Developer's Guide

RIM 950 Wireless Handheld™

Version 1.7

SDK User's Guide

RIM 950 Wireless Handheld — SDK User's Guide, Version 1.7
Last revised 9/18/00

Part Number: PDF-03037-001

Complies with software O/S version 2.0 and applications version 1.7.

Research In Motion Limited
295 Phillip Street
Waterloo, Ontario
Canada N2L 3W8
Tel. (519) 888-7465
Fax (519) 888-6906
Web site: www.rim.net
Email: info@rim.net

Printed In Canada JHM0700/sdk_uguide.doc

Contents

1

Introduction

The RIM 950 Wireless Handheld Software Developer's Kit (SDK) includes all the tools needed to begin application development quickly. The SDK provides an extremely powerful development environment that utilizes Microsoft Developer Studio 5.0 or later (Visual C++ 5.0 or later), supporting Windows 95 and Windows NT.

The RIM Wireless Handheld™ features a 32-bit Intel386™ processor which executes the same instruction set as a Windows 95 computer. The operating system provided with the wireless handheld is a multitasking operating system with built-in messaging. You can write new components for the wireless handheld without needing to rewrite entire applications.

About the SDK

The SDK package includes:

- software
- application utilities
- PC-based simulator (herein referred to as the "RIM Wireless Handheld simulator")

If you have used a previous version of this SDK, you should read the release notes section of this document (see page 15). The release notes section describes changes required to make old applications run under this version of the software. For example, it describes the new versioning scheme and outlines compatibility requirements in new releases.

About the network

The RIM Wireless Handheld operates over the BellSouth Intelligent Wireless Network℠ in the United States and the Rogers™ AT&T® wireless data network in Canada. Both networks offer broad coverage, nationwide roaming, fast messaging, and reliable delivery. Even if your RIM Wireless Handheld is turned off or temporarily out of coverage, your messages will be stored and forwarded automatically when you return to coverage.

About Research In Motion

Research In Motion (RIM) manufactures high performance radio modems for use with the Mobitex network. These modems and other Mobitex products manufactured by RIM are used on Mobitex networks around the world.

RIM has an international reputation for developing high performance RF technology in the narrowband PCS data communications industry. RIM's wireless technology has set new standards for battery life, transceiver performance, physical size, ease-of-use, and price.

By helping companies design a broad range of wireless data products, RIM has a detailed understanding of what end users and equipment manufacturers need to use the Mobitex network effectively and affordably.

The result of this close co-operation is a family of advanced wireless data products that are easy to use and integrate quickly, including the RIM 902M™ OEM radio modem and the RIM 950 Wireless Handheld.

About this guide

These Developer's Guides are meant to assist you in creating your applications. This guide includes general information on the SDK itself.

Throughout this guide, the simulator will be referred to as the 'RIM Wireless Handheld simulator.'

Some information is set apart typographically in the following ways:

Note Notes provide additional information to help complete a task.

Tip Tips offer an alternative method of performing an action

WARNING Warnings follow any procedure or paragraph containing instructions that, if followed improperly, could result in damaging the RIM Wireless Handheld or software.

The names of program groups, icons, folders, windows, etc. will appear in **bold** text. An example would be the **Research In Motion** program group.

References to buttons have the hot key underlined where applicable. An example would be the Next> button.

All programming elements such as returns, events, constants, structures, parameters, etc. will appear in `Courier New` font. An example would be the `COMM_RX_ERROR` error code.

2

Installing the SDK

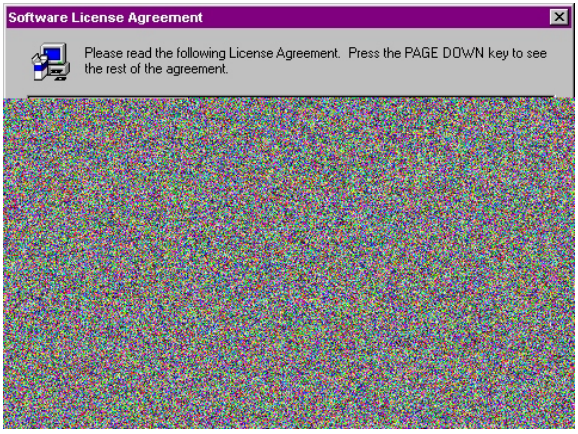
In order to run the RIM 950 Wireless Handheld Software Developer's Kit, you need a PC capable of running Microsoft Developer Studio 5.0 (or later). This chapter describes the process of installing the RIM 950 Wireless Handheld SDK and configuring Microsoft Developer Studio 5.0 (or later) to work with the SDK.

You can install and run the RIM 950 Wireless Handheld SDK in both Windows 95 and Windows NT.

To install the SDK

1. Double-click on the installer icon.

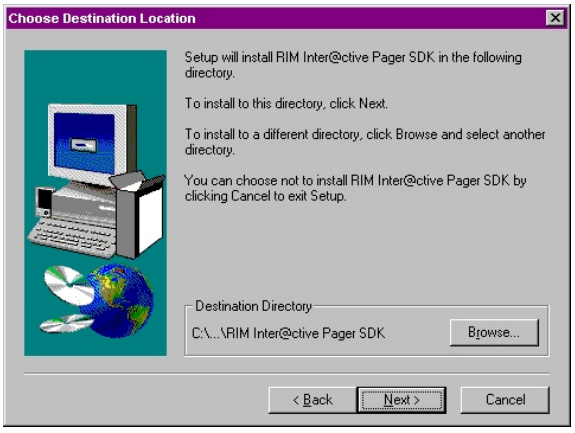
2. The RIM Wireless Handheld simulator setup desktop will appear. Follow the instructions on your screen throughout the setup process.



Software License Agreement window

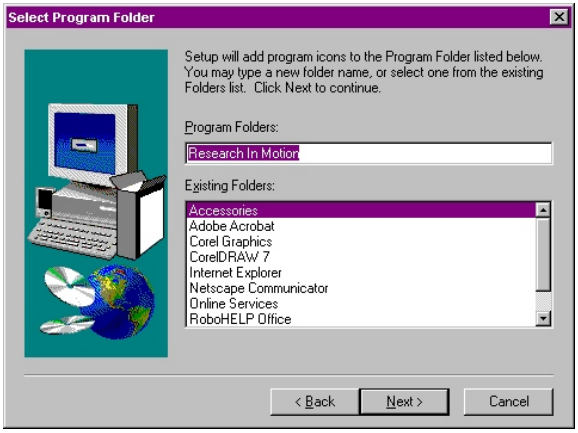
Note Please read the license agreement carefully. Proceeding with the installation indicates that you agree with the conditions in the license agreement.

3. When the **Choose Destination Location** window opens, a default directory will be displayed. To change this directory, click the Browse button, and the **Choose Directory** window will open. Select the path, drive, and directory that you would like for your destination. Click OK to return to the **Choose Destination Location** window.



Choose Destination Location window

4. Once you have selected the destination directory, click Next> to continue. The **Select Program Folder** window opens.



Select Program Folder window

5. Type in a folder name in which the setup items will be stored. The default folder name is **Research In Motion**. Alternatively, you may select a folder from the **Existing Folders** section.

6. At the end of the installation, an **Installation Query** window will open to ask if you would like to view the README file.

7. A new window will open to display the **Research In Motion** program group.

The RIM 950 Wireless Handheld SDK contains all of the tools and information needed to build an application.

See page 55 for information on the process of building an application for the RIM Wireless Handheld and the structure of an application. See Chapter 3 – Tools Guide on page 19 for information on the tools used to create a wireless handheld application and for a description of their function and usage.

To configure Microsoft Developer Studio

The RIM 950 Wireless Handheld SDK is designed to make use of the libraries and features in Microsoft Developer Studio 5.0 (or later). Handheld applications are built as a Windows DLL target, but they are not used as Windows applications. The final DLL is stripped of extraneous information and then ported into the RIM Wireless Handheld operating system.

Because Windows sees the RIM Wireless Handheld applications as DLLs, Developer Studio allows you to use nearly all development facilities available for Windows, including using the Integrated Development Environment, source level debugging, and breakpoints. However, DLLs must not make calls to the Windows API, even when running under Windows.

You must be familiar with Microsoft Developer Studio to develop applications. You must purchase and become familiar with this application separately from the RIM 950 Wireless Handheld Software Developer's Kit.

Follow these steps to configure the Microsoft Visual C++ 5.0 (or later) environment for building RIM 950 Wireless Handheld applications.

1. Start the Microsoft Developer Studio and select File > New > Project to create a new project.
    - Give the project a name.
    - Define the project type as `WIN32.DLL`.

      Although the target is a Windows DLL, the RIM Wireless Handheld application is not compatible with the Windows operating system.
2. Select Project > Settings to set up the project parameters.
    - Go to the Settings For: tab.
    - Select **All Configurations**.
3. Go to the Debug tab.
    - Set the executable for the Debug Session as `OSLOADER.EXE`.
    - In the Program arguments field put a line in the form:

      `[OS DLL][OS parameters][application DLLs]`

      `[OS DLL]` should be `OSPGRMB.DLL` if you are simulating a RIM 950 and `OSHHMB.DLL` if you are simulating a RIM 957™. (If you're using `OSLOADER.EXE`, you don't need to specify a complete path because the program looks first in its current directory.) `[OS parameters]` are any command line switches you need to pass to the simulator.
    - Finally, list all of the application DLLs at the end of the line, making sure your application is the last one listed.

      In order for the `OSLOADER.EXE` to find the DLLS, you must either place the DLLs in your `PATH` or place them in the same directory and use that directory as your working directory.
4. Go to the C/C++ tab.
    - Set the Category to **Preprocessor**.
    - Enter the paths for the SDK include files in the Additional Include Directories field. These paths are to the

subdirectories \include and \include\internal of the RIM 950 Wireless Handheld SDK installation directory.

5. Change the Category to **Code Generation**.
    - Set the Struct Member Alignment to 2 bytes. This will minimize memory usage.
    - Set the Processor to 80386.
6. Change the Category to **C/C++ Language**.
    - Make sure that *all* check boxes are unchecked.
    - The Enable Exception Handling checkbox should *always* be unchecked.
7. Switch to the Link tab.
    - Set the Category to **General**.
    - At the end of the Object/Library Modules field, include the files `RIMOS.LIB`, `OSENTRY.OBJ`, at the beginning, followed by any `.LIB` files that are required (e.g. `UI32.LIB`, `ADDRESS.LIB`, etc.). Place `LIBC.LIB` at the end.
    - Check the checkbox Ignore all default libraries. Set the Category to **Input**.
    - Place the path to the subdirectory `\lib` of the RIM 950 Wireless Handheld SDK installation directory in the Additional library path.
8. Save these settings.
9. Your setup is complete. Now you are ready to create your application.

3

Release notes

This section describes some of the changes between versions 1.6 and 1.7 of the RIM 950 Wireless Handheld SDK. It describes changes you must make to your existing code base before attempting to build applications for RIM 950 Wireless Handheld 1.7.

WARNING The 1.7 environment is *not* binary compatible with 1.6. You *must* recompile your code before running it on a 1.7 system.

RIM 950 Wireless Handheld 1.7 comes with a new operating system, a new simulation environment and some new methods and classes. This section lists the methods and functions that have changed for 1.7. You should pay close attention to this list and modify your code if you used the old forms.

Recompiling your applications

There is a new OS library, named `RIMOS.LIB`. Change the linker settings for your development environment and replace the old OS library (`PAGER950.LIB`) with `RIMOS.LIB`.

There are also a number of new methods and classes for 1.7. Check the sections below for any code changes you might require. Be sure to modify your code appropriately to match the new method signatures.

Message

The `create_message()` call has been modified. It was:

```
bool create_message (NetworkMessage * message = NULL)
```

It is now:

```
bool create_message ( void )
```

See the Message API Developer's Guide for more information.

OS

Version 2.0 of the OS removes certain constants. The constants LCD_WIDTH, LCD_HEIGHT, LCD_HEIGHT_BYTE, LCD_DISPLAY_SIZE, and LCD_DISPLAY_SIZE_BYTE that were available but deprecated in previous releases of the SDK are no longer available. The information they provided should be obtained using the LcdGetConfig call (from <LCD_API.H>). For quick recompiling, the header PRE20.H can be included, which defines those values as function calls. This solution should be avoided, however, since the overhead of the functions makes the use of the values very inefficient.

New simulator

For RIM 950 Wireless Handheld 1.7, a new simulator replaces the existing PAGER950 executable. To debug applications using the new simulator, replace PAGER950.EXE in the Executable for debug session field of the Debug tab in the project settings with OSLOADER.EXE. Use an OS DLL (for example, OSPGRMB.DLL) to the Program arguments field as the first argument. For more information on how to use the new simulator, see the section titled "Device Simulator" in this guide.

Ribbon changes

The call RibbonSetApplicationString() now has an extra parameter. If your application calls this function, you will get a link error and will need to make a small code change.

The previous version of the call in the 1.5 SDK was defined in RIBBON.H as:

```
void RibbonSetApplicationString(
   const char * const appName,
   const char * const string,
   int priority );
```

The new version of the call is defined as:

```
void RibbonSetApplicationString(
   const char * const appName,
   const char * const string,
   int priority,
   const BitMap * const bitMapPtr );
```

The `bitMapPtr` parameter is new, and refers to an optional image that can be displayed instead of one of the characters of `string`. The `string` has a normal maximum of five characters; if `bitMapPtr` is not NULL, the maximum is four characters, because one character position is used by the bitmap.

Address Book changes

Two functions have been added to notify the application of changes to the Address Book's data: `register_address_book_update()` registers an application, and `get_address_book_change_count()` returns the number of changes since the last reset.

The Address Book API now has a set of functions for dealing with group addresses. A group address is an address name that represents a list of other addresses. The new group addressing functions provide the following features for dealing with group addresses:

- Determine if an address is a group address
- Get number of members
- Get information about members
- Get UIN from address handle

4

Tools guide

This section provides information on how to make use of the tools provided in the SDK. These tools are provided:

- RIM Wireless Handheld simulator program (`SIMULATOR.EXE`/`OSLOADER.EXE`), used to test programs on a PC without loading them onto a wireless handheld
- Program Loader program (`PROGRAMMER.EXE`), used to load programs onto the RIM Wireless Handheld
- Conversion utilities for creating fonts and bitmaps (`BMP2DEF.EXE`, `BITMAP.EXE`, and `LCDFONT.EXE`)
- A utility (`DLLUTIL.EXE`) to display information about a RIM Wireless Handheld application

This chapter describes configuration and use for each of these tools. We assume you have a basic familiarity with Microsoft Developer Studio.

RIM 950 Wireless Handheld simulator

The simulator allows you to test applications on the RIM Wireless Handheld operating system without having to download the DLLs to the wireless handheld itself. The simulator supports the full functionality of the RIM Wireless Handheld itself, and can even be connected to a RAP modem via the PC's serial port, allowing the radio API to communicate with the live Mobitex network.

Note There are some differences between the RIM Wireless Handheld device and the simulator. The simulator responds to certain debugging calls, sending the output to the Visual Studio output pane; those calls have no effect on the RIM Wireless Handheld.

Running the simulator

There are two RIM Wireless Handheld simulator programs: `SIMULATOR.EXE` AND `OSLOADER.EXE`.

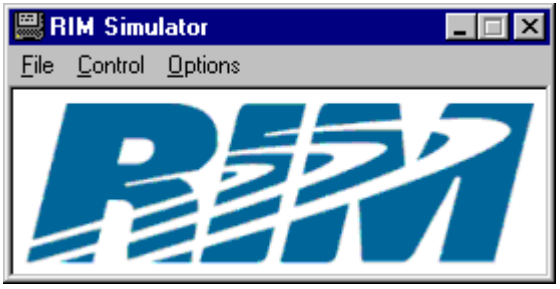
They are launched in different ways

- `SIMULATOR.EXE` provides a Windows interface to the `OSLOADER.EXE` program and is normally launched using shortcuts.
- `OSLOADER.EXE` provides a command-line interface and is normally launched using the command line.

For both simulators, you must load applications. Each DLL loaded into the simulator environment is a separate application; which DLLs you load depends upon which RIM 950 Wireless Handheld application you want to simulate.

Run the simulator using shortcuts

The install process creates two icons in the **RIM 950 Wireless Handheld 1.7 SDK** program group – an SDK **readme.txt** icon and a **Simulator icon**. Double-click the **Simulator** icon to start the simulator. The Simulator window appears:



The Simulator Window

1. First, select a simulation platform from the Options menu. The Configure… item in the Options menu allows you to add new simulation platforms or versions if you have them available as RIM OS DLLs. The Configure… item also allows you to specify the path to OSLOADER.EXE. Other than selecting a simulation platform, the defaults should be appropriate.

2. When you are ready to start the simulation, choose Start Simulation from the Control menu. You will be prompted for command line switches for the OSLOADER.EXE program; see below for descriptions of the possible switches.

3. After specifying the switches (if any), you will be prompted to load Application DLLs via a file selector window. You can load more than one file at a time by clicking Open after each file or by selecting more than one file in the window and clicking Open.

4. Once all files have been loaded, click Cancel in this window; the RIM Wireless Handheld simulator will start.

Running the simulator from the command line

You can also start the RIM Wireless Handheld simulator by running the executable OSLOADER.EXE. (This executable is located in the SDK install directory that you selected during the setup process.) The first parameter to OSLOADER.EXE must be an OS DLL, such as OSPGRMB.DLL.

You can optionally load applications automatically by naming them after the OS DLL on the command line, if the directory (or directories) containing the DLLs is specified in the Windows PATH. For example:

`OSLOADER.EXE [`*`Switches`*`] OsPgrMb.dll [`*`applications`*`]`

If you don't specify applications, you will be prompted with a file selector window. You can load more than one file at a time by clicking Open after each file or by selecting more than one file in the window and clicking Open.

Once all files are loaded, click Cancel in this window; the simulator will start.

Place the DLLs in your PATH or place them in the same directory and use that directory as the current directory when you execute the command.

The following table summarizes the command line switches and arguments, in alphabetical order; these are explained in greater detail later in this section.

Switches	Description
/A*n*	Use *n* sectors of flash memory for OS and application storage.
/D*n*	Use *n* sectors of flash memory for file system data storage.
/E	Erase flash allocation log.
/F*n*	Simulate *n* kilobytes of flash memory.
/Pf	Force prompt for more applications.
/Ps	Always suppress prompt for more applications.
/R*n*	Use serial port *n* for RAP modem.
/RDIR=*directory*	Store messages in *directory*.
/RSIM=*addr*	Use *addr* as the simulator's address.
/S*n*	Use Windows serial port *n* in place of the wireless handheld's physical serial port.
/T[*flags*]	Set or suppress traps

Any argument that does not begin with / is taken as a file name, the name of an application to be loaded.

Loading applications

Each DLL is a different application. The DLLs you load depend on which RIM Wireless Handheld applications you want to simulate.

For example, a simple application that uses only the functionality of the base API functions will not depend on other DLLs and can be loaded by itself. However, applications built with the User Interface Engine require loading the UI32.DLL file along with the UI-enabled application.

There is a restriction: Each DLL loaded in a group must require only the DLLs selected before or listed in the current set. Therefore, the application with the most dependencies should be added last. This

restriction does not hold if the required DLLs are in your PATH or the working directory that was used to start the simulator.

To simulate an application running with the full RIM Wireless Handheld messaging application, load all of the DLLs in the directory. This should be the following DLLs:

- ADDRESS.DLL
- AUTOTEXT.DLL
- COREAPI.DLL
- CRYPTOBLOCK.DLL
- DATABASE.DLL
- MESSAGE.DLL
- RIBBON.DLL
- TRANSPORT_MDP.DLL
- UI32.DLL

Please refer to the specific API documentation for each of these DLLs for more information.

Note Note that SDKRADIO.DLL is not a standard application DLL for the simulator; it is used when you want to use your RIM Wireless Handheld as a modem (see Using the RIM Wireless Handheld as the modem on page 30). Normally you will not load SDKRADIO.DLL into the simulator.

Using the simulator

When you run the simulator, a window featuring a picture of the wireless handheld appears on your screen. This main window contains pictures of a keypad, a trackwheel, and a LCD. You can operate the keypad and trackwheel from Windows, and the LCD will show you what would be displayed on the actual wireless handheld.

The RIM Wireless Handheld simulator

Keypad

You can operate all keys on the simulator window's keypad by clicking them with the mouse. In addition, you can operate the keys on the RIM Wireless Handheld keypad by using the corresponding keys on the PC keyboard.

TheRIM Wireless Handheld simulator

ALT key

Alternate symbols for the keys (in red) can be accessed using the ALT key. However, due to the special function assigned to the ALT key by

Windows, activate the wireless handheld's ALT key by pushing the CTRL key on your PC.

The current state of the Alt key is displayed on the upper right corner of the simulator. A reverse-video 'A' will be displayed when the ALT key is in effect.

SHIFT key

Press and release the CAP or SHIFT key to capitalize the next key you press. When this key is pressed, an up arrow will appear on the top right corner of the screen.

PC mode and Pager mode

The keyboard can be toggled between two modes: PC mode and Pager mode. In PC mode (the default), all symbols can be entered, and the simulator internally generates the keystrokes required to produce each symbol. In Pager mode, however, only the keys present on the wireless handheld can be used.

The Keyboard menu

Enabling backlighting

To switch on the backlighting, press the ALT key three times successively; the background of the LCD turns blue. The backlighting will stay on until an idle period of about 10 seconds has been detected.

Trackwheel

The trackwheel has two functions: to be rolled, and to be used as a pushbutton. You can operate the trackwheel using either the mouse or the arrow keys on the keyboard

Using the keyboard

The left and right arrow buttons on the PC keyboard simulate the clicking of the trackwheel while the up and down arrow buttons simulate the rolling.

Using the mouse

To perform the roller action using the mouse, click the trackwheel with the left mouse button and drag vertically.

To click the trackwheel, depress the right mouse button. This action allows the trackwheel to be rolled and clicked independently. In addition, the simulator provides full support for the Microsoft Intellimouse. You must first ensure that the correct Microsoft Intellimouse drivers are installed. Using the Microsoft Intellimouse, the mouse's roller can be used to operate the wireless handheld's trackwheel. However, the Intellimouse roller only works when the mouse pointer is over the RIM Wireless Handheld simulator.

Due to Windows filtering of events, sometimes trackwheel and right mouse button clicks are not handled during, and immediately after rotating the Intellimouse mouse wheel. This anomaly only occurs when using the trackwheel on the Intellimouse.

LCD

The LCD size can be toggled between full size (with correct aspect ratios), and actual size (how readable a display screen will be on the actual hardware). To change the size of the LCD on your simulator, select Display > Change size.

Simulation menu

The RIM Wireless Handheld simulator allows you to simulate the holster environment as well as various battery conditions of the Wireless Handheld. Under this menu, select the conditions you want to simulate.

The Simulation menu

In the Radio Simulator Control panel dialog box, you can simulate various coverage conditions.

Simulating serial I/O

The simulator has the ability to use one of the PC's serial ports as the RIM Wireless Handheld*'s* serial port. This option must be enabled via the `/S` command line option when you start the simulator. If no `/S` option is specified or the serial port is not available at startup, you cannot open or use the serial port from the applications running on the simulator.

When opening a serial port on the PC, the orientation of the serial port on the PC is as it would normally be. Thus, if you wish to simulate a connection between the RIM Wireless Handheld and the PC via the provided serial port, you must connect the PC's serial port to another PC's serial port via a null modem cable to communicate. If both your simulator and your application connected to the RIM Wireless Handheld are running on the same physical PC, they must be connected to different physical serial ports on your PC and the serial ports must be linked using a null modem cable.

Flash simulation files

In order to simulate the non-volatile nature of flash memory, the RIM Wireless Handheld simulator loads the flash contents from a file on startup and saves them back to the file on exit. The state of the simulated flash is preserved in a file in the working directory named `OS`*`XXXYY`*`.DMP`, where *XXX* is one of `PGR` (pager), `HH` (handheld), or `OEM`, and *YY* is one of `MB` (Mobitex) or `DT` (DataTAC).

When the RIM Wireless Handheld simulator starts, it determines the simulated flash size. The size can be specified using the /F command line option and the default size is 2048 KB. The simulator now checks to see if there is a file named OS*XXXYY*.DMP (where *XXXYY* corresponds to the current simulation platform) in the working directory. If the file exists, its size must be less than or equal to the simulated flash size, otherwise the simulator will report an error and abort. If the file does not exist, the RIM Wireless Handheld simulator will create it, and its size will be equal to the simulated flash size. Regardless of whether the DMP file existed at startup, the RIM Wireless Handheld simulator will always save the simulated flash contents on exit.

If the DMP file is smaller than the flash size, it will be extended with 0xFF's up to the flash size. This will invalidate any flash allocation information present in the file. This facility is provided for use with old DMP files only and should be avoided in normal operation.

You must be careful in the design of your algorithms so as to minimize the impact of the flash file system on their performance. The performance of your algorithms will depend on the file system. Therefore, you may have to alter your design so that it is compatible with the RIM Wireless Handheld file system. More information is available elsewhere in this guide.

Notes on older simulator versions

With older simulator versions, the flash file was named FILESYS.DMP. If you wish to use an old flash file in the new version of the simulator, simply rename it to the form OsXxxYy.dmp, and place it in the working directory. Note that this is only guaranteed to work if the flash file is the same size as the flash size for the simulator. If the file is larger than the flash memory of the device, the simulator will warn you and not load the file. If the file is smaller than the flash memory of the device, the simulation may work correctly. If in doubt, erase the DMP file.

Also note that older versions of the simulator will create a FILESYS.DMP that is the size of the file system, while newer versions of the simulator will create a DMP file that is the same size as the flash memory of the device that is being simulated. Newer versions of the simulator default to a 2048 KB flash size, while older versions default to a 1024 KB flash size.

Flash allocation

The available flash memory (either simulated or real) is divided into four areas:

- File system data area
- Unused area
- OS and application code area
- Fixed use area

Areas are allocated by writing an entry to a Flash allocation log, which is a special data structure stored within the fixed use area. The log may be completely erased using the `/E` command line option. Flash allocation information will no longer exist and a default allocation will be used until a new allocation is performed.

A new entry is added to the log whenever one or both of the `/D` or `/A` command line options are specified. When all of the available log entries have been used, the RIM Wireless Handheld simulator will not allow any more entries to be written and the log is erased.

If the simulator finds no valid log entry in the `DMP file`, a default flash allocation is used.

Modem simulation using a physical modem

When simulating the radio network using a RAP modem or the RIM Wireless Handheld, the RAP modem or wireless handheld becomes the simulator's radio modem. When data packet is sent by an application, the packet is sent to the physical modem via the serial port. Packets received by the physical modem, as well as status information, are sent through the serial port to the simulator running on the PC, and passed on to the applications.

When using the SDK with a physical modem, the applications are essentially dealing with a live radio network. Dealing with a live network requires that you have a RAP modem or the RIM Wireless Handheld with a valid subscription and that you are operating in an area that has radio coverage.

When using a physical modem with the simulator, you cannot simulate different coverage situations in software; you are limited by what the modem is actually experiencing. To simulate different coverage situations, change the position of the antenna or obstruct the antenna's coverage. The dialog box for manually controlling radio coverage situations is unavailable when running the simulator with a RAP modem, as its behaviour is determined by what the actual modem experiences on the live network.

Command line options for modem simulation using a RAP modem

`/R1 /R2 /R2...`

This option sets the radio serial port number. With this command line option, the simulator simulates network traffic by communicating to a real modem.

The simulator is able to communicate with the Allpoints wireless modem from Megahertz and all RIM modem products for your network, including the RIM 957, the RIM 902M OEMs, and the RIM 950 Wireless Handheld.

Using the RIM Wireless Handheld as the modem

The RIM Wireless Handheld is an integrated embedded system and radio modem. Although radio functionality is available to applications running on the Wireless Handheld, this functionality is normally not available through a serial connection. To allow the SDK's Windows simulator to communicate with the RIM Wireless Handheld, a special program, `SDKRADIO.DLL`, must be loaded onto the RIM Wireless Handheld to enable it to communicate radio information over the serial port. You can load `SDKRADIO.DLL` with the `PROGRAMMER.EXE` utility by typing:

```
PROGRAMMER LOAD SDKRADIO.DLL
```

When `SDKRADIO.DLL` is loaded, it automatically detects when the simulator is attempting to contact the RIM Wireless Handheld, and opens the COM port for communication. When not connected to the Windows simulator, the `SDKRADIO.DLL` does nothing. It can coexist with your regular applications on the RIM Wireless Handheld without harmful effects.

When using the wireless handheld as your modem, remember that the RIM Wireless Handheld was not designed as a high-capacity modem.

Battery capacity, as well as other factors cause the rate of packet delivery to be cut back after sending large amounts of data. In addition, when sending large amounts of traffic with the RIM Wireless Handheld, its battery life may be considerably shortened. During transmission of data packets, for example, the RIM Wireless Handheld requires three times more power than normal average power consumption.

Modem simulation using the file system

Unless you follow the instructions given in Modem simulation using a physical modem, the simulator uses the file system to simulate the modem. This allows you to test the communications functionality of an application without actually connecting to a radio network. This simulator offers the follow advantages over using the real network:

- No external hardware is required
- No network account is required
- Coverage situations can easily be created
- It is possible to monitor what is sent without the need for extra tools
- It is easy to write programs to send and receive data packets as a host side

When simulating the network using the file system instead of communicating data packets across a live network, the modem communicates packets across the file system. To send a data packet, the simulator creates a file whose name indicates the destination address. Conversely, the simulator checks for the presence of files whose name indicate that they contain data addressed to it.

A separate control panel window, the Radio simulation control panel dialog box, is available which allows the user to set coverage conditions.

For information specific to your network, such as the default simulation environment, see the Radio API Developer's Guide.

Command line options for modem simulation using the file system

The simulation environment, with no command line options, will default to simulating in the current directory, with the Radio simulation control panel dialog box not visible upon startup.

`/RSIM=<address>`
This option sets the address to simulate.

`/RDIR=<directory>`
This option sets the directory to use for the simulation. The default directory is the current directory. Radio simulators must point at the same directory in order to be able to communicate with each other.

Simulating email

Sending and receiving email through the `MESSAGE.DLL` requires a service book entry; the only way to get a service book entry onto the simulated RIM Wireless Handheld is to register with RIM server software. In turn, this requires a RIM Wireless Handheld with an active account.

The easiest way to register your simulated RIM Wireless Handheld device is to use your RIM Wireless Handheld as a modem; you can then register the simulated RIM Wireless Handheld with the RIM server software.

If your application uses the Radio API calls directly, you don't need to register the simulated RIM Wireless Handheld; registration with the server is only required if your application uses the calls from the Messaging API.

Radio simulation control panel dialog box

When the `/RDIR=<ADDRESS>` or `/RSIM=<directory>` options are used, the dialog box for controlling network traffic is automatically made visible. This dialog box can also be invoked, or brought to the foreground, by selecting Radio simulator control panel from the Simulation menu.

The specifics of the radio simulation control panel dialog box depend upon the wireless network you are using. See the Radio API Developer's Guide for information such as:

- Simulation file protocol
- Format of the files simulating data packets
- Limitations of the simulation
- Specific behaviors of the simulation

- Guidelines for writing a host-side application to simulate the network

This section describes the network simulation in general terms.

The control panel consists of two parts: reception and transmission. The Reception section of the dialog box allows you to set whether or not the modem is in coverage. The **Transmission** section allows you to set the success rate for packet transmission.

Reception section

In the **Reception** section; you may choose to have the modem out of coverage or in coverage; if you have the modem in coverage, you can control the Received Signal Strength Indicator (RSSI) by sliding the control bar back and forth. Note that the signal strength varies from -120 dBm to -50 dBm (dBm is dB with respect to milliwatts). A signal of -120 dBm is extremely weak, and the modem typically loses coverage before reaching this number. A signal of -50 dBm is stronger than you are likely to see on the actual network.

You may check for data packets immediately; many networks implement a power save mode, where the network periodically indicates which modem to send to. To conserve power, modems only turn on their receivers during the periods they are being sent data. To simulate this, the program only checks for packets every 10 seconds. You can force a check for packets by selecting the button.

This window also indicates the current status and the number of data packets received.

The following statuses are possible in the Active Status field:

State	Description
Radio Off	The radio is turned off.
Turning Off	The radio is in the process of turning off.
Stop Reception	Radio reception is stopped (`RadioStopReception` has been called).
Active	The modem is in coverage on the network.

State	Description
Checking...	The radio is checking for data on a 10 second interval, or the Check now button was pressed.
Checking for 10 sec.	The radio is checking continuously because of a recent data packet sent or received.
Out of coverage	The modem is out of coverage.

Transmission section

In the **Transmission** section, you may choose how frequently data packets are transmitted successfully. You may choose to have all transmits succeed, to be prompted for each data packet's success (useful for simulating very long transmit delays), or have a percentage of packets transmit successfully, packets chosen randomly.

This section of the dialog box also contains the Transmit status field, which indicates the status of the last or current packet that was submitted for transmission. It can be one of the following states:

State	Description
Pending	The modem is currently attempting to send a submitted packet.
Transmit Done	The last packet was successfully sent to the network
Transmit Failed	The last packet failed to reach the network.

Simulator command line options

The SDK environment can be configured using the following command line arguments:

/Sn

Use Windows serial port *n* in place of the wireless handheld's physical serial port. Without this argument, no serial port will be used, and applications will be prevented from opening a serial port.

Examples:

```
/S1 /S2 /S3 /S4
```

`/Rn`

Tell the RIM Wireless Handheld simulator to use serial port *n* for a RAP modem. The RAP modem is used in place of the physical RF hardware in the simulation environment. If this argument is omitted, no serial port is used, and simulated radio communications will not be possible.

Examples:

```
/R1 /R2 /R3 /R4
```

`/Pf`

Force prompting for more applications By default, the RIM Wireless Handheld simulator prompts for applications on startup only if no applications are specified on the command line. Cannot be used with `/Ps`.

`/Ps`

Suppress prompting for more applications. By default, the RIM Wireless Handheld simulator prompts for applications on startup only if no applications are specified on the command line. Cannot be used with `/Pf`.

`/Fn`

Simulate *n* kilobytes of flash memory. If this argument is omitted, the default size of 2048 KB is used. Note that older versions of the simulator will have a default flash size of 1024 KB.

Examples:

```
/F1024 /F2048
```

`/E`

Completely erase the flash allocation log. Flash allocation information will no longer exist and a default allocation will be used until a new allocation is specified. This option has to be used when all Flash allocation log entries have been used.

`/Dn`

Simulate *n* sectors of 64 KB each for file system data storage. If this option is omitted, the previous amount is preserved, unless `/E` is simultaneously specified. If `/E` is specified, a default amount is used. This can simulate the result of a `PROGRAMMER ALLOC` command.

Example:

```
/D15
```

/A*n*

Simulate *n* sectors simulated flash memory (in 64 KB sectors) to be used for OS and application code storage. If this option is omitted, the previous amount is preserved, unless /E is simultaneously specified. If /E is specified, a default amount is used. This can simulate the result of a PROGRAMMER ALLOC command.

Example:

```
/A16
```

/T[*flags*]

Modify the stale pointer trapping behaviour of the RIM Wireless Handheld simulator.

By default, the RIM Wireless Handheld simulator will, at various intervals, move the location of the simulated file system on yield and on file writes. The simulator then maps the old location as invalid memory. Used pointers to the file system that should have been reloaded from the handle table result in page faults. The /T option can be followed by these symbols:

+	Turn subsequent options on
-	Turn subsequent options off
W	Trap on flash write
Y	Trap on yield

By default, both options are turned on.

Example:

This enables a trap on flash write and disables the trap on yield:

```
/TW-Y
```

<Filename>

Any argument that does not begin with a slash is assumed to be an application filename. Multiple application filenames may be specified on the command line. Filenames containing spaces must be surrounded by quotes.

Example:

In this example, `APP2.DLL` is assumed to reside in the working directory

`C:\dev\app1.dll app2.dll.`

Debugging hints

Some tips for debugging applications using the simulator (and the RIM Wireless Handheld itself).

- When the machine fails and requests a reset, you can often get additional information by typing `dbug`. This displays the contents of different registers.
- You may be able to get additional information on a reset request by typing `info`. If the failure was caused by an unresolved OS call, this command displays the name of the call.
- For some reset requests, pressing the R key will cause a reset.
- You can force a reset request with Alt-Shift-Backspace.

Program loader

The RIM 950 Wireless Handheld SDK includes a command line utility (`PROGRAMMER.EXE`) to load the compiled Windows DLLs onto the physical wireless handheld. This utility will take the same DLL that was debugged under Windows, and perform all necessary steps to load it on to the Wireless Handheld. The utility can also be used to manipulate applications that are already installed on the wireless handheld.

The `PROGRAMMER.EXE` utility performs many of the application loader functions of the Desktop Manager utility shipped with every RIM Wireless Handheld; it also performs tasks more suited to programmers' needs.

Program loader command line options

The PROGRAMMER.EXE utility requires a set of command line options to denote serial port configuration, the action required, and what files are to be manipulated. The command line template is:

PROGRAMMER [[-P*port*] [-S*speed*] [-W*password*] [*command*]

If no command action is given, PROGRAMMER.EXE displays help information.

Universal command line options

The -P, -S and –W options may be given with any action.

-P*n*
: Use serial port *n*. The default is COM1.

-S*speed*
: Set baud rate for the connection to the RIM Wireless Handheld. The default is 115200 baud.

-W*password*
: Use given *password*, if password protection has been enabled on the RIM Wireless Handheld.

Program loader commands

The following command actions are supported:

Command	Description
LOAD [-S] [-G] [-V] [-M] [-D] *apps* or (*groups*)	Load applications or groups of applications on the RIM Wireless Handheld
ERASE [-A|-E] *apps*	Erase applications and optionally OS both from Wireless Handheld
DIR [-S]	List applications currently on the Wireless Handheld
HELP [*command*]	Display help information, possibly about command

Command	Description
BATCH *filename*	Run PROGRAMMER.EXE commands stored in a file
VER	List applications currently on the Wireless Handheld, including version information
MAP [-F] [-R]	Display detailed flash and RAM maps
LOADFS	Load file system data from a .DMP file onto RIM Wireless Handheld
SAVEFS	Save file system data from RIM Wireless Handheld into a .DMP file
WIPE [-F | -A]	Irreversibly erase applications or file system or both from RIM Wireless Handheld
NUKE	Erase all flash on RIM Wireless Handheld, including file system, applications, and password
ALLOC [-E] [-D *sectors*] [-A *sectors*]	Moves breakpoint between application memory and file system memory.

LOAD command

```
LOAD [-S] [-G] [-V] [-M] [-D]<apps or groups>
```

Where

-S specifies that symbol information for all new applications should be appended to a DEBUG.DAT file in the current directory.

-G specifies that the first application or group of applications should be grouped with the last group found on the RIM Wireless Handheld.

-V specifies checking file versions. Normally PROGRAMMER only checks API versions and dependencies, since developers are unlikely to update the file information for every build.

-M specifies mappings of unresolved OS calls should be displayed. As of version **1.7**, the loader maps unresolved OS calls to an internal

API, `RimCatastrophicAPIFailure()`. In all cases, other unresolved links fail; only unresolved OS calls can be mapped.

`-D` specifies that unresolved OS calls should *not* be mapped to `RimCatastrophicAPIFailure()`.

`<files or groups>` are one or more files or groups of files to be loaded onto the RIM Wireless Handheld. Individual files are specified alone. Groups of files are enclosed in parentheses, brackets, or braces. There must be spaces surrounding the brackets, as in the example below.

Description

The `LOAD` command loads new applications or the application environment onto the RIM Wireless Handheld. Any old applications by the same name are erased. Applications can and should be grouped to reduce wasted space on the RIM Wireless Handheld. However, if an application is to be replaced separate from other applications on the RIM Wireless Handheld, it should be placed in its own group.

Note When loading or replacing the application environment (`PAGER950.BIN`), it must be specified first on any Load command.

Grouping of programs

Because the flash memory can only be erased one 64 kilobyte sector at a time, any application that is to be erased and have its space reclaimed must not overlap with other applications in the same 64 kilobyte sector. Grouped applications are contiguous, without regard to the 64k sector boundaries. As such, when invalidating an application that is part of a group, the space cannot be reclaimed without erasing other applications as well. Applications that are not grouped each occupy one or more 64 kilobyte sectors, with the remainder of the last sector used being wasted.

Note The ability to group programs is specific to the `PROGRAMMER.EXE`. Although the desktop loader program will recognize groups already on a RIM Wireless Handheld, it cannot load files in groups.

Examples

The following command will load the application environment and applications onto the RIM Wireless Handheld:

```
PROGRAMMER LOAD PAGER950.BIN UI32.dll ( Address.dll
AutoText.dll Message.dll Options.dll
Transport_RTP.dll )
```

The following command will load a new calculator application, grouping it with the other applications:

```
PROGRAMMER LOAD -G Calculator.dll
```

ERASE command

```
ERASE –A
```

```
ERASE <application names>
```

Where

`-A` specifies that all applications and the application environment are to be deleted.

`<application names>` are the names of specific applications to be deleted.

Description

The `ERASE` command erases applications currently loaded on a RIM Wireless Handheld. The names are not case sensitive and can be obtained using the `PROGRAMMER DIR` command.

The space occupied by erased applications is only reclaimed when the entire group that it resides in is erased.

Example

The following command will erase all applications on the RIM Wireless Handheld:

```
PROGRAMMER ERASE -A
```

The following command will erase only the address book application:

```
PROGRAMMER ERASE ADDRESS.DLL
```

DIR command

```
DIR [-S]
```

Where

`-S` specifies a short listing of only the application names.

Description

The `DIR` command generates a listing of the applications currently loaded on a RIM Wireless Handheld. Unless the `-S` option is specified, this listing includes the names of the applications and the amount of flash and RAM occupied by the applications. The applications are grouped as they are grouped on the RIM Wireless Handheld.

Example

The following command will list the applications on the RIM Wireless Handheld:

```
PROGRAMMER DIR
```

VER command

```
VER
```

Description

The `VER` command generates a listing of the applications currently loaded on a RIM Wireless Handheld, including version information. "Version information" includes release tags and build times. The applications are grouped as they are grouped on the RIM Wireless Handheld.

Example

The following command will list the applications on the RIM Wireless Handheld, including release tags and build times:

```
PROGRAMMER VER
```

BATCH command

```
BATCH <batchfile>
```

Where

`<batchfile>` is the name of a file containing commands.

Description

The BATCH command allows multiple commands to be placed in a file and executed with a single, short command. This command is useful when repeatedly performing the same load process, and for overcoming the fixed limit imposed on command lines.

The batch file may contain one or more of the LOAD, ERASE, DIR, or BATCH commands. The results are committed to the RIM Wireless Handheld only if all commands are successfully completed. Each line of a batch file can be no more than 256 characters long. Single commands can be broken into multiple commands, if you have long file names.

WIPE command

```
WIPE [-F | -A]
```

Where

-F specifies that the file system should be wiped.

-A specifies that the application area should be wiped.

If no option is specified, both the file system and application are wiped.

HELP command

```
HELP [<command>]
```

Where

<command> is the name of the command for which you want help, or the word errors.

Description

The HELP command invokes the built-in help system. With no options a generic help message is produced. Help for a specific command can be obtained by specifying it as an option. Help about error messages can be obtained by specifying the errors command.

If the output is not redirected to a file, the help system uses a built-in paging system. Press any key at the <MORE> prompts.

Example

To get help on the LOAD command, type:

```
PROGRAMMER HELP LOAD
```

To get help on errors, type:

```
PROGRAMMER HELP ERRORS
```

To get a main help page, type:

```
PROGRAMMER HELP HELP
```

ALLOC command

```
ALLOC [-E] [-D <sectors>] [-A <sectors>]
```

Where

`-E` specifies the File Allocation Sector to be erased before writing an entry.

`-D<sectors>`
specifies the new size of the file (or data) area in flash sectors; the minimum size is 1.

`-A<sectors>`
specifies the new size of the OS and applications area in flash memory.

If no options are specified, the command reports on current usage.

Description

This command writes a new entry to the Flash Allocation Log and moves the breakpoint between File area and the OS and applications area of memory. It is possible to move only one, but then you would have an unused sector.

The command will only reduce the size of an area if the sectors to be removed are completely blank. This is rarely a problem with the OS and applications area, but may be a problem with the File area, because there are no guarantees about placement of information.

When the File Area size is decreased, at least one blank sector must remain within the area. Back up and pack your data before issuing the ALLOC command.

Troubleshooting

Most error messages are self-explanatory. The following errors deserve more discussion.

Error: Unable to connect to device

An error occurred trying to initiate communications with the RIM Wireless Handheld. Make sure that it is connected to the computer properly.

Error: Insufficient flash or RAM

There is not enough flash or RAM memory remaining to load a new application. Make sure that you have erased any old applications. If you have been erasing and loading often, you may have fragmented your flash space. In this case, try erasing all applications and start again. Loading applications as part of the same group will cause them to occupy flash more efficiently.

Error: Relocation failed

It was not possible to relocate an application. This typically occurs if the application environment (`PAGER950.BIN`) is not specified first in a `LOAD` command. When loading or replacing the application environment, it must always be specified first in the `LOAD` command.

Error: Bad load

This error occurs when a `LOAD` is interrupted or done improperly. This is a serious error that requires the RIM Wireless Handheld to be reset and its operating system reloaded. In order to reset the RIM Wireless Handheld, insert a small pointed instrument, such as a paper clip, into the small reset hole on the back of the RIM Wireless Handheld. To reload the operating system, follow the example instruction in the `LOAD` section on page 39 of this manual.

Error: Not all imports resolved

An application is requesting imports from another application that cannot be exported by the other application. Ensure that you are loading all applications that provide the exports that you need.

DLLUtil

The DLLUTIL.EXE utility provides information about DLLs as they will be when loaded onto the RIM Wireless Handheld, similar to the output of PROGRAMMER DIR and PROGRAMMER VER. There are two forms of the command:

```
DLLUTIL SIZE [-R] files
DLLUTIL VER files
```

The *files* argument is a list of DLLs and/or OS files, and can contain wildcards (* and ?).

In the first form (SIZE), the utility displays a dump of flash memory, RAM, and thunk resources. With the –R switch, the utility displays the size of the DLLs without the .reloc (DLL relocation) information loaded. This is how programs were loaded by versions of the PROGRAMMER.EXE utility 16 through 18. Without –R, the utility displays the sizes as loaded by PROGRAMMER.EXE versions 20 and greater.

In the second form (VER), DLLUTIL displays version information. Beside each valid file, it lists versions for both imported and exported APIs. It also lists valid operating systems along with file version information.

Conversion utilities

The SDK contains three small conversion utilities intended for creating fonts and bitmaps for the RIM Wireless Handheld.

For information about using these utilities, see chapter 3 of the OS API Developer's Guide.

BITMAP

The BITMAP.EXE executable creates bitmap header file from a definition file (with a .DEF extension). The header file is then included in a resource DLL or an application.

The syntax is:

`BITMAP` *`definition header`*

The definition file format is described in the OS API Developer's Guide, under "Creating bitmaps."

BMP2DEF

The `BMP2DEF.EXE` executable creates a definition file from an existing bitmap (`.BMP`). The definition file can be turned into a RIM Wireless Handheld bitmap header file using `BITMAP.EXE`.

The syntax is:

`BMP2DEF` *`bitmap definition`*

LCDFONTS

The `LCDFONTS.EXE` executable creates a font header file from a definition file. The header file is then included in a resource DLL or an application.

The syntax is:

`LCDFONTS` *`definition header`*

The definition file format is described in the OS API Developer's Guide, under "Creating custom fonts."

5

An overview of the system

This section describes parts of the RIM Wireless Handheld's system for developers, concentrating on the operating system.

When developing for the RIM Wireless Handheld, it's important to realize that the RIM Wireless Handheld has its own operating system. Some of the features of the operating system (such as the use of flash memory rather than a traditional file system) are based on the requirements of a portable wireless device.

When developing applications for the RIM Wireless Handheld, you should be aware of the following:

- A co-operative scheduler in the multitasking operating system
- Tasks and threads are used to run applications
- Because the OS is co-operative, each task must be able to yield so that other tasks may use the CPU
- Tasks communicate using MESSAGEs
- The system uses two types of memory; this is normally hidden by the file system handles
- The user interface is somewhat different from a traditional GUI interface; users and applications communicate through the task wheel, the keyboard, and the display; applications may be selected through the application ribbon
- For some (but not most) applications, you may need to know details of the underlying wireless network; see the Radio API Developer's Guide for information about your network. There are a large number of APIs provided

The rest of this chapter describes these things in slightly more detail.

RIM co-operative scheduler

The RIM Wireless Handheld's operating system (OS) is designed using a co-operative multitasking model. For more information on the operating system and the OS functional specifications, see the RIM 950 Wireless Handheld Operating System API Developer's Guide.

Tasks and threads

A normal application is developed as a task or thread. In order to create a new task:

- a `PagerMain()` function must be defined
- an application name must be chosen
- an application stack size must be specified

Each `PagerMain()` function is called during the initialization process and is used to construct and initialize objects needed by the application. For more information and an example, see "Coding an application" on page 55. For more information on creating new tasks, see the RIM 950 Wireless Handheld OS API Developer's Guide.

Task yielding

Since the operating system is a co-operative multitasking model, the `PAGERMAIN()` function (and all other code) must include a task yield function call to allow other tasks to make use of the CPU. The application can be restarted in a number of ways:

- In response to a `MESSAGE` posted by a system thread (for example, the real time clock)
- In response to a `MESSAGE` posted to the task by another task
- In response to a user selection through the Ribbon (selecting an icon on the main screen)
- In response to a user selection through the Options List

For more information on the MESSAGE posting process, see "MESSAGE concept" below as well as the RIM 950 Wireless Handheld Operating System API Developer's Guide.

MESSAGE concept

The Operating System allows for inter-task communication through posting and receiving MESSAGES. For more information, see the RimPostMessage(), RimSendMessage(), and RimGetMessage() functions described in the RIM 950 Wireless Handheld Operating System API Developer's Guide. The RimPostMessage() and RimSendMessage() functions allow a task or system service to send a MESSAGE to a specific task or tasks. The RimGetMessage() function allows a task to yield control of the CPU until a MESSAGE is posted for it.

The PagerMain() function should include code to construct and initialize the necessary application objects, followed by a message loop: an infinite loop containing a RimGetMessage() function and code to call the appropriate application function for each expected RIM MESSAGE.

If you're using the UI, a message loop function is included.

Memory use

The RIM Wireless Handheld contains two types of memory: RAM and flash. RAM memory is volatile and can be accessed directly. Flash memory is non-volatile and can be accessed either directly or indirectly through calls to a file system. File system handles can be assumed to always point to the appropriate data. Flash memory can be accessed directly as if it were RAM, but with the following two restrictions:

- Flash memory is read only memory.
- The data may move if the application code yields control of the CPU (by either calling RimTaskYield() or RimGetMessage(), by making a file system call, or by calling another application that does one of these things).

For more information on file system use, see the RIM 950 Wireless Handheld OS API Developer's Guide.

User interface

The user can acquire data from the application through the RIM Wireless Handheld's LCD. In order to use the LCD, the application must make a request to become the foreground task. For more information on making an application a foreground task, see the RIM 950 Wireless Handheld OS API Developer's Guide. Once the task has become the foreground task, it can use OS level calls to display data on the LCD or alternatively use higher-level objects defined by the UI Engine.

The application can acquire data from the user through the keyboard, a system level device that posts a `MESSAGE` whenever a key is pressed. The application can acquire the information by getting that `MESSAGE` through `RimGetMessage()`.

A RIM Wireless Handheld user can manually activate an application by selecting that application's icon on the main screen. You create such an icon by registering a name and a bitmap using the `RibbonRegisterApplication()` function. The `DEVICE_RIBBON` will then post a `MESSAGE` to the application's task that can be used to invoke the appropriate application code. For more information, see the RIM 950 Wireless Handheld Ribbon API Developer's Guide.

The user can also activate an application manually by selecting its entry in the Options List. You create such an entry by registering a name and a callback function using the `OptionsEntry()` function. This function calls the callback function that has access to the LCD (has the foreground) through the Options thread. This approach should only be used to make application parameter changes and other changes of a limited nature.

API overview

This section provides an overview of the RIM Wireless Handheld Application Programming Interfaces (APIs), how they are organized, and some of the features that can be used in RIM Wireless Handheld applications.

The following APIs are provided. All but the most trivial applications will need to make use of the APIs described in the OS API Developer's guide and the User Interface Developer's Guide.

API	Description	Documented in
Address Book API	access to the database used by the Address Book application; create, edit, and retrieve contact information	Address Book API
AutoText API	access to the AutoText application; customize the editing of editable user text	AutoText API
Database / File system API	access file functions; the file system on the RIM Wireless Handheld is non-standard, to make better use of resources	OS API
Database API	access to the database functions; define new records, and to create, edit, and retrieve information stored in the database	Database API
Keypad API	customize the keypad; actual keystrokes are passed as messages from the application server	OS API
LCD API	low-level display functions, screen buffers, and string management; use the UI APIs for dialogs, menus, and edit classes	OS API
Messaging API	access to email and other messages	Messaging API
Options API	responsible for controlling system-wide programmingof the features, such as the date, time, and screen / keyboard settings	Ribbon API
Radio communications API	simplified access to radio network and status of packet communications	Radio API
Ribbon API	register an application on the functions list in the HOME screen	Ribbon API

API	Description	Documented in
Serial communications API	access and configure serial port on the RIM Wireless Handheld	OS API
System services API	Access to thread handling, messaging, and task switching, memory allocation, timers, and the system clock	OS API
User Interface API	Calls for high-level display of information and handling input, such as creating screens, menus, and dialogs	User Interface API

6

Building applications

This chapter describes the basics of writing an application for the RIM Wireless Handheld.

Coding an application

This section briefly outlines the required components of an application on the wireless handheld. The basic program structure essentially consists of an infinite loop that receives messages from the OS, and code to process those messages.

For every RIM Wireless Handheld application:

- Create an entry point named `PagerMain()`
- Register the application, providing a version string and a stack size
- Set up a message loop
- Define version information
- Build the application as a Windows DLL target

The PagerMain entry-point function

Each RIM Wireless Handheld application must have an entry point function named `PagerMain` with the following prototype:

```
void PagerMain( void )
```

The `PagerMain()` function has the format shown in the example on the following page. The following guidelines will help you create the `PagerMain()` function:

- Begin the `PagerMain()` function with the construction of any permanent objects required by the application. You should avoid accessing objects in other DLLs because until the `PagerMain()` functions in these DLLs have had a chance to run, the construction of these objects may not have occurred.
- After this local initialization phase, call `RimTaskYield()` to allow the `PagerMain()` functions of other tasks to run.
- After `RimTaskYield()` is called, you can expect that the other tasks' permanent objects have been created. Now call those functions that make the `PagerMain()` function itself known to other tasks (for example, `RibbonRegisterApplication()`).
- The function can display any user interface elements that will be visible to the user if the task is brought to the foreground.
- Finally, if your application needs to respond to `MESSAGES` posted by other tasks or by system devices (such as the real time clock), include an infinite loop containing a `RimGetMessage()`. The `RimGetMessage()` task yields control of the CPU until a `MESSAGE` is posted for the task.

For more information on `MESSAGES`, system devices, and task yielding, see the RIM 950 Wireless Handheld Operating System API Developer's Guide.

Registering the application

RIM Wireless Handheld applications must have two variables defined globally — the version string and the stack size.

The version string is used to register the application with the task switcher in the operating system. The name will appear when the task switcher is run. The definition should look like this:

```
char VersionPtr[] = "My Application";
```

The stack size is used by the system when creating the initial thread for the application. The value should be sized according to the needs of your application (the `RimStackUsage()` call will help with this). The definition should look like this:

```
int AppStackSize = 4096;
```

Here is some example `PagerMain()` code for an application that makes use of the database:

```
#include <Pager.h>
#include <Ribbon.h>
//Definition of bitmap registered with ribbon
#include <bitSample.h>
char VersionPtr[] = "Database Sample";
int AppStackSize = 5000;
// Function which creates the main thread
void
PagerMain( void )
{
   // Yield to let other applications initialize
   RimTaskYield();
   // Register the application with the Ribbon
   // so an icon is shown on the main screen
   // The & designates 'D' as the hotkey.
   RibbonRegisterApplication( "&Database Sample",
      NULL, 0, 0 );
   // Initialization complete:
   // set up LCD and enter message loop
   .
   .
   .
};
```

Entering the message loop

Handheld applications spend a great deal of time in the message loop. This code block should get required application messages and perform required operations. It is not necessary for all applications to react to all messages.

If your application makes use of the UI engine, there are several routines for processing specific kinds of messages (such as `ProcessScreen()` and `ProcessMenu()`).

If you need to write a message loop, the typical structure for this loop is a non-terminating loop with a switch statement to delegate message processing to different functions. For example:

```
#include "Pager.h"

void PagerMain( void )
{
   // initialization as in previous section
   .
   .
   .
   // Message loop
   MESSAGE msg;
   // perform initialization
   for (;;) {
      RimGetMessage( &msg );
      // respond to events
      switch( msg.Device ) {
      case DEVICE_SYSTEM:
         // handle SYSTEM messages
         break;
      case DEVICE_TIMER:
         // handle TIMER messages
         break;
      case DEVICE_KEYPAD:
         // handle KEYPAD messages
         break;
      }
   }
}
```

Note Applications are not required to be strict state machines.

It is possible to call `RimGetMessage()` from anywhere in the code. For more complex programs, every screen of your program should have its own message loop. Messages that do not pertain to the screen, such as radio messages, should be handled by calling a message handler, or even a separate thread.

Becoming the foreground task

Only foreground tasks may display on the screen, so an application that wishes to display data must first become the foreground task.

In this example, the application should be brought to the foreground if the user activates the application by selecting an icon on the main screen. When the user selects the icon, the OS sends a `MESSAGE` to the

application from the RIBBON device, with the event set to RIBBON_GRAB_FOREGROUND. If we get a RIBBON message, we call RimGetCurrentTaskID()for the current task handle. We bring the application to the foreground using RimRequestForeground().

```
if( msg.Device == DEVICE_RIBBON ) {
   if( msg.Event == RIBBON_GRAB_FOREGROUND ) {
      // Bring this application to the foreground
      // so it can use the LCD
      RimRequestForeground( RimGetCurrentTaskID() );
   }
}
```

Returning from the PagerMain() call will return the user to the task switcher, but it's more appropriate to return to the ribbon with RibbonShowRibbon(). This also gives up the foreground.

Using the Ribbon API

Instead of a system menu, the RIM Wireless HandheldRIM 950 Wireless Handheld uses a simple ribbon interface to select applications. To add your application to the ribbon:

1. Add #include <ribbon.h> to the beginning of your program.

2. Select Project > Settings and click on the Link tab. Add RIBBON.LIB to the Object/Library Modules field before LIBC.LIB.

3. Call the RibbonRegisterApplication() function at the beginning of PagerMain(), before the program enters the message loop. See the RIM 950 Wireless Handheld SDK Ribbon API Developer's Guide for more information on this function.

Your application can return to the ribbon by calling RibbonShowRibbon().

Handling user input

User input can be handled at the low level, but this is inefficient. We presume you're using the UI Engine.

If the user presses a key, the OS sends a MESSAGE to the application from the KEYPAD device. If we get a KEYPAD message, we pass the MESSAGE to the UI Engine, which returns a result code:

```
if( msg.Device == DEVICE_KEYPAD ) {
   // Let the UI process the message
   result = m_ui_engine.HandleInput( message );
   //Process UI Engine result
}
```

MESSAGEs resulting from a key being pressed on the keyboard are sent only to the foreground task.

If the UI Engine returns CLICKED, the application should display a menu. If the UI Engine returns UNHANDLED, the user may have pressed the BACKSPACE key, in which case we want to return to the main screen (RIBBON).

```
if( result == CLICKED ) {
   // Set up, display, and handle menu result
}
if( result == UNHANDLED ) {
   //Check for the backspace key.
   //If the user pressed backspace
   //we want to go to the home screen
   if( message.Event == KEY_DOWN &&
      message.SubMsg == KEY_BACKSPACE ) {
         RibbonShowRibbon();
   }
}
```

PagerMain function example

This is a more complicated example of a PagerMain() function; it includes task yielding, and makes use of items from other DLLs.

```
#include <Pager.h>
#include <Ribbon.h>
char VersionPtr[] = "Example Application";
int AppStackSize = 4096;
// Function identifying task to the OS
// These values are actually the defaults for
// a PagerMain() thread; see RimCreateThread()
// for more details
```

```
static
void
set_task_pid( void )
{
   PID pid;
   pid.Name = "Example Application";
   pid.EnableForeground = true;
   pid.Icon = NULL;
   RimSetPID( &pid );
}
// Function which creates the main thread
void
PagerMain( void )
{
   MESSAGE msg;
   set_task_pid();
   // Yield to let other applications initialize
   RimTaskYield();
   // Register the application with the Ribbon so an
   // icon is shown on the main screen
   RibbonRegisterApplication( "Database Sample",
      &bitmapSample,  0, 0 );
   // Initialization complete: set up LCD and
   // enter message loop
   for (;;) {
      RimGetMessage( &msg );
      // respond to events
      switch( msg.Device ) {
      case DEVICE_SYSTEM:
         // handle SYSTEM messages
         break;
      case DEVICE_TIMER:
         // handle TIMER messages
         break;
      case DEVICE_KEYPAD:
         // handle KEYPAD messages
         break;
      }
   }
}
```

To set up a basic program structure

1. Implement a message loop:

   Any application running on the device will need to handle messages from the OS that will notify the application of key presses, radio events, etc. The most efficient process is to do this is to use a message loop.

   The basic structure of the message loop is:

```
for ( ; ; ) {
   // waits until a message is given
   // to the application by the OS
RimGetMessage (&message);
   // processes message
}
```

2. An application that wishes to display data on the device's LCD screen must first become the foreground task.

   In this example, the application should be brought to the foreground if the user activates the application by selecting an icon on the main screen. When the user selects the icon, the OS sends a `MESSAGE` to the application from the `RIBBON` device, with the event set to `RIBBON_GRAB_FOREGROUND`. If we get a `RIBBON` message, we call `RimGetCurrentTaskID()` for the current task handle. We bring the application to the foreground using `RimRequestForeground()`.

```
if( message.Device == DEVICE_RIBBON ) {
   if( message.Event == RIBBON_GRAB_FOREGROUND ) {
      // Bring this application to the foreground
      // so it can use the LCD
      RimRequestForeground(RimGetCurrentTaskID());
   }
}
```

3. If the user presses a key, the OS sends a `MESSAGE` to the application from the `KEYPAD` device. If we get a `KEYPAD` message, we pass the `MESSAGE` to the UI Engine, which returns a result code:

```
if( message.Device == DEVICE_KEYPAD ) {
   // Let the UI process the message
   result = m_ui_engine.HandleInput( message );
   //Process UI Engine result
}
```

MESSAGEs resulting from a key being pressed on the keyboard are sent only to the foreground task.

4. If the UI Engine returns CLICKED, the application should display a menu. If the UI Engine returns UNHANDLED, the user may have pressed the BACKSPACE key, in which case we want to return to the main screen (RIBBON).

```
if( result == CLICKED ) {
   // Set up, display, and handle menu result
   }
if( result == UNHANDLED ) {
   //Check for the backspace key.  If the user
   // pressed backspace we want to go to the home
   // screen
      if( message.Event == KEY_DOWN &&
         message.SubMsg == KEY_BACKSPACE ) {
            RibbonShowRibbon();
      }
}
```

Example programs

Source code for sample applications is available in the SAMPLES directory where the SDK was installed.

You may want to read the Messenger Developer's Guide, which describes a sample application that makes use of the OS and database APIs.

Compiling RIM Wireless Handheld applications

You can create a RIM Wireless Handheld application by recompiling an application created for an earlier version of the wireless handheld.

1. Set up a Microsoft Developer Studio project (version 5.0 or later) as described above.

2. Select Project > Add to Project > Files…. Select all the source and header files that should be included in the project.

You should now be able to compile and debug your application using the RIM 950 Wireless Handheld SDK.

Installing the application

There are two utilities available to install the application on a RIM Wireless Handheld, the `PROGRAMMER.EXE` utility and the Desktop Manager software shipped with every RIM Wireless Handheld. Installing the application with `PROGRAMMER.EXE` has been covered (see page 39 for a description of the `LOAD` command action). Note that you cannot run the Desktop Manager and `PROGRAMMER.EXE` at the same time.

Normally, installation of a distributed application is done using the Application Loader in the Desktop Manager software. To make your application visible to the Desktop Manager software, you need to set certain registry keys; the Desktop Manager looks for those keys and creates an entry for your application.

Depending upon your distribution method, you can either set the keys directly or you can create an application loader information file with a `.ALI` file extension.

Registry keys for installation

The registry keys are installed in the hive under HKEY_LOCAL_MACHINE/SOFTWARE/Research In Motion/Desktop Software/Loader/Applications/DisplayName. Note that the Display Name is both the name of the key and a value in the key.

Value Name	Value Type	Meaning
Description	REG_SZ	The text displayed in the description field of the Application Loader when one of the items in the app list is selected. Corresponds to the `<DESCRIPTION>` tag of the ALI file.
DisplayName	REG_SZ	The name of the app as displayed in the Application Loader.

Value Name	Value Type	Meaning
		Corresponds to the <DISPLAY_NAME> tag of the ALI file.
Flags	REG_DWORD	See below. Generally 0x04 is the best value. Inferred from the optional <SCREEN_SIZE> tag of the ALI files.
Path	REG_SZ	Fully qualified path to the actual app DLL. In ALI files, it is inferred from the pathname to the ALI file by the Application Loader.
Version	REG_DWORD	Indicates version information associated with the app DLL. This isn't used directly by the loader, but is required. Corresponds to the <VERSION> tag of the ALI file.

The Flags is a bitset indicating information about this particular application DLL:

Name	Mask	Meaning
Reserved1	0x01	Reserved; must be zero
Reserved2	0x02	Reserved; must be zero
Reserved3	0x04	Reserved; must be one
SMALLSCREEN	0x08	Indicates that this app is intended for use only on the pager-sized RIM Wireless Handhelds
BIGSCREEN	0x10	Indicates that this app is intended for use only on the palm-sized RIM Wireless Handhelds

ALI file format

The ALI file is read by the Application Loader, which then sets appropriate registry keys. From that point on, your application is known to the loader.

The ALI file is a text file that contains three lines. The format is:

```
<DESCRIPTION>Description of the application
<DISPLAY_NAME>Name to be displayed
<VERSION>Version number
<SCREEN_SIZE>Model (RIM 950™ or RIM 957™) RIM Wireless Handheld — optional line
```

The angle brackets (<>) around the key names are required.

`<DESCRIPTION>`
: A brief description of the application that is displayed in the Description pane of the application loader. The value is an arbitrary text string.

`<DISPLAY_NAME>`
: The name for the application, displayed with a checkbox in the selection pane of the application loader. The list of applications is sorted by display name. The value is an arbitrary text string. This value will also be used for the registry key name.

`<VERSION>`
: A version number for the application. A value is required but is currently ignored by the application loader. The value must be in numeric format.

`<SCREEN_SIZE>`
: This tag is optional, and represents the model of the RIM Wireless Handheld. Valid arguments for this tag are `LARGE`, `SMALL`, and `ANY`. Respectively, these set the `BIGSCREEN` bit, `SMALLSCREEN` bits, or no extra bits.

The ALI file must be in the same directory as the DLL it describes, and it must have the same basename as the DLL it describes. For example, the API file for `MEMOPAD.DLL` is named `MEMOPAD.ALI`.

Application loader notes

You should be aware of the following when packaging applications for the Application Loader for the RIM Wireless Handheld.

Each application requires an ALI file or registry settings

The ALI file contains information used by the loader to add an application to its list of available applications. The ALI file must have the same basename as the DLL it describes and it must be in the same directory as the DLL it describes. See above for the format. The Application Loader uses the contents of the ALI file to set registry entries. You may choose to set the registry entries directly.

Each application must have a unique name

Neither the application loader nor the RIM Wireless Handheld will allow two applications with the same name. Each application must have a unique name. This is not normally a problem; the internal DLL name is used for comparison purposes. (The internal DLL name is the name of the output file passed to the linker when creating the DLL.)

The application name is compiled into the DLL

Simply renaming the DLL with Windows Explorer will not allow you to have two copies of the MemoPad (for example) on your RIM Wireless Handheld. The check is based on the internal DLL name. If for some reason you want to have to different versions of the same application on your device, you will need to adjust your build process.

Appendix

C library compatibility for RIM Wireless Handheld applications

Only some functions within the compiler C library are safe to call from the RIM Wireless Handheld environment. The following table summarizes the information. There are equivalent functions for the groups marked "No"—see the list afterwards.

Function Groups	Compatible with RIM Wireless Handheld?		
	Yes	No	Some
Argument access macros	✓		
Buffer access macros	✓		
Byte classification		✓	
Character classification		✓	
Data conversion			✓
Debug		✓	
Directory control		✓	
Error and exception handling		✓	
File handling		✓	
Floating point			✓
Input/output		✓	
Locale dependent		✓	
Memory allocation		✓	
Process and environment control		✓	
Searching and sorting	✓		
String manipulation		✓	
System calls		✓	
Time		✓	

Functions that are compatible

Argument access macros

This set of functions is compatible with the RIM Wireless Handheld environment. This includes the macros `va_arg`, `va_start`, and `va_end`.

Buffer manipulation functions

This set of functions is compatible with the RIM Wireless Handheld environment. This includes `memccpy`, `memchr`, `memcmp`, `memcpy`, `_memicmp`, `memmove`, `memset`, and `_swap`.

Data conversion functions

The following data conversion functions may be used in a RIM Wireless Handheld application: `abs`, `atoi`, `_atoi64`, `atol`, `_itoa`, `_i64toa`, `labs`, `_ltoa`, `strtol`, `strtoul`, `__toascii`, `tolower`, `_tolower`, `toupper`, `_toupper`, and `_ultoa`.

Searching and sorting functions

The library functions `bsearch`, `_lfind`, `_lsearch`, and `qsort` should work in the RIM Wireless Handheld environment. Testing for these functions has not been completed at this time.

Functions that are not compatible

Byte classification (multibyte) functions

The RIM Wireless Handheld does not support multibyte characters.

Character classification functions

Since the RIM Wireless Handheld does not support multibyte or wide characters, multibyte or wide character functions are also not supported. Many of the other isxxx() functions and macros are locale dependent. Different locales have different sets of upper and lower case characters.

Debug functions

The RIM Wireless Handheld does not support the compiler library debugging support functions. Other functions are provided to support debugging of wireless handheld applications. For example, short

debugging messages can be output to the debug output window when running MS Developer Studio by calling `RimDebugPrintf`.

Directory control functions

The RIM Wireless Handheld file system is different from those used on desktop systems. As a result, the directory control functions are not supported by the RIM Wireless Handheld.

Error and exception handling functions

The RIM Wireless Handheld environment does not support exception handling. Serious failures, such as page faults, are handled by the system function `RimCatastrophicFailure`, which may be called by application code when an unrecoverable error is detected.

File handling functions

The RIM Wireless Handheld file system is different from those used on desktop systems. As a result, the file handling functions are not supported by the RIM Wireless Handheld.

Floating-point functions

Since the RIM Wireless Handheld has no floating point co-processor and does not support application handling of traps and exceptions, floating point functions may not be used on the RIM Wireless Handheld, unless all of the floating points are implemented by emulation without calls to the operating system. However, the following functions are incorrectly classified as floating point functions, and may be used without difficulty: `div`, `labs`, `ldiv`, `_lrotl`, `_lrotr`, `_max`, `_min`, `rand`, `_rotl`, and `_rotr`.

Input and output functions

Any stream, file, or console input/output functions are not compatible with the RIM Wireless Handheld environment. The RIM Wireless Handheld environment provides different mechanisms for keyboard input, LCD output, and file system access. Because of the way they are implemented in the compiler library, this category also includes `sprintf` and `sscanf`. These may not be used in a RIM Wireless Handheld application. Use `RimSprintf` or `RimVSprintf` instead of `sprintf`.

The port I/O functions, such as `inp` and `outp`, will not cause an error during compilation, linking, or loading. However, because RIM Wireless

Handheld applications run in a protected environment, calling these functions will cause a protection fault at run time.

Locale dependent functions

The RIM Wireless Handheld environment does not support locales, multibyte characters, or wide characters. However, it is useful to have many of the locale dependent functions compatible. These functions were in the C library before the `setlocale` function.

Memory allocation functions

The standard C memory allocation functions may not be used in the RIM Wireless Handheld environment. Use `RimMalloc`, `RimRealloc`, and `RimFree` instead.

The global C++ operators `new` and `delete` may be used. They are translated into calls to `RimMalloc` and `RimFree` respectively. Operator new differs from the standard one in that, if there is insufficient memory to perform the allocation, it returns a `NULL` pointer rather than throwing an exception.

Process and environment control functions

Because the process model of the RIM Wireless Handheld environment is quite different from that of desktop systems, the compiler C library process and environment control functions are not applicable.

The functions `setjmp` and `longjmp` are also in this class. They are not available because the library implementations make system calls in order to perform stack unwinding. You could write a version of these functions that would work on the RIM Wireless Handheld, if it is acceptable not to call the distracters of C++ stack objects between the `setjmp` and the `longjmp`.

String manipulation functions

Many of the string manipulation functions are locale dependent or require the use of `malloc`. These functions are, therefore, not supported by the RIM Wireless Handheld environment.

System call functions

Windows system call functions may not be used in the RIM Wireless Handheld environment. Because of the way the RIM Wireless Handheld

operating system allocates application stacks, Windows functions are not safe to call even when running on the simulator.

Time functions

The RIM Wireless Handheld operating system represents time differently from desktop systems. As a result, time functions are not compatible with the RIM Wireless Handheld environment.

Index