MOTOROLA

FOMA M1000 SDK Users Guide

Version : 1.1

Date : 01-Aug-2005

Revision History

Revision #	Date	Description
0.1	15-Nov-2004	Initial draft based on A1000 SDK User Guide.
0.2	20-Jan-2005	Removal of Windows NT support
0.3	23-Jan-2005	Update information on using logical drives for emulator performance.
0.4	28-Jan-2005	Using “devices.exe” command from UIQ SDK and FOMA M1000 emulator information added.
0.5	11-Feb-2005	Updated J2ME documentation
0.6	11-Mar-2005	Updated J2ME documentation and Removed WinNT RAS support.
0.7	28-Mar-2005	Updated J2ME signed MIDlets path. Updated HAL data.
0.8	01-Jun-2005	Addition of J2ME classfiles location. Update of GDBStub section due to lack of serial cable. Updated HAL data.
1.0	01-Jul-2005	Disclaimer added in Introduction. Updated HAL data.
1.1	01-Aug-2005	Added Motorola USB GDBstub details. Update of Redirector section due to lack of serial cable. Removed references to Motorola Gaming API. Removed reference to JSR 135. Removed references to MOMAP1510 ABI Updated HAL data.

Contents

1. Introduction

1.1 Purpose

The purpose of this document is to provide a users guide for the FOMA M1000 SDK. Topics on setting up the development environment and issues with using the emulator are covered in this document.

1.2 Disclaimer

This SDK is intended to create applications for the FOMA M1000. Neither DoCoMo nor Motorola will be held responsible for problems caused by applications created for non-M1000 terminals.

If applications or any other products created by a developer using the FOMA M1000 SDK cause difficulties or problems to users of such applications or products or to third parties, the developer may be held legally responsible. Developers are to take sufficient care to avoid that possibility.

Some applications may be deleted by the security scan application installed in the FOMA M1000 depending on its content.

1.3 Target Audience

This document is intended to be utilized by application developers of the FOMA M1000.

Familiarity with Symbian OS 7.0, UIQ 2.0 or 2.1, and Java 2 Micro Edition is assumed and recommended.

1.4 Uninstallation

Special note: Cancellation of the uninstall is not supported for this SDK. Once the uninstall process is started, it cannot be cancelled. However, new files created or files that are not part of the FOMA M1000 SDK install will not be deleted.

2. Development Environment Configuration

2.1 Overview

The FOMA M1000 SDK can be used as a standalone environment or in conjunction with the Symbian 7.0 SDK for UIQ 2.0 or 2.1. The SDK is easier to use as a standalone environment, but this section will allow developers to use the Symbian 7.0 SDK for UIQ framework to work with the FOMA M1000 SDK.

2.2 FOMA M1000 SDK w/o Symbian 7.0 SDK for UIQ

If you do not have the Symbian 7.0 SDK for UIQ installed, then the FOMA M1000 SDK will require the following:

- ActiveState Perl
- Java Runtime Environment

Because the built-in applications use the same file paths as on the hardware, it is required to map the FOMA M1000 SDK environment to a logical drive for proper operation of the emulator. This is accomplished by using the “subst.exe” command.

```
subst Q: C:\Symbian\M1000SDK
```

Developers familiar with Symbian programming prior to Symbian 7.0 will recognize the use of the “subst.exe” command to keep multiple SDK installations separate. Since this logical drive disappears after rebooting the PC, you may wish to run the “subst.exe” command at Windows startup to ensure the logical drive is always ready when your PC boots up.

With the above environment, the following environment variables will need to be added:

```
EPOCROOT=\
PATH=\epoc32\gcc\bin;\epoc32\tools;%PATH%
```

With the above PATH and EPOCROOT environment settings, it will be necessary for all of the Symbian SDKs to be mapped to a logical drive with the \epoc32\ directory for the SDK located in the root of the drive.

2.3 FOMA M1000 SDK with Symbian 7.0 SDK for UIQ

2.3.1 device.exe Kit Management

If the Symbian 7.0 SDK for UIQ has been installed, managing separate SDKs is handled differently. While it is still required to run the FOMA M1000 SDK emulator in a logical drive through the use of the “subst.exe” command, the “devices.exe” command from the Symbian 7.0 SDK for UIQ automatically establishes the EPOCROOT and PATH settings.

With installation of the Symbian 7.0 SDK for UIQ, the path

```
c:\program files\common files\symbian\tools
```

has been added to the PATH of the PC. This directory contains several “stub” files which point to the \epoc32\tools and \epoc32\gcc\bin directories of the active “kit”. This scheme replaces the setting of the PATH and EPOCROOT environment variables described in the prior section.

The SDK kit assignment is handled by the “devices.exe” command. After installing the Symbian 7.0 SDK for UIQ, you may already have one or two UIQ SDK “kits” installed. To create an FOMA M1000 SDK kit, run the following from the command line after setting the FOMA M1000 SDK to a logical drive:

```
devices –add q:\ q:\ @M1000:com.motorola
```

assuming q:\ is the logical drive for the FOMA M1000 SDK.

Now that there is an FOMA M1000 SDK kit, the “stub” utilities in the c:\program files\common files\symbian\tools directory can point to the FOMA M1000 SDK \epoc32\tools directory in three ways.

- To always set the FOMA M1000 SDK as the default device environment, run the following from the command-line:

  ```
  devices –setdefault @M1000:com.motorola
  ```

- To set the FOMA M1000 SDK as the device environment for a MS-DOS window, run the following from the command-line:

  ```
  set EPOCDEVICE=M1000:com.motorola
  ```

- To only set the FOMA M1000 SDK as the device environment for a specific command, add the @device-id as a command argument. For example:

  ```
  bldmake bldfiles @M1000:com.motorola
  ```

It is important to stress that the use of devices.exe will not correct WINS emulator issues with file paths. Setting C:\Symbian\M1000SDK as a separate logical drive corrects those issues.

For more details on devices.exe, refer to the UIQ SDK documentation.

2.3.2 Unique FOMA M1000 Tools

As noted above, the Symbian 7.0 SDK adds the PC directory--C:\Program Files\Common Files\Symbian\Tools--to the PC's PATH environment variable.

The use of devices.exe from the UIQ SDK points the application stub files in the C:\Program Files\Common Files\Symbian\Tools directory to the \epoc32\tools directory assigned in the devices.exe syntax.

Although the Symbian 7.0 SDK for UIQ includes stub files for most GCC and generic Symbian tools, it lacks two tools that are unique to the FOMA M1000 SDK which need to be added to the C:\Program Files\Common Files\Symbian\Tools directory:

- signmidlet – for digital signing of J2ME MIDlets
- wsp – for Winsock operation

To ensure that these two tools are available for a FOMA M1000 SDK kit, copy the contents of C:\Symbian\M1000SDK\epoc32\stub\tools into the C:\Program Files\Common Files\Symbian\Tools directory.

2.4 Native Build Targets

Although the FOMA M1000 SDK supports two Application Binary Interfaces (ABIs)—ARMI and THUMB, most application developers should build for THUMB for device releases. ARMI libraries are included due to dependencies on some Motorola THUMB components.

2.5 J2ME Development

The FOMA M1000 device supports the following Java 2 Micro Edition APIs:

JSR 118 - MIDP 2.0
JSR 120 – Wireless Messaging API
JSR 82 – Java APIs for Bluetooth
JSR 185 – Java Technology for the Wireless Industry (JTWI)

These are all documented in the FOMA M1000 J2ME Developer Guide located at http://www.motocoders.com.

All classfiles used by the FOMA M1000 virtual machine are located in the C:\Symbian\M1000SDK\j2me\classfiles.zip archive. These include standard Sun Java APIs and also custom Motorola APIs.

The FOMA M1000 SDK does not contain tools or compilers for J2ME development. Motorola recommends the following development environment for J2ME development (all are available as a free download from Sun or from Motorola).

- Application Development: J2SE SDK v1.4.2 (http://java.sun.com/j2se/1.4.2/download.html)

- Application Development: J2ME Wireless Toolkit 2.2 (http://java.sun.com/products/j2mewtoolkit/index.html)

- Device emulation: FOMA M1000 SDK (http://www.motocoders.com)

- Device loading and testing: Motorola Desktop Suite (http://www.motocoders.com)

Sample code and technical papers for J2ME and MIDP 2.0 can be found at various locations on the internet.

The DemoVaders MIDlet located at Symbian's Developer Network (http://www.symbian.com/developer/techlib/papers/java_MIDP.asp) is a good demonstration of MIDP 2.0 (look for the section "Game development across the MIDP versions").

Another useful sample MIDlet is the PlayAudio MIDlet located at http://developer.sonyericsson.com/site/global/techsupport/tipstrickscode/java/p_playaudiomidp2.0.jsp but please note that the iMelody format used in the MIDlet is not supported by the FOMA M1000.

3. Motorola APIs

3.1 Overview

This section briefly describes the various Motorola APIs supported in the FOMA M1000 handset supported by the SDK.

For Symbian 7.0 and UIQ 2.0 and 2.1 APIs and usage, please consult the Symbian 7.0 SDK for UIQ 2.0 or 2.1 documentation.

3.2 FOMA M1000 C++ Modifications to UIQ

Motorola has made some changes to a number of UIQ classes. The modifications are a result of bug fixes, function enhancements, or simply a different version of the UIQ component.

Documentation pointing out the changes from FOMA M1000 modifications to the UIQ implementation is described in the following SDK HTML:

```
C:\Symbian\M1000SDK\docs\UIQModified\index.html
```

3.3 C++/EPOC APIs

Documentation of the contents and usage of the following APIs are located in the SDK directory: c:\Symbian\M1000SDK\docs.

3.3.1 Audio

In addition to the Media Server audio interfaces from the UIQ SDK, the FOMA M1000 handset uses a Motorola Audio API for enhanced sound generation and audio playback. Note that audio playback is not supported on the WINS emulator so testing of applications using the Audio API must be performed on the FOMA M1000 device.

3.3.2 Camera Capture

The FOMA M1000 handset uses Camera Directed Navigation Links (DNLs) to allow developers to take image snapshots with the device’s built-in digital camera. The WINS emulator does not support the Camera Capture API, so applications using it will need to be tested on the target hardware.

3.3.3 Power Server

The FOMA M1000 handset uses the Power Server API for managing power-based conditions and events. The WINS emulator does not support the Video Playback API, so applications using video will need to be tested on the handset.

3.3.4 Video Playback

The FOMA M1000 handset uses the Video Playback API for playback of MPEG-4 video content. The WINS emulator does not support the Video Playback API, so applications using video will need to be tested on the handset.

4. FOMA M1000 Key Mappings

4.1 Overview

In addition to the touchscreen, the FOMA M1000 device has 15 separate hardware buttons. The FOMA M1000 buttons are assigned to specific scancodes and keycodes which are detailed here.

For details on the usage of scancodes and keycodes please consult the UIQ SDK documentation.

4.2 Restrictions

Most device buttons are assigned to specific applications and are unavailable for use by developers. The following keys are “consumed” by other applications:

- Send – consumed by Phone application
- End – consumed by Phone application
- Browser/Shortcut – launches Browser app or Shortcut app (if held down)
- Speakerphone – toggles speakerphone functionality
- Volume Up/Down – consumed by audio drivers and by Camera application for zoom
- Voice Tag – consumed by Contacts application
- Camera – consumed by Camera application

4.3 Available Keys

The buttons that are accessible for developer use are as follows:

- Up – Scancode: EStdKeyDevice4, Keycode: EQuartzKeyFourWayUp
- Down - Scancode: EStdKeyDevice5, Keycode: EQuartzKeyFourWayDown
- Left - Scancode: EStdKeyDevice6, Keycode: EQuartzKeyFourWayLeft
- Right - Scancode: EStdKeyDevice7, Keycode: EQuartzKeyFourWayRight
- Select - Scancode: EStdKeyDevice8, Keycode: EQuartzKeyConfirm
- Game A - Scancode: EStdKeyApplicationA
- Game B - Scancode: EStdKeyApplicationB

The scancodes and keycodes above can be referenced in the e32keys.h and QuartzKeys.h header files.

Note: The FOMA M1000 joystick is an 8-way switch. When a diagonal direction is pushed, the joystick hardware actually generates two separate keycodes rather than an individual unique keycode. The emulator currently cannot emulate this hardware behavior.

5. Miscellaneous

5.1 Handset IMEI

Some developers wish to use the handset’s IMEI number as a unique identifier.

The FOMA M1000 SDK supports the reading of the device IMEI value through Symbian’s PLP interfaces.

Located in the PLPVARIANT.H header file, the function:

```
void GetMachineIdL(TPlpVariantMachineId &aId)
```

will return the device IMEI number in the aId variable.

5.2 RTimers and FOMA M1000

Although RTimers are described in the UIQ documentation, usage of the RTimers in the FOMA M1000 handset requires additional consideration.

The power management for the FOMA M1000 handset is particularly aggressive when it comes to saving handset battery life. Therefore, RTimer events on the FOMA M1000 behave differently than described in the UIQ documentation in the following ways:

- RTimer::At() events will not trigger if the FOMA M1000 is off.
- RTimer::After() handlers are vulnerable to the handset entering “low-power mode” shortly after the timer event.

If an application requires the FOMA M1000 to stay in “run mode” to perform activities after a timer event (e.g. scheduled pull application for content), usage of the Power Server API is needed.

By using the InformConditionTypeStart(EMConditionTypeEmail) function in the Power Server API, the application can manually force the handset into “run mode” until running InformConditionTypeStop(EMConditionTypeEmail).

Note: Incorrect usage of the InformConditionTypeStart() and InformConditionTypeStop() can dramatically reduce the battery life of the FOMA M1000 handset. Developers must avoid circumstances where the FOMA M1000 is forced into “run mode” but is not returned to “low-power mode” after normal operations or error handling.

6. GDB On-Device Debugging for EPOC

6.1 Overview

The FOMA M1000 device supports on-device debugging of C++/EPOC applications with the help of the GNU Debugger (GDB). GDB runs on a host computer that is connected to the device via USB or serial interface. A “stub” program running on the device provides the debugging services to GDB.

For details on setting up your project for GDB please consult the UIQ SDK documentation.

For general usage and documentation on the GNU Debugger, please consult the GDB website at http://www.gnu.org/software/gdb/gdb.html.

Refer to 10.5Appendix C - Example GDB session on PC for an example of using GDB with the BasicApp sample application from the UIQ 2.1 SDK.

6.2 Restrictions

The GDBstub.exe, GDBseal.dll, and GDBseng.dll files on the FOMA M1000 ROM (the Z: drive) only support on-device debugging over the serial port and will not work with the USB cable. For debugging with the USB port, the installable version of GDB provided in the SDK directory \epoc32\tools must be used.

Since GDB on the PC is limited to COM ports 1 through 4, the virtual COM port assigned by Windows when plugging in the USB cable needs to fall in that range. If Windows has assigned a virtual COM port outside of COM ports 1 through 4, then it will be necessary to free up Modem devices (via Start->Settings->Control Panel->Phone and Modem options).

Since GDB requires a free COM port, Windows PCs with Symbian Connect/Desktop Suite installed will need to disable the COM port in mRouter to avoid blocking GDB from using the port.

Manually running GDBstub.exe on the FOMA M1000 requires access to the file system. Therefore, a file manager like QFileman is required to manually run the stub.

After quitting installable version of GDBstub (entering the “q” character twice using the virtual keyboard), if the developer wishes to use the FOMA M1000 as a modem, it will be necessary to unplug and reattach the USB cable.

6.3 Running GDBstub.exe on the FOMA M1000

Developers wishing to use a USB cable for on-device debugging must follow the below steps to run the GDB stub program.

Precondition: Install the GDBstub_thumb.sis application located in the SDK directory c:\Symbian\M1000SDK\epoc32\tools\ in the FOMA M1000 internal drive (C: drive).

1. Ensure the virtual COM Port assigned to the USB port is free and unused by Desktop Suite or mRouter (unchecked).

2. Launch File Manager (qfileman) on the device.

3. Goto C:\System\Programs directory on the device.

4. Click on gdbstub.exe from the device File Manager.

5. Run GDB on the PC to monitor the virtual COM port (see Appendix C for an example session).

An example project—DebugBasicApp—has been made available in the directory:

```
C:\Symbian\M1000SDK\docs\examples
```

The File Manager (qfileman) can be downloaded from Symbian's developer website:

http://www.symbian.com/developer/downloads/tools.html

7. Redirector On-Device Debugging for Java

7.1 Overview

The FOMA M1000 device supports on-device debugging of J2ME applications with the help of the Redirector application. Redirector takes over all three of Java's standard I/O streams—System.out, System.in, and System.err—and provides you with the means to redirect these streams to a console window. In addition it also allows you to redirect output to a log file. Usage of the serial port output is not supported.

Two versions of the Redirector are provided for developers. One is for the WINS target environment and is already bundled on the WINS emulator. The other is built for the THUMB target environment and can be installed on FOMA M1000 hardware. The Redirect_thumb.SIS application is located in the C:\Symbian\M1000SDK\epoc32\tools directory.

For general information on the Redirector tool please consult the UIQ SDK documentation.

7.2 Restrictions

Developers cannot use Redirector to output standard I/O over the serial port.

Note: Installing the redirect_thumb.sis will result in an incompatible warning. This can be ignored for the FOMA M1000 because the tool did not use UIQ 2.0 or FOMA M1000 IDs for product/platform version compatibility during packaging (see 9.3.1 Product/Platform Version Compatibility).

7.3 Starting Redirector on the FOMA M1000

Running Redirector with Java applications for console or file output on the FOMA M1000 is described in the UIQ SDK documentation.

8. Winsock

8.1 Overview

To aid developers in testing internet-aware applications, the FOMA M1000 SDK emulator provides a mechanism for accessing the socket connection of the development PC using a Winsock connection.

8.2 Restrictions

The Winsock implementation for the WINS emulator does not support proxies used by the development PC’s network connection.

The Winsock implementation also disables Windows RAS operations when enabled.

8.3 Usage

The Winsock tool is located in the \epoc32\tools directory of the FOMA M1000 SDK.

8.3.1 Status

To check whether the Winsock component is enabled or disabled for the WINS emulator, run the following command:

```
wsp s
```

Note: the emulator has the Winsock implementation enabled by default.

8.3.2 Enable

To enable the Winsock implementation for the WINS emulator, run the following command:

```
wsp e
```

8.3.3 Disable

To disable the Winsock implementation for the WINS emulator, run the following command:

```
wsp d
```

9. Digitally Signed Applications

9.1 Overview

The FOMA M1000 supports digitally signed applications, including J2ME MIDlets. This section outlines issues with signing applications—EPOC or J2ME—with digital certificates.

For details on digitally signing a SIS file, see the UIQ SDK documentation.

9.2 Restrictions

Only J2ME MIDlets signed with a RSA-based key are supported in the FOMA M1000 secure installer. J2ME MIDlets signed with a key not based on RSA will be treated as an unsigned MIDlet.

9.3 SIS PKG Files

9.3.1 Product/Platform Version Compatibility

The FOMA M1000 supports the product/platform version compatibility feature for SIS file packages. The UID should identify the earliest possible platform version or product to maximize the number of phones the package can be installed on.

As a UIQ 2.0 device, the FOMA M1000 supports the UIQ 2.0 Platform ID. Newer versions of the FOMA M1000 may also support the Motorola A920 Platform ID. The FOMA M1000 does not have the UIQ 2.1 platform ID.

For generic UIQ 2.0 applications, it is recommended to use the UIQ 2.0 ID by inserting the following line into the PKG file:

```
(0x101F617B), 2, 0, 0, {"UIQ20ProductID"}
```

For FOMA M1000 applications where unique Motorola APIs are used, it is recommended to use the Motorola M1000 Platform ID by inserting the following line into the PKG file:

```
(0x10207EA2), 1, 0, 0, {"MotorolaM1000ProductID"}
```

If your PKG file supports more than one language, the dependency string will need to be defined multiple times. For example, for a PKG file with support for three languages, the following line should be used:

```
(0x101F617B), 2, 0, 0, {"UIQ20ProductID", "UIQ20ProductID",
"UIQ20ProductID"}
```

9.3.2 Condition Block Usage

The UIQ SDK documentation on PKG file format details the use of condition blocks for controlling the installer based on device attributes in the \epoc32\include\hal_data.h file.

Refer to Appendix B - HALData Attributes for the HALData attribute settings for the FOMA M1000.

9.4 J2ME Signing Procedure

The procedure for digitally signing and installing a J2ME MIDlet is as follows:

1. Identify the location of the MIDlet JAR and JAD files.

2. Identify the location of the secure key and certificate on the PC hard drive.

3. Run the “signmidlet” tool in the C:\Symbian\M1000SDK\epoc32 with the following syntax:

   ```
   signmidlet.exe -k <.key file> -s <.cer file> -d <.jad file> <.jar file>
   ```

   where <.key file> is the path and filename of the private key file, <.cer file> is the path and filename of the public key certificate file, <.jad file> is the path and filename of the J2ME JAD file, and <.jar file> is the path and filename of the J2ME JAR file.

Note that digital keys and certificates are not provided by Motorola.

10. Emulator Application Install

10.1 Overview

For development testing, applications can be installed on the FOMA M1000 SDK emulator in a number of different ways. This section outlines the different options available to the developer.

10.2 C++/EPOC

10.2.1 WINS UDEB Build Target

By building on a WINS UDEB build target, the developer automatically inserts compiled applications into the SDK emulator environment. The next time the emulator is run, the App Launcher will automatically find the application and display it in the main menu.

10.2.2 Copying Files Into WINS UDEB Directory

If the developer has compiled files, manually copying the compiled files into PC directory

```
\epoc32\release\wins\udeb\z\system\apps\<app name>\
```

will “install” the application into the emulator.

Like building in the WINS UDEB build target, the App Launcher will automatically find the app and display it in the main menu.

10.2.3 Using Emulator App Installer

If the developer wishes to test the integrity of a packaged SIS file, copy the SIS file into \epoc32\wins\c\ and run the App Installer from the App Launcher->Launcher->Install on the WINS emulator.

Note that using the App Installer to uninstall the app is also possible with this implementation.

10.3 J2ME MIDlets

Unlike C++/EPOC applications, J2ME MIDlets cannot be copied into the emulator directories directly. Therefore, to install J2ME MIDlets, use the same steps outlined in 10.2.3 Using Emulator App Installer.

10.4 Simulating External Flash

As described in the UIQ 2.1 SDK documentation, adding the line

```
_EPOC_DRIVE_D <absolute path>
```

to the epoc.ini file in the \epoc32\data directory will simulate the external flash card at the location defined in the <absolute path>.

Note that the name of the drive that will appear in the Application Installer on the emulator will be the volume name of the drive in the path which may be the same volume name as the C:\ drive.

10.5 Language Considerations

Because of the FOMA M1000 target market in Japan, the FOMA M1000 WINS emulator supports both Japanese and English.

Because of licensing issues, the Advanced Wnn Japanese conversion system by Omron Software is not supported on the emulator. Instead, a partial implementation is used providing some of the look-and-feel of the actual system on the handset and allowing developers to enter Japanese characters into device memory.

Prompts for Motorola applications can be switched from English or Japanese. To change the language setting on the emulator, run the device Control Panel. On the General tab is a Language Selection option. After changing the language option, the emulator must be closed and restarted for the language change to take effect.

Appendix A - Remote Access Service PC Setup

Overview

Windows 2000 comes with internet access software which can be setup to use a PPP connection between the WINS emulator and the Remote Access Service (RAS) server.

The UIQ SDK documentation details how to configure the SDK emulator to use RAS. This section details the steps on setting up the RAS service on a Windows 2000 PC.

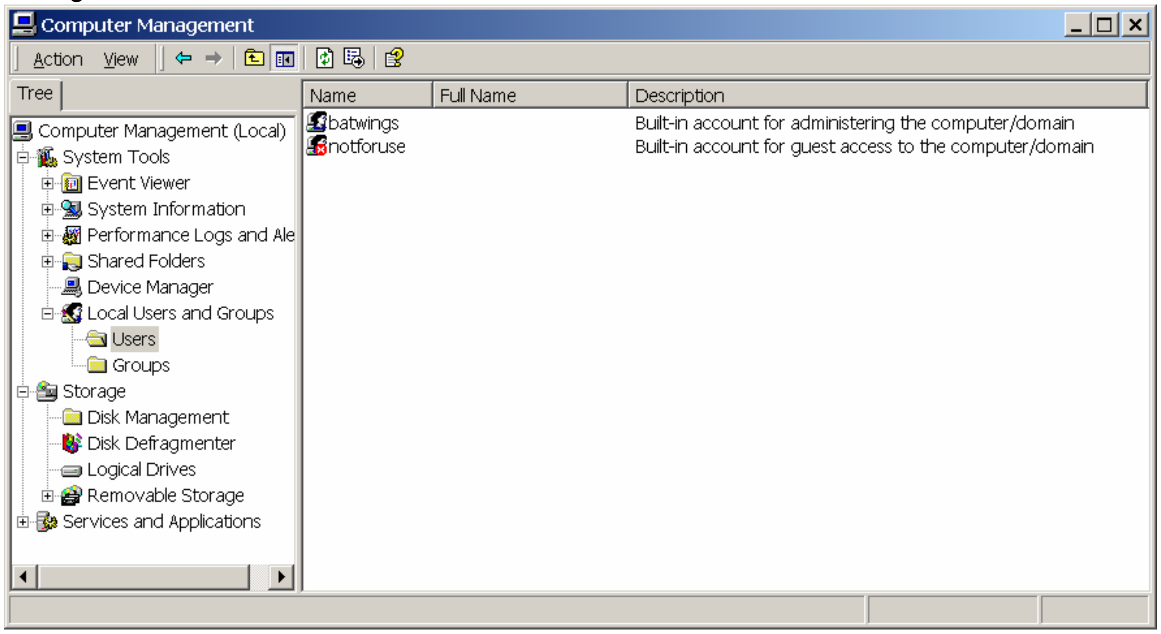
Restrictions

RAS will not work with Winsock support enabled.

Note that the FOMA M1000 emulator uses the PC COM1 port. The following steps assume you are using the PC’s COM2 port for the RAS server. If you are using a PC with only one COM port, you cannot use the same machine as both the emulator and the RAS server.

Windows 2000 Setup

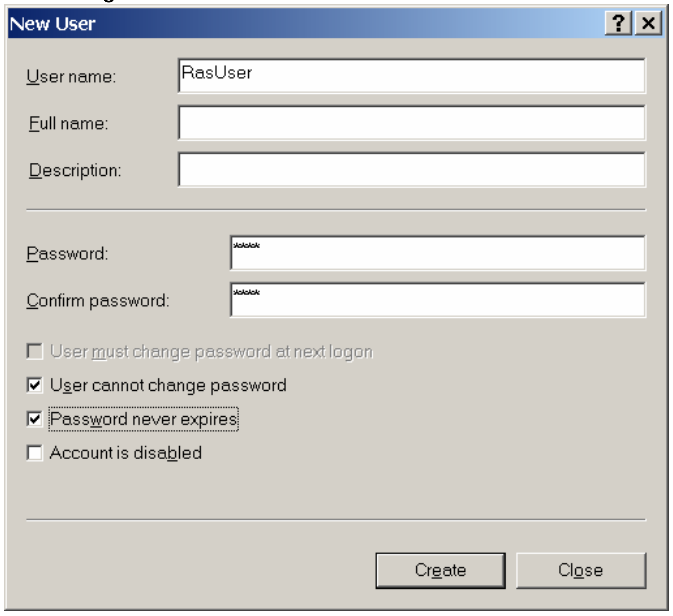
1. From the **Start** menu, choose **Programs**, **Administrative Tools**, and select **Computer Management**.
   As an alternative, you can select **Settings, Control Panel**, and **Administrative Tools** from the **Start** menu and then select **Computer Management.**



Name	Full Name	Description
batwings		Built-in account for administering the computer/domain
notforuse		Built-in account for guest access to the computer/domain

2. In the left-hand panel of the **Computer Management** dialog box, expand “Local Users” and “Groups”, and then select “Users”. Right-click in the panel and select “Add New User” to create a new account. Create account with an account name of “RasUser” and a password of “pass”. This corresponds to a preexisting internet account in the FOMA M1000 emulator called “NT RAS”.

3. Double-click on the “RasUser” account to check the settings. Under “General”, the boxes labeled “User cannot change password” and “Password never expires” boxes should be checked, and the box labeled “Account is disabled” should be unchecked. Under “Member Of”, the account should be a member of “Guests”. Click “OK” to save the settings.



New User

User name: RasUser

Full name:

Description:

Password: ****

Confirm password: ****

☐ User must change password at next logon

☑ User cannot change password

☑ Password never expires

☐ Account is disabled

4. From the **Start** menu, choose **Settings, Control Panel**, and select **Phone and Modem Options**. Click on the “Modems” tab. If “Communications cable between two computers” is installed on COM1, then remove that modem. If “Communications cable between two computers” is installed on a COM2, then skip to step 6.

5. Click “Add…” to open the modem wizard. In the next page, check the box labeled “Don’t detect my modem” and then click “Next”. In the following page, click on “Communications cable between two computers”, and then click “Next”. In the following page, select the serial port that you want to use for RAS connections. Now, click “Next” followed by “Finish” to close the wizard.

6. Right-click on “Communications cable between two computers”, and then select “Properties”. Select a COM2 as the serial port. Set the maximum port speed to 115200 baud, and then click “OK” twice to save the new settings.

7. From the Start menu, choose **Settings, Network and Dial-up Connections**, and select **Make New Connection** to start the Network Connection Wizard. In the first page of the wizard, click “Next”. In the second page, select “Accept incoming connections”, and then click “Next”. In the third page, select “Communications cable between two computers”, and then click “Next”. In the fourth page, De-select “Do not allow virtual private connections”, and then click “Next”. In the fifth page, make sure that the box labeled “guest” is checked, and then click “Next”.

8. In the next page, select “Internet Protocol (TCP/ IP)”. Double-click on “TCP/IP” and make sure that “specify TCP/IP address” checked. Then enter a range of IP addresses that will be used by the RAS server and the devices connecting the server. (e.g. 10.0.0.1 to 10.0.0.2 will assign 10.0.0.1 to the server and 10.0.0.2 will get assigned to one device when Ras is started, Use this server address in “hosts” file) Then, click “OK”, click “Next”, and then click “Finish” to exit the wizard.

Starting RAS Service

To start the RAS service, at the command line prompt, enter the following:

```
NET START "REMOTE ACCESS CONNECTION MANAGER"
NET START "REMOTE ACCESS SERVER"
NET START "SIMPLE TCP/IP SERVICES"
NET START EVENTLOG
START RASMON
```

It would be best to save these commands in a suitable batch file.

RAS should now be running and should be listening to COM2. Enter the command ‘NET START ’; this should indicate that RAS is running.

When you connect a computer to your PC’s COM2 port, the RASMON program should display a ‘CD’ light when you make a connection. You can choose options in the RASMON program which will display data-transmitted/received activity indicators.

Troubleshooting

- Check that RAS is running by typing NET START at the command line prompt.
- Run the RASMON program to monitor activity on the port.
- Check the following points in the Modems icon in the Control Panel.

1. Click the **Connection** tab

2. Click Connection

   - Data bits: 8

   - Parity: None

   - Stop bits: 1

3. Click Advanced
   In case you use PCMCIA ADAPTER:

   - Check Use flow control

   - Check Hardware (RTS/CTS)

4. Un-Check Use flow control

Appendix B - HALData Attributes

This information is provided to the developer for condition block usage in PKG files. The \epoc32\include\hal_data.h header file may need to be consulted to identify the appropriate enum or value.

Note however, several of these values set in the FOMA M1000 software may be incorrect. For example, ECPUABI = armi even though Motorola recommends developers use native build target for FOMA M1000 applications is THUMB.

The following are values for the FOMA M1000 (as of the build in the SDK).

EManufacturer	=	Motorola
EManufacturerHardwareRev	=	0x3
EManufacturerSoftwareRev	=	0x001
EManufacturerSoftwareBuild	=	0x97f40
EModel	=	0x1020423F
EMachineUid	=	ParagonEuropean
EDeviceFamily	=	quartz
EDeviceFamilyRev	=	0x001
ECPU	=	arm
ECPUArch	=	0x400
ECPUABI	=	armi
ECPUSpeed	=	172032
ESystemStartupReason	=	cold
ESystemException	=	0
ESystemTickPeriod	=	15625
EMemoryRAM	=	4
EMemoryRAMFree	=	4
EMemoryROM	=	12
EMemoryPageSize	=	0x1000
EPowerGood	=	1
EPowerBatteryStatus	=	good
EPowerBackup	=	1
EPowerBackupStatus	=	good
EPowerExternal	=	1
EKeyboard	=	full
EKeyboardDeviceKeys	=	0
EKeyboardAppKeys	=	0
EKeyboardClick	=	1
EKeyboardClickState	=	1
EKeyboardClickVolume	=	0
EKeyboardClickVolumeMax	=	1
EPen	=	1
EPenX	=	208
EPenY	=	320
EPenDisplayOn	=	0
EPenClick	=	1
EPenClickState	=	1
EPenClickVolume	=	0
EPenClickVolumeMax	=	1

```
EMouse                        = 0
ECaseSwitch                   = 0
ELEDs                         = 0
EIntegratedPhone              = 0
ESystemDrive                  = 0x2
EDisplayXPixels               = 0
EDisplayYPixels               = 0
EDisplayXTwips                = 0
EDisplayYTwips                = 0
EDisplayColors                = 65536
EDisplayState                 = 1
EDisplayContrast              = 0
EDisplayContrastMax           = 0
EBacklight                    = 0
EBacklightState               = 0
EDisplayIsMono                = 0
EDisplayIsPalettized          = 0
EDisplayBitsPerPixel          = 0
EDisplayNumModes              = 0
EDisplayMemoryAddress         = 0
EDisplayOffsetToFirstPixel    = 0
EDisplayOffsetBetweenLines    = 0
EDisplayPaletteEntry          = 0
EDisplayIsPixelOrderRGB       = 0
EDisplayIsPixelOrderLandscape = 0
EDisplayMode                  = 0
EDisplayBrightness            = 0
EDisplayBrightnessMax         = 0
EDebugPort                    = 0
ELocaleLoaded                 = 0
ELanguageIndex                = 1
EEnableTouchScreen            = 0
EDisableTouchScreen           = 0
EDigitiserSwitchOn            = 0
EDigitiserSwitchOff           = 0
```

Appendix C - Example GDB session on PC

Motorola has provided a debug BasicApp example in the SDK directory:

C:\Symbian\M1000SDK\docs\examples\

Build parameters have been modified from the UIQ 2.1 SDK example to demonstrate the changes needed to support GDB on-device debugging. In the BasicApp project, the gdb.ini presumes the SDK environment has been subst'ed to a logical drive of Q: and that Microsoft Windows has assigned a virtual COM 4 port for the USB connection.

The following steps show how GDB can be used to debug the BasicApp example.

Note: It is recommended to use **gdb -nw** as a command (instead of **gdb**) because the GUI front-end *Insight* used by the graphical **gdb** is considered unreliable.

```
Q:\BasicApp>gdb -nw
GNU gdb 4.17-psion-98r2
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This version of GDB has been modified by Symbian Ltd. to add EPOC support.
Type "show epoc-version" to see the EPOC-specific version number.
This GDB was configured as "--host=i686-pc-cygwin32 --target=arm-epoc-pe".
Breakpoint 1 at 0x10001010: file .\\Basicapp.cpp, line 13.
(gdb)
(gdb) run
Starting program:
Breakpoint 1 at 0xff700010: file .\\Basicapp.cpp, line 13.
warning: Application started but no document specified.
The application may panic at some point if it is document based
and if there is no existing default document.
```

Note: GDB is case sensitive to file names. Use **info sources** to find out what it has "decided" that the file name is. Also note **.\\**

```
Breakpoint 1, NewApplication () at .\\Basicapp.cpp:13
13              return new CQBasicApp;
Current language:  auto; currently c++
(gdb) info breakpoints
Num Type           Disp Enb Address    What
1   breakpoint     keep y   0xff700010 in NewApplication(void)
                                       at .\\Basicapp.cpp:13
        breakpoint already hit 1 time
(gdb) break .\\Cqbasicappui.cpp:27
Breakpoint 2 at 0xff700fc0: file .\\Cqbasicappui.cpp, line 27.
```

```
(gdb) break .\\Cqbasicappui.cpp:99
Breakpoint 3 at 0xff70129c: file .\\Cqbasicappui.cpp, line 99.
(gdb) info breakpoints
Num Type           Disp Enb Address    What
1   breakpoint     keep y   0xff700010 in NewApplication(void)
                                       at .\\Basicapp.cpp:13
        breakpoint already hit 1 time
2   breakpoint     keep y   0xff700fc0 in CQBasicAppUi::ConstructL(void)
                                       at .\\Cqbasicappui.cpp:27
3   breakpoint     keep y   0xff70129c in CQBasicAppUi::HandleCommandL(int)
                                       at .\\Cqbasicappui.cpp:99
(gdb) continue
Continuing.

Breakpoint 2, CQBasicAppUi::ConstructL (this=0x4089c8)
    at .\\Cqbasicappui.cpp:27
27              CQikAppUi::ConstructL();
(gdb) next
30              iCategoryModel = QikCategoryUtils::ConstructCategoriesLC(R_BAPP_
DEFAULT_CATEGORIES);
(gdb) next
31              CleanupStack::Pop(); // iCategoryModel
(gdb) print iCategoryModel
$1 = (CQikCategoryModel *) 0x408f54
(gdb) cont
Continuing.

Note: User input on BasicApp on the device will cause the breakpoint to be hit
again.

Breakpoint 3, CQBasicAppUi::HandleCommandL (this=0x4089c8, aCommand=1)
    at .\\Cqbasicappui.cpp:104
104                             SwitchViewL(KUidQBasicAppGraphicsListView,*iGrap
hicsListView);
(gdb) list
99                      {
100                     // Handle graphics view button, and details view's go ba
ck button
101                     case EReturnToListView:
102                     case ESwitchToGraphicsView:
103                 {
104                             SwitchViewL(KUidQBasicAppGraphicsListView,*iGrap
hicsListView);
105                             break;
106                             }
107                     // Handle text view button
108                     case ESwitchToTextView:
(gdb) next
105                             break;
(gdb) cont
Continuing.

Note: User input on BasicApp on the device will cause the breakpoint to be hit
again.

Breakpoint 3, CQBasicAppUi::HandleCommandL (this=0x4089c8, aCommand=1)
```

```
    at .\\Cqbasicappui.cpp:104
104                             SwitchViewL(KUidQBasicAppGraphicsListView,*iGrap
hicsListView);
(gdb) kill
Kill the program being debugged? (y or n) y
(gdb)
```