be taken to minimize the stack usage on the BREW device. Most devices have a relatively small stack space, so overrunning the stack is very possible if proper programming techniques are not used.

- Avoid deep recursive chains. Each recursive call has the overhead of an additional stack frame, which can quickly fill the stack.
- Minimize the use of automatic variables. If possible, shift local buffers and automatic variables used across multiple methods to the applet structure. This relocates the allocated memory from the stack to the heap. Use pointers to pass structures to methods; this prevents the structure from being copied on the resulting stack frame.

DBGPRINTF()/File logging

If you connect to your device using the BREW Logger (refer to About the BREW Tools Suite for more information), you can inspect the information printed using the DBGPRINTF() function. Unfortunately, some devices do not properly support DBGPRINTF(). To work around this limitation, you need to log to a file. The function below is used to perform simple file logging for a specified input string.

```
#include "AEEFile.h"
#define LOGFILE "log.txt"

void LogToFile(const char * const buf)
{
 AEEApplet * app = (AEEApplet *)GET_APP_INSTANCE();
 IShell * pIShell = app->m_pIShell;
 IFileMgr * pIFileMgr;
 if (ISHELL_CreateInstance(pIShell, AEECLSID_FILEMGR, (void **)&pIFileMgr)
== SUCCESS)
 {
  IFile * pIFile;

  if (IFILEMGR_Test(pIFileMgr, LOGFILE) != SUCCESS)
   pIFile = IFILEMGR_OpenFile(pIFileMgr, LOGFILE, _OFM_CREATE);
```

```
    else
     pIFile = IFILEMGR_OpenFile(pIFileMgr, LOGFILE, _OFM_APPEND);

    if (pIFile != NULL)
    {
     JulianType jt;
     char timestamp[30];

     GETJULIANDATE(GETTIMESECONDS(), &jt);

     SPRINTF(timestamp, "[%02d:%02d:%02d] ", jt.wHour, jt.wMinute,
jt.wSecond);
     IFILE_Write(pIFile, timestamp, STRLEN(timestamp));

     IFILE_Write(pIFile, buf, STRLEN(buf));
     IFILE_Write(pIFile, "\r\n", 2);
     IFILE_Release(pIFile);
    }

    IFILEMGR_Release(pIFileMgr);
  }
}
```

Common issues

Application removed when handset power-cycled

If your application has been removed from the phone after power-cycling, the most likely cause is that your phone has not been test-enabled. To test-enable your phone, it must be sent to the QUALCOMM phone center.

Signature failures

Any errors in your test signature result in a digital signature failure when you launch your application. Some of the common causes for signature failures are shown below.

1026 (Signature file missing)	No signature file was found; ensure that a signature file is generated and placed within the application directory.
1028 (Expired signature)	Try generating a new signature file, the lifetime on the previous signature may have expired. Also verify that your handset has the proper date and time set.
1030 (Invalid ESN)	Check that your signature was generated using the proper ESN. If an error was made during signature generation, you must generate a new signature with the correct ESN.

Interpreting error codes

When an application returns a specific error code to indicate the cause of the failure, you must refer to the AEEError.h file to figure out the meaning for a given value.

NOTE: Error values in the AEEError.h file are given in the form of a decimal offset number added to a hexadecimal base error code. To find the actual error code value, you may need to perform a conversion between decimal and hexadecimal before performing the addition. For example, AEE_NET_ECONNREFUSED is defined as NET_ERROR_BASE+17, where NET_ERROR_BASE is 0x200. In this case, the actual error code would be 0x211 = 529, NOT 0x217 = 535.

Unexpected behavior on a particular device

Some devices have known issues that cause undesirable behavior in BREW. For a listing of known issues and possible workarounds, consult the proper Device Data Sheet on the BREW Devices page.

Where to go for help

Many of the issues that are encountered while programming in BREW are discussed in the documentation included in the relevant installation package. Be sure to familiarize yourself with the guides for general BREW questions; the search feature within the guide is also used to locate information relevant to specific issues. If you are unable to find a solution in the documentation, check the developer section of the QUALCOMM BREW website. QUALCOMM maintains an extensive list of FAQs and technical assistance articles to aid in resolving BREW problems. Further assistance may also be found in the BREW Developer Forums, where developers from around the world discuss a wide variety of issues related to the BREW environment.