NOKIA M2M PLATFORM APPLICATION DEVELOPMENT KIT

EVALUATION MODULE USER GUIDE

NOKIA

Copyright © 2003 Nokia. All rights reserved. Issue 1.0 9356525

Contents

Legal Notice

Copyright © 2003 Nokia. All rights reserved.

Reproduction, transfer, distribution or storage of part or all of the contents in this document in any form without the prior written permission of Nokia is prohibited.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Nokia operates a policy of continuous development. Nokia reserves the right to make changes and improvements to any of the products described in this document without prior notice.

Under no circumstances shall Nokia be responsible for any loss of data or income or any special, incidental, consequential or indirect damages howsoever caused.

The contents of this document are provided "as is". Except as required by applicable law, no warranties of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose, are made in relation to the accuracy, reliability or contents of this document. Nokia reserves the right to revise this document or withdraw it at any time without prior notice.

1. ACRONYMS AND TERMS

Acronym/term	Description
AC	Alternating Current
ADK	Application Development Kit
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSD	Circuit Switched Data
EEPROM	Electrically Erasable Programmable Read-Only Memory
EM	Evaluation Module
EMCA	Evaluation Module Controller Application
GIOP	General Inter-ORB Protocol
GSM	Global System for Mobile Communication
HW	Hardware
I/O	Input/Output
IDL	Interface Definition Language
LED	Light Emitting Diode
M2M	Machine-to-Machine
MCU	Micro Controller Unit
MSB	Most Significant Bit
ORB	Object Request Broker
OS	Operating System
PC	Personal Computer
RAM	Random Access Memory
RTOS	Real-Time Operating System
SCI	Serial Communications Interface

2. INTRODUCTION

2.1 WHAT IS EVALUATION MODULE

The Evaluation Module (EM) is a part of Nokia Machine-to-Machine (M2M) application development environment. The EM is an embedded device that has both digital and analog input and output connectors as illustrated in Figure 1. The states of these connectors can be controlled remotely from a computer application called Evaluation Module Control Application (EMCA).

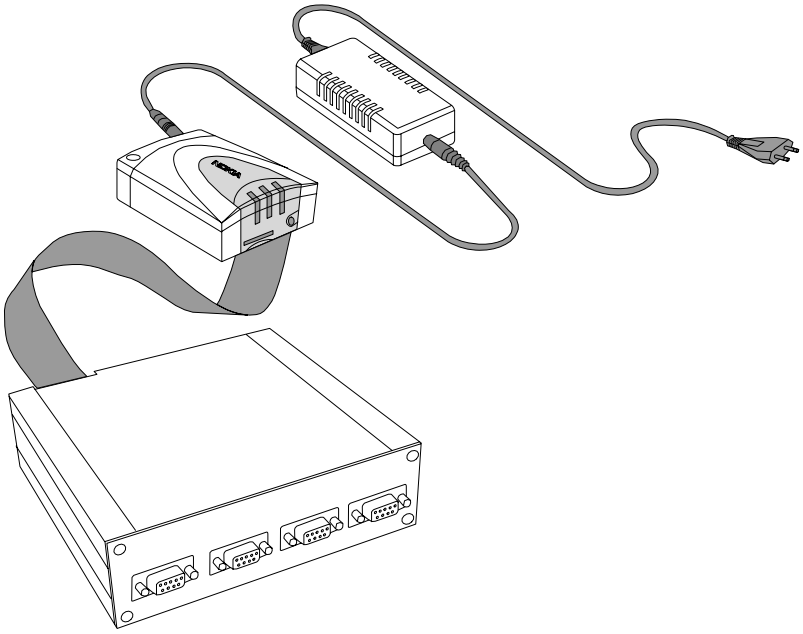


Figure 1. The Evaluation Module

The Evaluation Module runs a real time software capable of communicating with external devices through electrical interfaces. An example interface is a digital output pin, which can have a status of either logical one (3.3 V) or zero (0 V). The output pin can be connected to an existing device for controlling purposes.

The Java™ based Evaluation Module Control Application runs on a personal computer (Figure 2). This server application can communicate with the Evaluation Module by using the Nokia M2M Platform components. The control application can control remotely all the inputs and outputs of the Evaluation Module. As a simple example application it can control only one Evaluation Module at a time.

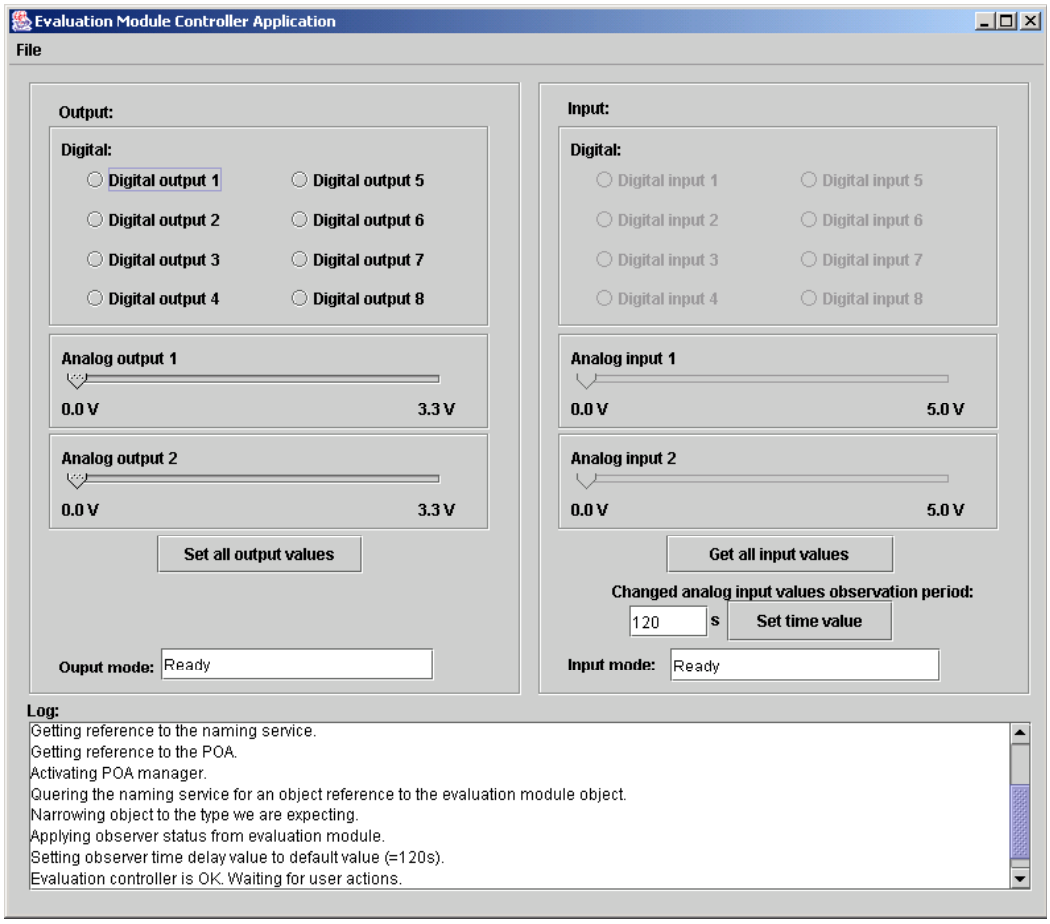


Figure 2. The Evaluation Module Control Application.

The Evaluation Module with its control application forms an example implementation of a machine communication application built on Nokia M2M Platform.

2.2 WHERE TO USE EVALUATION MODULE

The Evaluation Module can be used for several purposes, including controlling existing devices and for learning M2M application development with Nokia M2M Platform. Since the EM runs on an embedded environment and can be easily re-programmed, embedded software can be easily tested in real hardware environment.

The EM application can also be a testing tool when the Nokia M2M Platform is built up.

The source code of both embedded and server side application are freely available and can be used as references when designing other applications for Nokia M2M Platform.

2.3 EVALUATION MODULE I/O INTERFACES

The EM has the following I/O interfaces:

- 8 digital inputs
- 2 analog inputs
- 8 digital outputs
- 2 analog outputs

2.3.1 DIGITAL INPUTS

The Evaluation Module has 8 digital inputs whose logical state ('0' or '1') can be indicated. Logical '0' stands for 0V and logical '1' for 3.3V. To get the exact logical level characteristics, please see the hardware documents of the Evaluation Module.

The logical state of the inputs is read to RAM database periodically. The refresh period of the RAM database is 100 ms. This means that no shorter pulses than 200 ms can be detected. The hardware also supports interrupt-controlled input processing, but it is not supported by the software. This is because it is desirable to keep the software architecture of the Evaluation Module as simple as possible.

The server application can query the current state of any individual input or any combination of up to 8 inputs from the RAM database at any time. The server application can register its ORB to get indication immediately if any state transition is detected in the input lines. All inputs have an internal pull-up resistor.

2.3.2 ANALOG INPUTS

The Evaluation Module has two analog inputs whose input voltage can be indicated by the A/D converter. The input voltage must be within the range of 0.0mV to 5000mV. The internal A/D converter value 0 (dec) stands for 0.0mV and 1023 (dec) for 5000mV. The tolerance of the A/D converter is 10 bits (approx. 3.2mV). For the exact analog input characteristics, please see the hardware documents of the Evaluation Module.

The input voltage samples are read to the RAM database periodically. Both the last measured sample and the average value of the last ten samples are stored on the RAM database for both analog inputs. The refresh period of the RAM database is 100 ms. It means that pulses shorter than 200 ms cannot be detected.

There are no compensation functions for the offset error and non-linearity error inside the EM. The software of the Evaluation Module can also be configured to send the average input voltage to the server application periodically, if the

voltage has changed since the last sent voltage. The default value for the indication period is two minutes.

	**Note:** Because the remote control interface specified for analog inputs by the IDL utilises direct mV-units, the server application does not need to take care of the scaling.

2.3.3 DIGITAL OUTPUTS

The Evaluation Module has 8 digital outputs which can be set either to the logical state '0' or '1'. Logical '0' stands for 0V and logical '1' for 3.3V. To get the exact logical level characteristics, please see the hardware documents of the Evaluation Module.

The logical state of the outputs is refreshed from the RAM database to the output port periodically. The server application can set the new state to the RAM database for any individual output or any combination of up to 8 outputs at any time. The refresh period of the digital outputs is 100 ms. This means that pulses shorter than 200 ms cannot be generated.

After the start-up self tests the initial state of the digital outputs is logical '0'.

2.3.4 ANALOG OUTPUTS

The Evaluation Module has two analog outputs that can be set to any voltage level from 0.0mV to 3300mV by the D/A converter. The tolerance of the D/A-converter is 8 bits (approx. 12.9mV). The internal D/A converter value 0(dec) stands for 0.0mV and 255 for 3300mV. To get the exact analog input characteristics, please see the hardware documents of the Evaluation Module.

The output voltage level is periodically refreshed from the RAM database to the output lines. The server application can set the new output voltage value to the RAM database individually for both analog outputs at any time. The refresh period of the analog output lines is 100 ms. This means that pulses shorter than 200 ms cannot be generated.

After the start-up self tests, the initial level of the analog output lines is 0.0mV.

	**Note:** Because the remote control interface specified for analog outputs by the IDL directly utilises mV-units, the server application does not need to take care of the scaling.

2.4 ABOUT THIS DOCUMENT

This document has three major parts:

Chapter 3 introduces how to use the Evaluation Module as a remote controller product with Nokia M2M Trial Gateway.

Chapter 4 tells how to set up the software development environment for the Evaluation Module embedded application including C-language compilers. The chapter includes instructions for updating the EM firmware.

Chapter 5 is meant for the developers who want to have a closer look to the embedded software and its functionality.

3. GETTING STARTED

3.1 OBTAINING NEEDED COMPONENTS TO USE THE EVALUATION MODULE

To use the Evaluation Module you need to have the following components:

1. PC with a free COM port
2. Evaluation module sales package
    - Package carton
    - Evaluation module
    - Power supply with AC cable
    - RS-232 data cable X 2
    - Nokia M2M ADK Installation CD-ROM
    - Product note
3. Nokia GSM Connectivity Terminal (N30 or N31) package (X2), each containing
    - Nokia GSM Connectivity Terminal
    - RS-232 adapter
    - RS-232 data cable
    - Power supply (ACW-5) with AC cable
4. Nokia M2M Software
    - Nokia M2M Gateway Trial Version
    - Nokia M2M ADK
    - Nokia GSM Connectivity Terminal Configurator software

The Evaluation Module sales package can be ordered through Forum Nokia website, http://www.forum.nokia.com, from the Nokia M2M section. You can order either an Evaluation Module sales package or the complete ADK sales package, containing also two Nokia GSM Connectivity Terminal packages and the Nokia M2M Application Development Kit on a CD.

The Nokia GSM Connectivity Terminals are also sold separately, for further information please refer to your local Nokia Sales Office.

The Nokia M2M ADK, Nokia M2M Gateway Trial Version, and the Nokia GSM Connectivity Terminal Configurator Software can be downloaded free of charge

from the Forum Nokia website, http://www.forum.nokia.com, Nokia M2M section. If you have bought the complete ADK sales package containing also the Nokia M2M Application Development Kit on a CD, you should check if there is a more recent version of the ADK available at the Forum Nokia website.

3.2 INSTALLATION

To be able to connect to the Evaluation Module remotely over the air, you need to have a Nokia M2M ADK environment installed and properly set up. The installation of the Nokia M2M ADK is explained in the *Nokia M2M Platform Application Development Kit Installation Guide*. Follow the instructions on how to install the ADK, Nokia M2M Gateway Trial version, Nokia GSM Connectivity Terminal Configurator software, and the Java environment. Also the configuration needed for the remote terminal (terminal attached to the Evaluation Module with the system connector cable) is described in that document.

The physical connections needed to run the Evaluation Module Controller Application (EMCA) are twofold. The server end, where there are EMCA, gateway, and the modem terminal is described in

Figure 3.

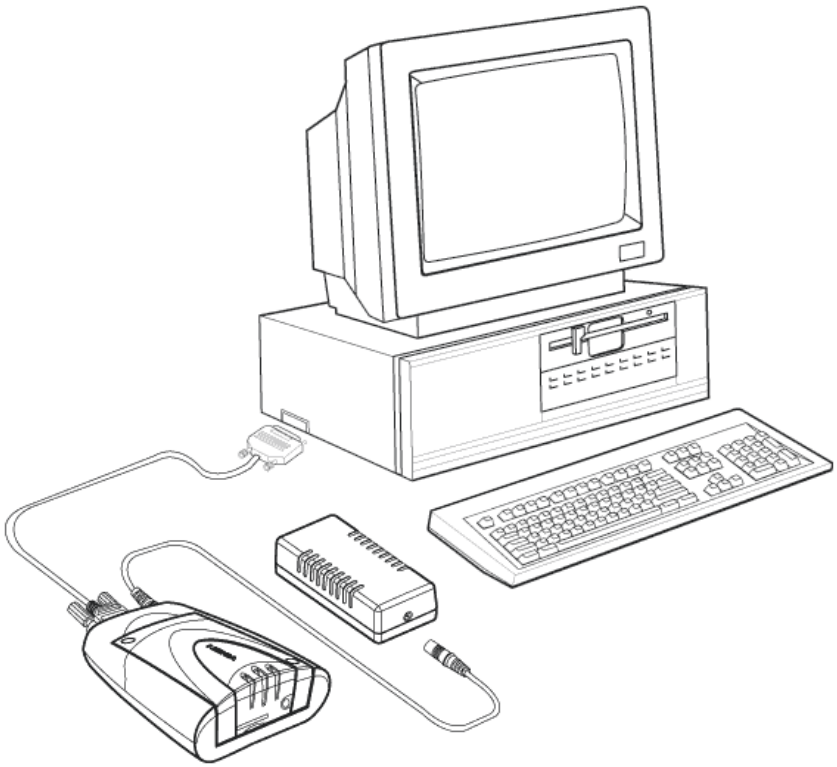


Figure 3

The remote end with the Evaluation Module and the remote terminal is illustrated in Figure 1.

	Note: There are two modes in the Evaluation Module: reprogramming mode and normal mode. The mode can be selected by using the mode switch as described in Figure 4. The reprogramming mode is used only when new software is being updated to the EM. In all other situations the EM should be in the normal mode. To check the mode of the EM you need to have a Nokia GSM Connectivity Terminal attached to the EM with system connector (flat cable) and either the terminal or the EM must be connected to power supply. If the EM is in the normal mode, the leftmost LED of the terminal blinks.

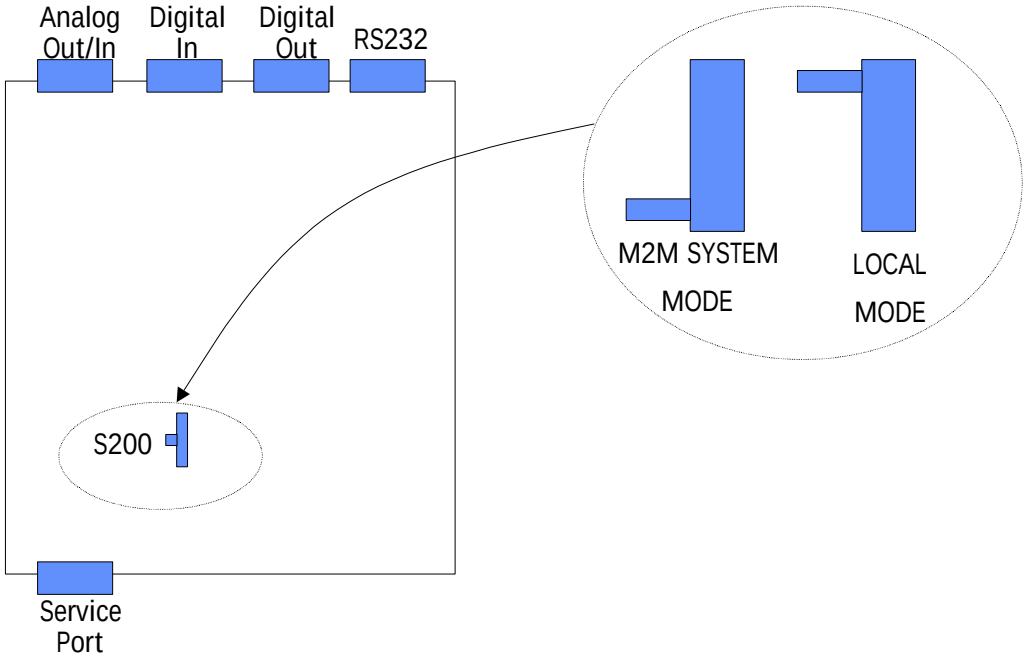


Figure 4

3.3 RUNNING THE EVALUATION MODULE CONTROLLER APPLICATION

3.3.1 STARTING THE NOKIA M2M GATEWAY TRIAL VERSION

Modem terminal must be connected to the computer via a serial cable before launching the gateway. Make sure that no other application is blocking the COM port that the gateway uses for communication with the modem terminal.

The Nokia M2M Gateway Trial Version can be started from a program menu shortcut that was installed during the gateway installation. Default settings for

the application use CSD as the wireless bearer, therefore choose the *Start Gateway with CSD support* command.

When the Nokia M2M Gateway Trial Version starts, it also starts another application, called naming service. The naming service is used by the gateway to obtain references to the remote objects. The naming service starts in its own console. When you wish to shut down the gateway, you can also shut down the naming service.

3.3.2 STARTING THE EVALUATION MODULE CONTROLLER APPLICATION

Open a command prompt in the [evaluation_module_installation_path]\ServerSide directory. You can run the application by executing the following command:

```
java -DInitialPort=900 -classpath classes
com.nokia.m2m.adk.em.emca.EvaluationModuleControllerApplication
–bearer csd –terminalID term123 –port 1
```

The –DInitialPort parameter is given to the ORB to define the port where a CORBA naming service is located. In the above command the EMCA has been given parameters about which bearer, terminalID and port to use. Actually the values passed in the above command are the default values, so the EMCA would use the same values also if it were to be run without any parameters.

There is also a batch file called EMCA.bat, which can be used to start the application with the default parameters.

3.3.3 USING THE EVALUATION MODULE CONTROLLER APPLICATION

The Evaluation Module Controller Application is a server side application that controls the Evaluation Module hardware remotely over the air. EMCA can be used to set the states of the 8 digital and 2 analog outputs, or to get the states of the 8 digital and 2 analog inputs. Also, the EMCA sets an observer to the Evaluation Module that keeps sending the voltages of the analog inputs to the EMCA with a user defined time interval. However, if the voltage has not changed since the last sent observation, an observer event is not sent.

Figure 2 shows a situation where EMCA is up and running. You can now modify the values of digital and analog outputs and send them to the Evaluation Module hardware by clicking *Set all output values* button. By clicking *Get all input values* button you can get all digital and analog input values. By clicking the *Set time value* button you can set the interval that the observer will use, i.e. the analog input pin observation interval in seconds. The default delay value for the observer is 120 seconds. From the Input mode and the Output mode text boxes you can see if the Evaluation Module Controller Application is ready for

new commands. From the Log text area you can see what the application is currently performing.

You can exit the application by choosing exit from the File menu or by pressing the x icon in upper right corner of the window.

	**Note:** You must terminate the EMCA before terminating the Gateway. If you stop the Gateway while the EMCA is still running, an observer will not be terminated from the Evaluation Module, causing it to try to establish a new connection to the EMCA. If such situation takes place, you need to restart the Evaluation Module by switching the power off.

4. EMBEDDED SOFTWARE DEVELOPMENT

This chapter describes how to update the Evaluation Module embedded software and how to compile the embedded software from the source code to the downloadable binary with supported compiler environment.

The Evaluation Module software package includes all the source code needed for compiling the binary. See Chapter 5 for the description of the EM embedded software.

4.1 RE-PROGRAMMING EVALUATION MODULE

The user can update the Evaluation Module firmware. The list of needed tools and equipment can be found in the Chapter 3.1.

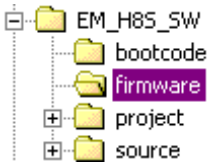
4.1.1 CONNECTING EVALUATION MODULE TO PC

In order to re-program the Evaluation Module, you need to connect the EM directly to your PC via serial cable. Use any of RS-232 data cables included the Nokia M2M ADK or Nokia GSM Connectivity Terminal product packages.

Before connecting the serial cable, turn the power off from the EM and from the GSM terminal. The RS-232 cable is connected to the ”Service port” in the back of the Evaluation Module (see Figure 4) and in any available communication port of the PC.

	**Note:** If the Nokia GSM Connectivity Terminal is connected to the Evaluation Module, please make sure that the Evaluation Module is in ”Reprogramming Mode”. For more information about the modes, please refer to the Chapter 3.2.

4.1.2 LOCATING DEFAULT FIRMWARE



The Evaluation Module firmware can be easily updated with the tools provided in the Application Development Kit. A *target binary file* contains the software that is already compiled and ready to be downloaded to the Evaluation Module. To locate the default firmware binary file included in Evaluation Module software package, go to the directory

```
[ADK_installation_path]/Nokia/Nokia M2M
ADK/EvaluationModule/EM_H8S_SW/firmware
```

	**Note:** If customized software is downloaded to the Evaluation Module, the binary can be found from different directories. See Chapter 4.2 to locate the custom compiled binaries.

4.1.3 USING FLASH DOWNLOAD UTILITY

For transferring the binary file to the Evaluation Module, a *flash download utility* software is needed. To locate this application, see

```
[ADK_installation_path]/Nokia/Nokia M2M ADK/tools
```

The flash download utility is a Win32 console application called `AM_flash.exe`. You may copy this tool to a directory that is included in your system PATH variable.

Command line parameters of AM_flash.exe are:

/H	Help switch, show help
/P	Port switch, defines communication port
/L	Defines optional log file

Example: `AM_flash EM_H8S_SW.hex /p1`

Flashes EM_H8S_SW.hex file via communication port 1 (COM1).

Example 2: `AM_flash EM_H8S_SW.hex /p2 /l test.log`

Flashes EM_H8S_SW.hex file via communication port 2 and uses test.log for log file.

Example 3: `AM_flash /h`

Shows the help text of the AM_flash application.

4.1.4 DOWNLOADING THE BINARY

Before starting the downloading procedure, make sure the Evaluation Module is connected to the PC as described in Chapter 4.1.1. Do not connect power to the EM or to the GSM terminal.

Start the flash download utility as described in Chapter 4.1.3 with the selected communication port and binary file. Following lines are displayed on console:

```
 Flash Download Utility For Application Modules, version 1.1
 Copyright (c) Nokia Corporation 2003. All rights reserved.

 Port COM1 and baud rate 115200 selected
 Connecting to AM
.........................................
```

Now connect the power to the Evaluation module. The flash downloading starts:

```
Checking and calculating HEX file size ...
 Start address: 0x00010000   End address: 0x00045f0a
 Total length:  0x00035f0a (220938 bytes)
 Receiving version data from M2M application ...
```

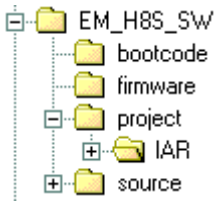
```
HW-version: 1.0
Protocol version: 1.0
Flash manufacturer ID:   0x2000
Flash device ID:   0x5b00
Flash erase complete.
210688 bytes remaining to flash
```

After the downloading has been successfully completed, a following notice is displayed:

```
0 bytes remaining to flash
Checking flash CRC ...
Flash programming successfully complete
```

The Evaluation Module software starts running after the flash procedure.

4.2 USING IAR EMBEDDED WORKBENCH FOR H8



Nokia M2M Evaluation Module software package contains a readymade project file for the IAR Embedded Workbench for H8S. IAR Embedded Workbench is a commercial C and assembler compiler product with a dedicated Windows user interface released by IAR Systems (http://www.iar.com).

4.2.1 GETTING IAR COMPILER TOOLS

In order to use IAR compiler environment, the IAR Embedded Workbench commercial or trial version must be installed to computer. Please refer the IAR web pages for information obtaining the IAR Embedded Workbench (http://www.iar.com → Products → Embedded Workbench).

The IAR Embedded Workbench must support Hitachi H8S target.

4.2.2 BUILDING EM_H8S_SW.HEX BINARY WITH IAR

Start the IAR Embedded Workbench for H8S application. Search the project file for the Evaluation Module (EM_H8S_SW.prj) by using menu File | Open. Figure 5 illustrates the situation.

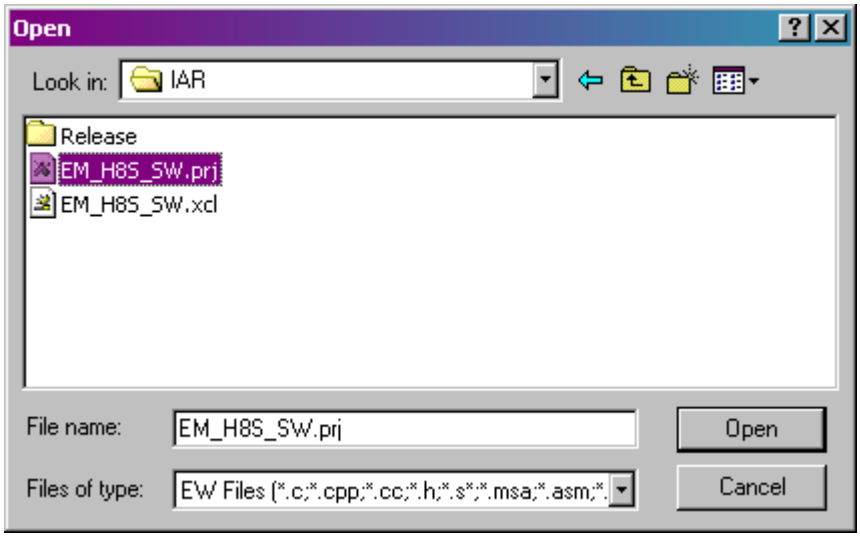


Figure 5. Opening IAR project file.

After opening the project, the file list of source files must be imported to the Release target. Click right mouse button on the top of Release icon and select quick menu Import File List as illustrated in Figure 6. This importing procedure must be done because the IAR Embedded Workbench does not support relative paths for source file locations.

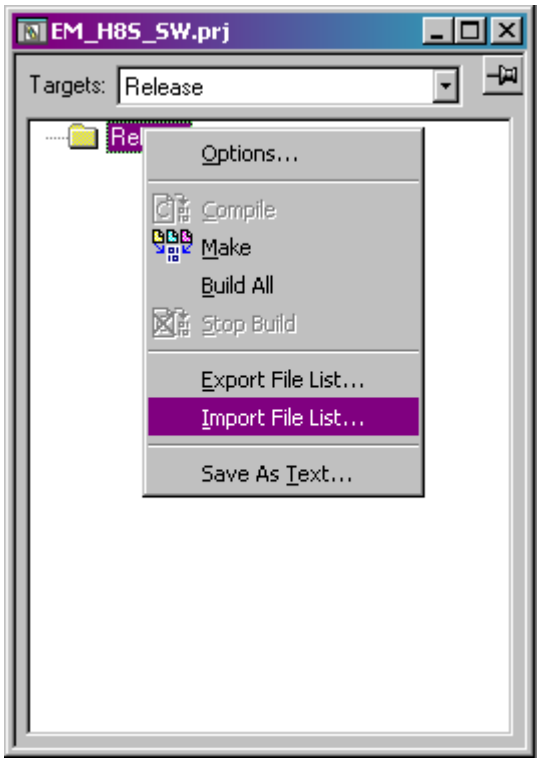


Figure 6. Importing File List.

Select EM_H8S_SW_file_list.txt as illustrated in Figure 7.

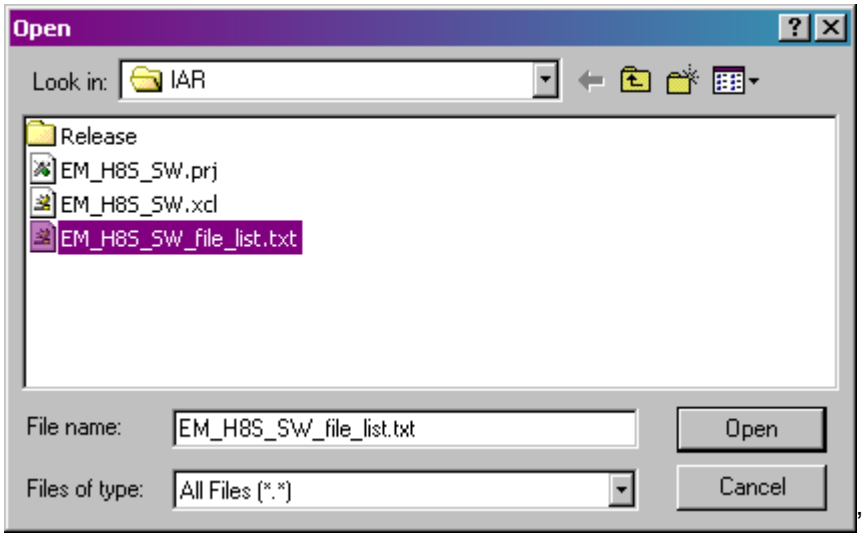


Figure 7. Selecting the file list for importing.

After importing the file list, the Release target has several groups containing application source files as illustrated in Figure 8. Next time the IAR Embedded Workbench is started there is no need to import the file list again.

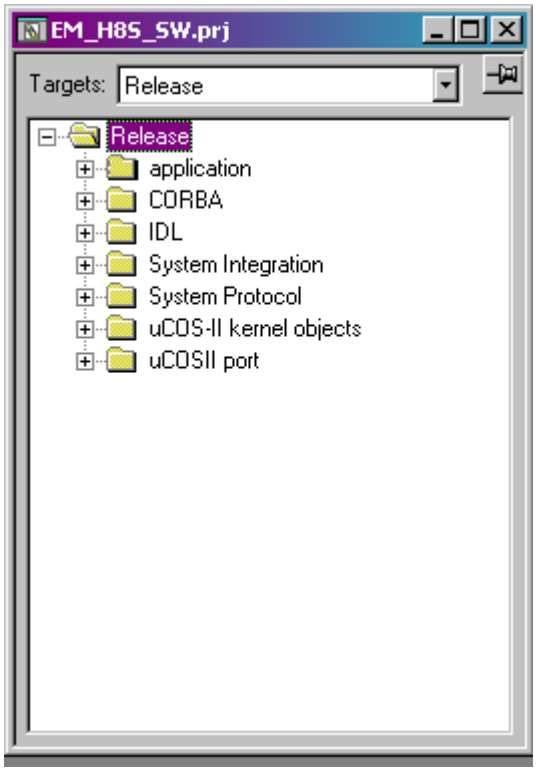


Figure 8. Imported file groups.

Now the Evaluation Module binary can be compiled. Select menu option Project | Make or just press F9 to build to project. Figure 9 illustrated the Messages window after compiling and linking the project files.

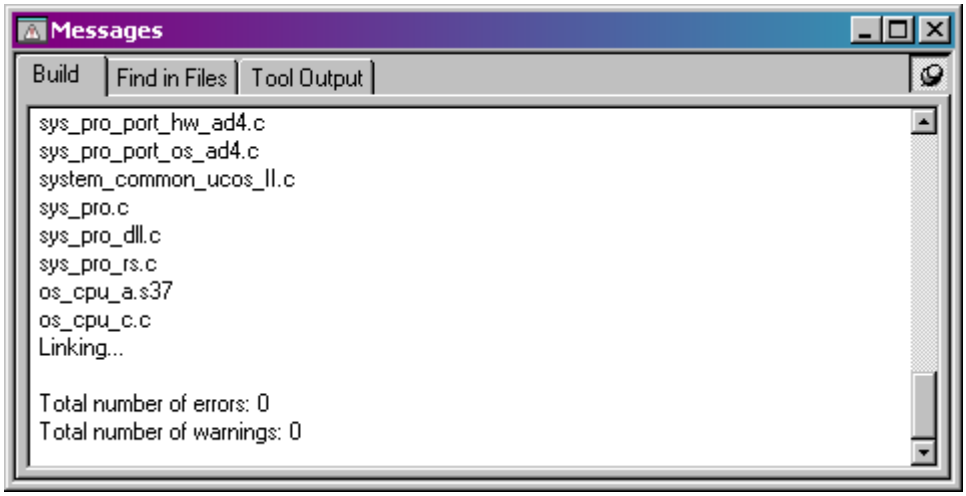


Figure 9. Message window after successful creation of the binary file.

The target binary `EM_H8S_SW.hex` is generated into the directory

```
[ADK_installation_path]/Nokia/Nokia M2M
ADK/EvaluationModule/EM_H8S_SW/project/IAR/Release/Target/
```

See Chapter 4.1 for instructions how to download the binary into the Evaluation Module.

4.3 DEBUGGING AND TRACING

There are mainly two ways to debug the Evaluation Module software. Firstly, the Evaluation Module is capable of sending ASCII trace data via serial port to a PC. Secondly, the serial level communication protocol traffic flow between the EM and the GSM terminal can be dumped to a file.

For more information about using serial tracing, see Chapter 5.4.1.

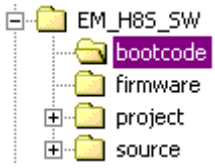
For guidance of setting the protocol dumping environment see Appendix A: Using M2M Dumper

5. EVALUATION MODULE H8S EMBEDDED SOFTWARE

This chapter describes briefly the Evaluation Module embedded software. The functionality or the interfaces of the current implementation may need to be changed, or the software may be used as an example for a custom M2M application.

Evaluation Module embedded software is constructed from two separate software parts: the boot code and the actual embedded software.

5.1 BOOT-UP CODE



The system startup is always initiated by a hardware reset caused by switching the power off and on, or if the processor watchdog mechanism makes a reset. Immediately after the reset, the boot up code will check if the flash programming device is connected to the EM. If it is connected, the EM starts the flash downloading process. The Chapter 4.1 describes how the flash downloading process can be started.

If there is no flash programming device connected the boot up code starts executing the actual embedded software of the Evaluation Module.

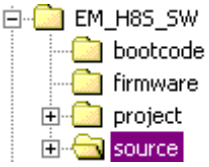
For more detailed information about boot-up code and flash downloading algorithm, please refer to Appendix B: Boot-up code.

The boot code is located in the directory

```
[ADK_installation_path]/Nokia/Nokia M2M
ADK/EvaluationModule/EM_H8S_SW/bootcode/
```

	**Note:** The boot code is pre-programmed to the Evaluation Module flash memory. For safety reasons the boot code section cannot be re-programmed with the tools provided in Application Development Kit.

5.2 EMBEDDED CORE SOFTWARE



The embedded software of the Evaluation Module is based on a template software called *AD4CoreSW*.

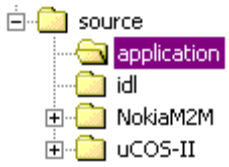
AD4CoreSW includes core functionalities needed to control embedded hardware environment similar to the EM. AD4CoreSW contains also software components for running CORBA middleware with μC/OS-II real-time operating system.

AD4CoreSW runs in Hitachi H8S series processor and it can be used as a platform for any custom remote end M2M application. A remote end application is also called Application Module (AM) in the Nokia M2M context.

Evaluation Module embedded software is an example implementation of an AM based on AD4CoreSW. The AD4CoreSW and Evaluation Module I/O Service are located in the directory:

```
[ADK_installation_path]/Nokia/Nokia M2M
ADK/EvaluationModule/EM_H8S_SW/source/
```

5.2.1 APPLICATION



The embedded software application logic and routines are located in *application* directory.

5.2.1.1 Startup routines

The boot-up code (described in Chapter 5.1) calls assembly start-up code called *start*. The start routine does H8S specific software initialization, like zeroing needed memory segments and doing mandatory hardware settings.

For the last, the start routine calls a C-routine called *main*.

For the related source code, see assembly file `cstartup.s37` for the IAR compiler.

5.2.1.2 System module

In addition to the main function, the *System* C-module contains initialization routines for some H8S hardware, real-time operating system and dynamic memory handling. The system contains also some useful functions to be used by other modules.

The main function continues system initialization started by the assembly routine start. The tasks of the main function are

- initializing H8S processor registers used later by the software
- initializing interrupt vectors used later by the software
- starting the real-time operating system and the first OS task called *system_task*.

System task continues the system initialization by calling set-up routines of other modules, including dynamic memory, serial tracing and EEPROM.

System module of the AD4CoreSW also takes care of other application specific initialization. In the Evaluation Module the System initializes Evaluation Module I/O Services.

For the related source code, see C-file `system.c` and `system.h`.

5.2.1.3 Software interrupts

Typical real-time embedded software functionality is based on using interrupts. An interrupt is triggered either by an external stimulus or internal event of the processor. Typically interrupts are used by low-level, fast software routines that require processor execution time instantly for a short period.

An example of the interrupt usage is the real-time operating system (RTOS) tick routine that must be run every ten milliseconds. A H8S counter is used for calculating ten milliseconds. When the counter has counted the time, an interrupt occurs. All other MCU activity is halted, and the interrupt routine executes the RTOS tick routine.

The interrupt vectors are defined file `int_vectors.c`. The assembly language interrupt handlers are implemented for example in `int_handler_a.s37` file for the IAR compiler.

5.2.1.4 Control Module

The AD4CoreSW offers one CORBA interface called *core/Control* for accessing software basic information through GSM network. This CORBA interface is described in `core.idl`. A server application can remotely

- get information about the device, including software name and version number,
- get and reset the latest reasons for software failure situations,
- get and reset the counter value of software restarts,
- reset the software, and
- read and write EEPROM for debugging purposes.

Control module also checks out the GSM terminal state after software boot-up using the CORBA interface ET of the GSM terminal. The routine notices if the GSM terminal is locked or is not connected to a GSM network. The Control can be configured so that it monitors some events of the GSM terminal during the software execution.

Other subsystems can send and receive text messages through Control text message service.

Control module implementation can be found in `control.c`. Interface definitions are in `control.h` file.

5.2.1.5 Serial tracing

In embedded software development phase it is often necessary to trace the software execution. One way to follow program steps is to use a serial tracing service.

Typically serial traces are readable texts that are sent to PC at a certain point of software. For example, a function may send ASCII characters that form a sentence “Example function started” through the H8S serial interface. A PC is then used for receiving these ASCII characters.

Caution: Generating traces consumes relatively lot of CPU processing time and memory. Enabling too much traces and using high serial communication bit rate may cause software to crash.

Tip: Chapter 5.4.1 describes how to enable and use serial tracing in the Evaluation Module software.

Serial tracing can be accessed through serial tracing interface in `trace.h`. The module implementation can be found from `trace.c`.

5.2.1.6 Micro controller Input/Output ports

The H8S micro controller has several ways to communicate with external devices. One interface is a set of processor input and output ports, including digital on/off, analog-to-digital and digital-to-analogue type channels.

The hardware configuration defines which I/O channels can be used. In the Evaluation Module there are eight digital inputs, two analogue inputs, eight digital outputs and two analogue outputs connected to the processor.

Evaluation Module HW Interface Specification describes the electrical specification of supported I/O.

MCU I/O can be accessed through MCU interface defined in `mcu.h`. For the MCU implementation, see `mcu.c`.

5.2.1.7 Serial communication

Another interface of the H8S micro controller is the *Serial Communication Interface* (SCI). The SCI block can handle three simultaneous, bi-directional serial communication sessions.

The Evaluation Module uses one SCI channel for communicating with the connected GSM terminal and one channel to communicate with the PC (if used for serial tracing).

Typically the asynchronous serial communication channel is first initialized for a supported bit rate. The bit rate can be 9600, 19200, 38400, 57600 or 115200 bits per second. The faster the communication is the more processor time it takes to handle the serial data.

The channel can be opened to receive data and/or the data can be sent. There must be a block of code to interpret incoming serial data and also a routine to generate the data to be sent.

H8S Serial Communication Interface can be accessed through SCI interface defined file `sci.h`. For SCI module implementation, see `sci.c`.

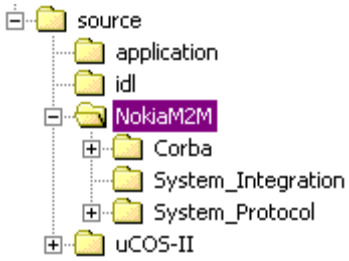
5.2.1.8 EEPROM

A typical hardware environment includes a non-volatile memory that stores the data when the power is switched off. The Evaluation Module contains an EEPROM chip that is connected to the H8S. The EEPROM can be used for saving a configuration data, for example.

The capacity of Evaluation Module EEPROM chip is 8192 bytes.

The EEPROM interface and the configuration are in the file `EEPROM.h`. The EEPROM device handlers are implemented in the module `EEPROM.c`.

5.2.2 M2M COMMUNICATION COMPONENTS



In order to communicate with server side application, there must be software components for the communication routines. *NokiaM2M* directory contains needed CORBA software components to communicate with server side application.

5.2.2.1 ORB

Object Request Broker (ORB) routines are the core components of CORBA (Common Object Request Broker Architecture) middleware functionality. Nokia M2M Platform uses CORBA to distribute application logic to remote and server ends.

For more information about CORBA standard, see http://www.omg.org. The *Nokia M2M Platform Remote Application Programming Guide* gives a detailed description of the Nokia C-ORB implementation.

ORB interfaces and implementation can be found in the directory `NokiaM2M/Corba`.

5.2.2.2 M2M System Protocol

The Nokia M2M System Protocol is a serial communication driver that handles all the transport and data link layer communication between the GSM terminal and the Evaluation Module. The M2M System Protocol service is typically used by the ORB routines.

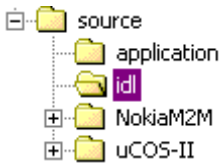
See *Nokia M2M Platform Remote Application Programming Guide* for detailed description of M2M System Protocol components.

5.2.2.3 Integration files

Integration part of Nokia M2M communication components includes those components that vary between different hardware and operating system environments. These so-called port files need to be implemented for used run-time target.

	**Tip:** Core parts of the Nokia C-ORB and M2M System Protocol are in ANSI C-language and can be used in variety of run-time environments. However, port files need to be implemented to the hardware and operating system environment the application uses. In Evaluation Module embedded software, this porting work has already been done for EM hardware (H8S micro controller) and uC/OS-II real-time operating system. See *Nokia M2M Platform Remote Application Programming Guide* for information on porting M2M communication components to other systems.

5.2.3 CORBA INTERFACE STUBS AND SKELETONS



CORBA Interface Definition Language (IDL) defines interfaces that bind the distributed parts of the system including embedded and server software.

CORBA interfaces can be used as a source to automatically generate programming language specific code. With Nokia C-ORB, an *IDL-to-C* compiler is used for generating CORBA stubs and skeletons from the IDL definitions. For information on how to use the IDL-to-C compiler, see *Nokia M2M Platform Remote Application Programming Guide*.

A *stub* is a routine that handles a client request to call a method defined in the CORBA interface. The stub uses ORB library routines to generate a CORBA message, pack parameter data into it and deliver the message to the correct server process. Generated C-language stubs are in files that end with `_Client.c`.

A skeleton is a server routine that handles an incoming CORBA request. The skeleton uses ORB library routines to unpack parameter information from the CORBA message. The skeleton calls a C-implementation of the method call.

The server routine also takes care of sending response to the originating client process. Generated skeletons are in files ending with `_Server.c`.

Figure 10 illustrates the roles of stubs and skeletons.

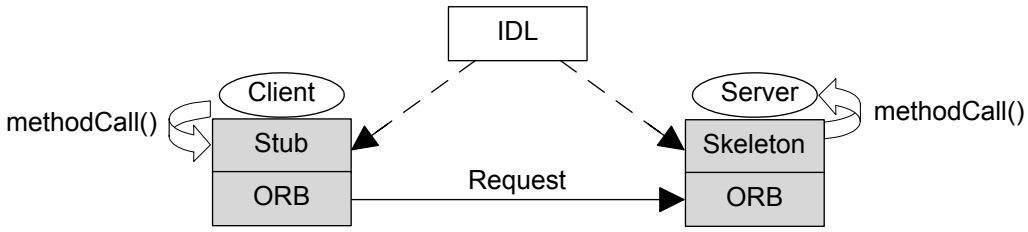
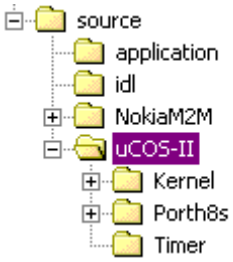


Figure 10. Role of stub and skeleton codes.

An implementation is the custom server code that is executed by the CORBA server process (skeleton). The CORBA interface does not define how the implementation is done.

The implementations are in files ending `_Impl.c`. However, often the implementation code is located under the corresponding locigal C-module. For example, the implementations of *core/Control* CORBA interface reside in `control.c` module instead of `core_Impl.c`.

5.2.4 REAL-TIME OPERATING SYSTEM



Typical embedded applications use the services provided by a Real-Time Operating System (RTOS) to share efficiently processor resources like CPU time.

5.2.4.1 Kernel

The operating system of the EM is µC/OS-II (MicroU/OS-II), preemptive multistack real-time kernel for embedded micro controllers from Micrium (http://www.micrium.com). The Evaluation Module software uses a binary license of the µC/OS-II operating system, so there is no kernel source code available.

The kernel functionality can be used via the RTOS interface defined in file `ucos_ii.h`. However, the RTOS configuration has been predefined for the Evaluation Module software package. If this configuration needs to be changed, the kernel source code must be obtained and recompiled.

The binary object files for IAR compilation environment are located in the directory

```
[ADK_installation_path]/Nokia/Nokia M2M
ADK/EvaluationModule/EM_H8S_SW/project/IAR/Release/uCOSII_Obj/
```

Caution: The µC/OS-II kernel is distributed under a binary license for Evaluation Module product only. If the kernel is used for any other custom application (e.g. Evaluation Module with modified software), a proper license must be obtained from Micrium (http://www.micrium.com).

5.2.4.2 H8S Ports

The µC/OS-II operating system needs to be ported for the used micro controller and the C-compiler. Various operating system ports can be found in Micrium's home page (http://www.micrium.com → Products → µC/OS II RTOS → Ports).

The Evaluation Module software package includes µC/OS-II port for H8S micro controller and for the IAR compiler.

5.2.4.3 Timer module

The µC/OS-II kernel does not offer any timer services. However, there is a software package called *uC/OS-II Building Blocks* available from Micrium that includes also timing services. The module is located in file `Tmr.c`.

The timer has a resolution of 100 milliseconds and a number of timers can be running at the same time. When a timer expires a defined callback function is called. For example the M2M System Protocol uses this timer service.

The Evaluation Module software package uses a binary license of the µC/OS-II Building Blocks, so there is no module source code available.

The timer functionality can still be used via Tmr interface defined in `Tmr.h` file. The file `Tmr_config.h` includes timer identifications that must be unique for each timer.

Caution: The µC/OS-II Tmr module is distributed under a binary license for Evaluation Module product only. If the Tmr block is used for any other custom application (e.g. Evaluation Module with modified software), a proper license must be obtained from Micrium (http://www.micrium.com).

5.3 EVALUATION MODULE I/O SERVICE

The Evaluation Module I/O service has been implemented to one module called `io.c`. The module and the corresponding interface definition file `io.h` are located in `application` directory.

The module contains several tasks that handle different kind of operations and routines related to physical I/O of the EM. The module also implements the CORBA *IOModule/IOControl* interface.

IOModule/IOControl interface definition is located in

```
[ADK_installation_path]/Nokia/Nokia M2M
ADK/EvaluationModule/IDL/
```

5.3.1 TIMEOUT TASK

When the I/O Sevice is started the timeout task is the first I/O Service task to run. Timeout task initializes needed CORBA objects and starts timers called IO_INPUT_UPDATE_TIMER and IO_OUTPUT_UPDATE_TIMER. After the initialization, the timeout task waits timers to expire. All the further activity is based on timeouts.

There are three timers used in I/O Service:

IO_INPUT_UPDATE_TIMER is used for generating a periodic event that saves digital and analog input pin states to the I/O Service data structure. The event also checks if the values of the digital inputs have changed since last checking. If the digital observer is on and a digital input state has changed the task asks the event task to send a digital observation event to the server application (see IO_EVENT_DIGITAL command in Chapter 5.3.3.

IO_OUTPUT_UPDATE_TIMER is used for generating a periodic event that sets the digital and analog output pin states to hardware according the I/O Service data structure.

IO_ANALOG_OBS_TIMER is used for generating a periodic event that asks the event task to send an analog observation event to the server application if an analog input value has been changed since the latest checking (see IO_EVENT_ANALOG command in Chapter 5.3.3). The observation event is sent only if the analog observer is on. The timeout value can be configured via IOModule/IOControl CORBA interface.

5.3.2 EEPROM TASK

The EEPROM task takes care of keeping an updated copy the I/O Service data structure in volatile EEPROM.

There are three commands that can be sent to the task:

IO_DATA_MAKE_RAM_COPY tells the task to retrieve the I/O Service data structure from EEPROM and save it to RAM memory. This event is performed for example in the I/O Service startup when the previous states of output pins are restored.

IO_DATA_SET_DEFAULTS writes a default I/O Service data to EEPROM. This action is performed for example when the software starts the first time and EEPROM memory is empty.

IO_DATA_UPDATE_EEPROM writes RAM copy of the I/O Service data structure to EEPROM. This action is performed for example when the I/O settings or the output pin states are changed via the CORBA interface.

5.3.3 EVENT TASK

The event task handles commands that are related to the I/O Service observation events. The possible events are:

IO_EVENT_SET_OBSERVER is used for setting a new observer object. If there is an observer object already registered into the I/O Service it is freed before setting a new one. The observation can be set for digital and/or analog input pins. This command is typically called from IOModule/IOControl CORBA interface.

IO_EVENT_REMOVE_OBSERVER is used for removing existing observer object from the I/O Service. This command is typically called from IOModule/IOControl CORBA interface.

IO_EVENT_ANALOG command calls the CORBA observer interface for notifying the server application about an analog input pin state change if the analog observer is on. This command is typically called from the timeout task.

IO_EVENT_DIGITAL command calls the CORBA observer interface for notifying the server application about a digital input pin state change. This command is typically called from the timeout task.

5.3.4 IOMODULE/IOCONTROL CORBA INTERFACE IMPLEMENTATION

The routines of *IOModule/IOControl* interface control I/O Service functionality. The analog observer delay can be read and set, an observer can be added or removed, the states of output pins can be changed and the states of input pins can be read.

See Chapter 5.2.3 for more information about CORBA stubs, skeletons and implementation.

The IOModule/IOControl stub and skeleton source files are in directory

```
[ADK_installation_path]/Nokia/Nokia M2M
ADK/EvaluationModule/EM_H8S_SW/source/idl/
```

5.4 PROGRAMMING EXAMPLES

This chapter introduces few examples how to alter the original Evaluation Module embedded software.

5.4.1 SWITCHING SERIAL TRACING ON

By default, the Evaluation Module embedded software does not print any serial traces through the “Service port”. However, the developer can easily turn the tracing on and see traces with a PC. For more information about serial tracing feature, see Chapter 5.2.1.5.

1. Open the configuration file `sw_confing.h` from directory `application`.

2. Find the line containing following text:

```
/* ---------------------------------------------------------------------
 * Tracing interface definitions (trace.c)
 */
/* EM_H8S_SW */
/*#define SWCONF_TRACE_PRINT_ENABLED*/      /* definition enables serial
                                               traces */
#define SWCONF_USED_TX_SCI_CHANNEL          SCI_CHANNEL_2
#define SWCONF_USED_TX_BIT_RATE             115200
```

The first definition, SWCONF_TRACE_PRINT_ENABLED, is a master flag for serial tracing. The SWCONF_USED_TX_SCI_CHANNEL defines which of the H8S micro controller Serial Communication Interface channel is used for serial tracing. In the Evaluation Module the D9 connector of the “Service port” is connected to the SCI channel 2. The SWCONF_USED_TX_BIT_RATE defines the used bit rate. Possible options are 9600, 19200, 38400, 57600 and 115200.

3. Enable tracing by removing C-comments around the define:

```
#define SWCONF_TRACE_PRINT_ENABLED          /* definition enables serial
                                               traces */
```

4. Save the `sw_config.h` file.

5. Compile and re-program the Evaluation Module software (see Chapter 4 for information how to compile and re-program the EM).

6. Connect a serial cable from the EM “Service port” to a PC communication port (the same communication port can be used as is used for re-programming purposes).

7. Start a PC program that is capable of displaying incoming characters from the used serial communication port. Configure this terminal application for the bit speed of 115200. Other serial communication setups are: 8 data bits, parity none, 1 stop bit and no flow control.

8. When the Evaluation Module is rebooted following serial trace should be displayed depending on the used compiler and software configuration:

```
*************************** Software bootup ****************************
[system] Software compiled with IAR C-compiler.
[system] uC/OS-II operating system v. 2.62
[system] software bootup counter:    31
[CORBA] Object Request Broker initialized
[CORBA] Object Adapter running
[I/O] Evaluation Module I/O Service started
[control] Nokia M2M Evaluation Module (EM) v.2.2.0
[control] Terminal is searching network...
[control] Terminal is searching network...
MEM (curr/peak) PART1:    1/   4 PART2:   10/  19 PART3:   12/  21
MEM (curr/peak) PART4:    6/  12 PART5:    2/   2 PART6:    0/   0
[control] Terminal is searching network...
[control] Terminal registered into GSM network
MEM (curr/peak) PART1:    1/   4 PART2:   10/  19 PART3:   12/  21
MEM (curr/peak) PART4:    6/  12 PART5:    2/   2 PART6:    0/   0
…
```

5.4.2 CHANGING SOFTWARE VERSION INFORMATION

The CORBA interface *core/Control* offers a possibility to enquire the software name, hardware name and the software version of the device. If the original software is altered, the user may also change the version information.

Typically the Control module (`control.c`) displays the module software version in system startup if the serial traces are enabled.

For changing software and hardware name follow the steps:

1. Open `sw_config.h` configuration file located in directory application. Find the following lines:

```
/* EM_H8S_SW */
#define SWCONF_HW_PRODUCT_NAME       "Nokia M2M Evaluation Module (EM)"
#define SWCONF_SW_PRODUCT_NAME       "Nokia M2M Evaluation Module (EM)"
```

These define the product software and hardware name.

2. Change SWCONF_SW_PRODUCT_NAME:

```
#define SWCONF_SW_PRODUCT_NAME       "Altered EM SW”
```

3. Save `sw_config.h` file

4. If the IAR compiler is used, select menu Project | Options…. Select Category “ICCH8”. Select the tab “#define”

`Modify the version codes as follows:`

```
SW_VERSION_RELEASE_CODE          = 1
SW_VERSION_MAJOR_CODE            = 0
SW_VERSION_MINOR_CODE            = 1
```

5. Compile and reprogram the Evaluation Module software (see Chapter 4 for information how to compile and re-program the EM).

Now the new software name is “Altered EM SW” and the version number is 1.0.1. (Release 1, version 0.1).

5.4.3 ENABLING I/O SERVICE SERIAL TRACES

To follow the I/O Service module run-time activity, the module specific traces can be enabled.

Note: Before using module specific serial traces, the master trace flag must be enabled. See Chapter 5.4.1 how to enable serial tracing.

1. Open `io.c` file from directory `application`

   Remove C-comments from trace flag definitions:

```
/* Tracing options
 */
#define IO_ERR_TRACING
/*#define IO_MSG_TRACING*/
/*#define IO_DATA_TRACING*/
```

→

```
/* Tracing options
 */
#define IO_ERR_TRACING
#define IO_MSG_TRACING
#define IO_DATA_TRACING
```

2. Save `io.c` file.
3. Compile and reprogram the Evaluation Module software (see Chapter 4 for information how to compile and re-program the EM).

Now the I/O Service module displays traces as the software runs.

6. TROUBLESHOOTING

6.1 THE TERMINAL'S LED INDICATING THE NORMAL MODE OF THE EVALUATION MODULE DOES NOT BLINK

Check that the mode switch is in normal mode (see Figure 4). If the leftmost LED of the terminal still does not blink, make sure that the remote terminal and the EM use the same baud rate on the system protocol (default in both of them is 9600 bps). You can set the baud rate of the terminal by using the Nokia GSM Connectivity Terminal Configurator application. On the *M2M System Settings* menu, choose the *M2M System Protocol* command. A dialog box opens. Click the *Read parameters* button. If the baud rate is not 9600, set that value and click the *Write parameters* button.

6.2 THE EVALUATION MODULE CONTROLLER APPLICATION CAN NOT CONNECT TO THE EVALUATION MODULE

If you have configured the ADK environment as instructed in the *Nokia M2M Platform Application Development Kit Installation Guide* but the EMCA cannot control the EM remotely, check the log of the EMCA. If there was a communication error (a CORBA COMM_FAILURE exception was raised) at some point in the Nokia M2M Platform, the EMCA shows a description of the error in the log of the application as shown in Figure 11.

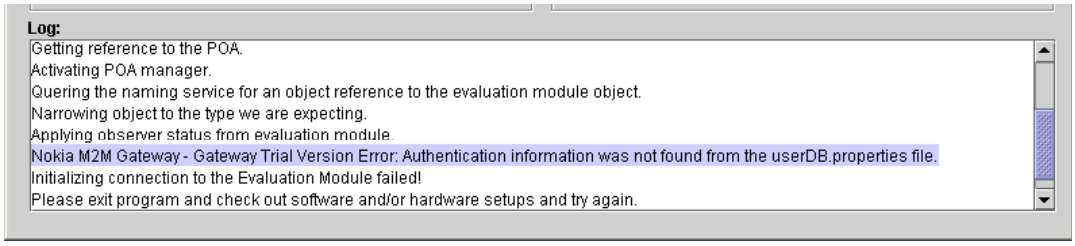


Figure 11 Log dialog of the EMCA

In the above case the Nokia M2M Gateway Trial version could not find the required authentication information from the userDB.properties file.

7. APPENDIX A: USING M2M DUMPER

7.1 GENERAL

The M2M_Dumper is a tool program for dumping system protocol messages from the EM. The dumper tool can be used for low-level protocol monitoring of M2M System Protocol.

7.2 CABLE CONNECTIONS

RS-232 cable connection from a PC is described in Figure 12.

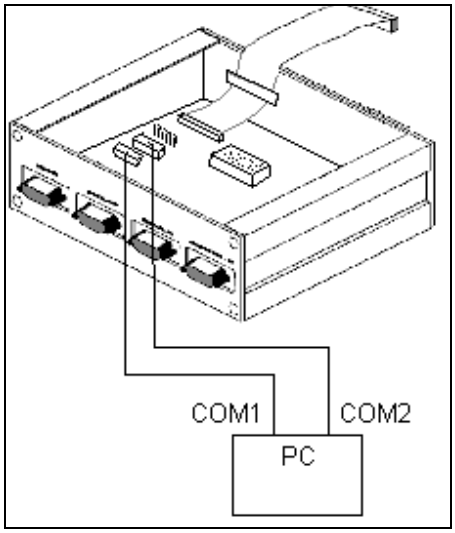


Figure 12. M2M Dumper cable connections.

7.3 USING M2M_DUMPER PROGRAM

Before the system protocol dumper can start the PC (M2M_Dumper.exe) program has to be started with the current parameters from the command prompt and then configure the program parameters to M2M_PARSER.INI file.

7.4 PARAMETER CONFIGURATION FILE (M2M_PARSER.INI)

The program reads the parameters when it starts. If you want to change the parameters the program must be restarted.

The M2M_PARSER.INI file accepts the following parameters:

speed=xxx	Speed of communications via com ports, normally 9600 bit/s
timestamp= xx	Timestamp printing on/off
data parser = xx	Data parser mode, in this mode the incoming system protocol data is parsed after being displaying.
length=xxx	In data parser mode, this parameter defines how long data is displayed on the screen.
Com1=xxxx	Defines what is the physical communication port in PC for COM1.
Com2=xxxx	Defines what is the physical communication port in PC for COM2.

Example file:

```
[PARAMETERS]
speed = 9600
timestamp = on
length = 400
Com1 = COM1
Com2 = COM2
```

7.5 SOFTWARE START-UP

The program supports three startup options: basic trace, trace with save log file and help screen. Additional information is presented later in this document.

Start commands:

M2M_Dumper.exe
M2M_Dumper.exe log.txt
M2M_Dumper.exe /h

Running commands:

“F2” key, numbered stamp.
“F3” key, clear timestamp.
“F4” key, numbered stamp and clear timestamp.

Stop commands:

“Enter” key
“Ctrl” + “c” key

When the program is started it tries to load the parameter file from the same folder where the program is located. If the parameter file is valid the program starts to listen for M2M system protocol packets. When a packet is received it is shown on the screen and optionally saved to hard disk. If the parameter file is invalid the software acknowledges the user about it and uses the default settings. If the file is not found the software acknowledges the user about it and creates a default parameter file.

7.6 SOFTWARE FUNCTIONS

The program is a very simple data tracer which only reads serial data, parses it, echoes it to the screen and saves it to disk. The data is presented in a GIOP-packet data format or hexadecimal mode.

Functions:

- Read data from user defined serial ports (COM1-COM9)
- Parse M2M system protocol packet
- Parse GIOP-packet
- Calculate check sum
- On screen GIOP-packet length adjustable
- Bookmark option
- Shows packet in command prompt. (Hex or data format)
- Save log file to hard-disk
- Possibility to put timestamp (only when saving to a log file)
- Possibility to reset timestamp while program is running.
- Speed change support
- Help

7.6.1 HELP SCREEN -ARGUMENT

Help screen (Figure 13) mode shows the help functionality of this program. It describes the use of the command line parameters.

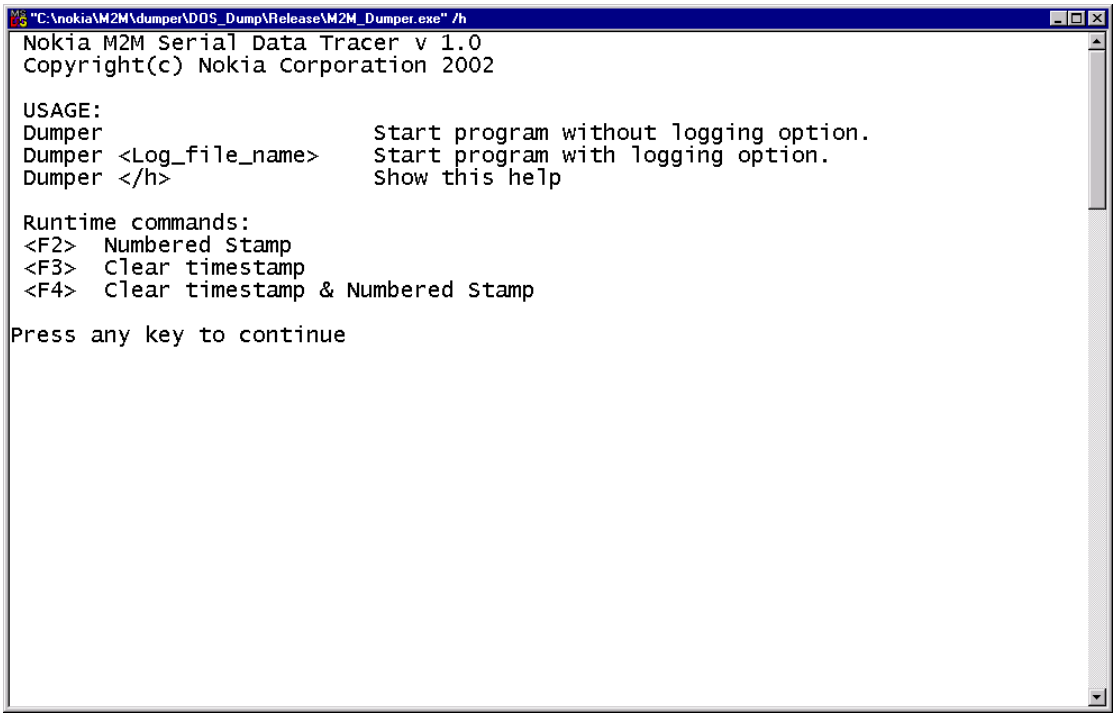


Figure 13. Example screen from serial data tracer.

7.6.2 SCREEN CAPTURE EXAMPLE

Figure 14 demonstrates how the program operates and how it looks like in Windows NT

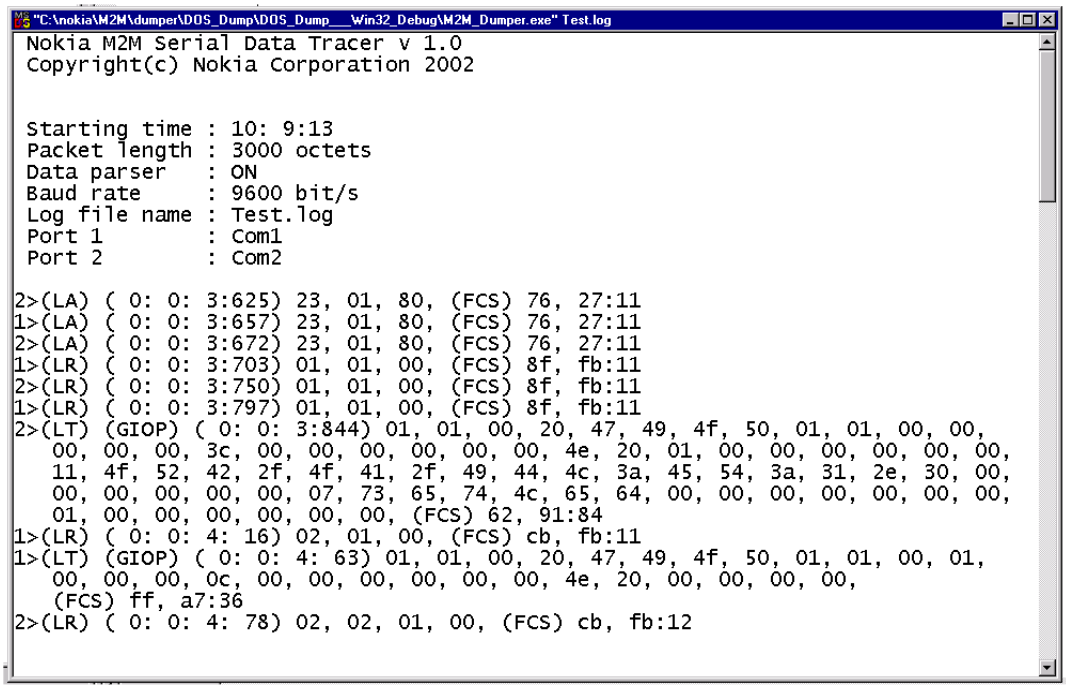


Figure 14. Example screen from serial data tracer

8. APPENDIX B: BOOT-UP CODE AND FLASH DOWNLOAD PROTOCOL

8.1 GENERAL

The EM boot code is separated into two different parts. Primary boot code is for normal EM start-up code and algorithm code is used only for flash programming and downloading.

The EM software can be updated by downloading the new software to the flash memory. The SW download should start after HW reset (either the main power is connected or the reset button is pressed). Supported binary file formats are the INTEL 32 bit HEX and the MOTOROLA S formats with 24 bit addresses.

8.2 STARTUP

The boot code will wait one second for data from the service port (that is located on back side of the EM). If the EM boot code finds “0xFF 0xAA” data from the service port software downloading from service port starts. Otherwise the normal start-up sequence continues.

SW downloading software runs on the 32-bit Windows (Windows 9X, ME, NT4, 2000, and XP)

The primary boot code is always executed at the start-up of EM. It waits one second for a detection stream from the service port. If the primary boot receives a detection stream from the service port, algorithm code will be copied to an external RAM area starting address 0x800020h and executed. Otherwise, normal start-up continues.

	**Note:** When downloading a new binary, the EM must be in reprogramming mode. See Chapter 3.2.

8.3 FLASH DOWNLOADING PROTOCOL

8.3.1 MESSAGE FRAME

Messages are described using the following illustration:

Byte(s)	Size (bytes)	field	Description

Where ‘Byte(s)’ tells the order numbers of byte(s) reserved for field and thus, the numbering begins from 1 not 0. ‘field’ tells the name of the field and ‘description’ gives a short description of the field.

The first two bytes in every message indicate message ID and size of message (in bytes). Evaluation module accepts only messages with IDs declared in this document, other message-IDs generate an error. Size of a message is compared to the actual amount of bytes received by the Evaluation Module and incorrect size of message generates an error.

8.3.2 MESSAGES FROM WORKSTATION TO EVALUATION MODULE

These messages are sent by tool software to the Evaluation Module.

8.3.2.1 FLASH_VERSION_GET_COMMAND

This message is used to query software and hardware versions of the Evaluation Module. The Evaluation Module replies with the message FLASH_VERSION_GET_RESPONSE. Protocol version must be 1.0, otherwise Evaluation Module returns an error.

1	1	command_id	Message ID = 0x56H (ASCII ‘V’)
2..3	2	size	Size of message in bytes
4..5	2	protocol_version	Version number of protocol

8.3.2.2 FLASH_INIT_DOWNLOAD_COMMAND

This message is used to initialise the flash data programming. The message informs the Evaluation Module of the starting address of data (in the Evaluation Module’s flash memory) and the size of the data. The Evaluation Module should at least check that the start address is valid and that the data is not too large to fit in the memory. The Evaluation Module replies with the message FLASH_INIT_DOWNLOAD_RESPONSE.

1	1	command_id	Message ID = 0x49H (ASCII ‘I’)
2..3	2	size	Size of message in bytes
4..7	4	start_address	Start address of software in memory (MSB first)
8..11	4	length	Size of data (MSB first)

8.3.2.3 FLASH_DOWNLOAD_DATA_COMMAND

This message is used to send data to program to the Evaluation Module. The message contains a start address for the data (in Evaluation Module's memory) and size of the data in bytes. The actual data is an array of (max 256) bytes, and a calculated 16-bit CRC checksum is used to check transfer errors. The Evaluation Module replies to every message with the message FLASH_DOWNLOAD_DATA_RESPONSE.

1	1	command_id	Message ID = 0x44H (ASCII 'D')
2..3	2	size	Size of message in bytes
4..7	4	address	Address of data in memory (MSB first)
8..9	2	length_of_data	Size of data in this packet (MSB first)
10..265	256	data	Program data
266..267	2	crc	CRC checksum of data (MSB first)

8.3.2.4 FLASH_EXIT_DOWNLOAD_COMMAND

This message is used to inform the Evaluation Module that the software download is completed. A message informs the Evaluation Module about the start address and the size of the data. The Evaluation Module responds with the message FLASH_EXIT_DOWNLOAD_RESPONSE. If *use_run_address* field contains value 0x0H, *run_address* is ignored by Evaluation Module. If *use_run_address* contains 0x1H, *run_address* contains an address at memory where software must be run after successful flash programming. This feature is not implemented in current 1.0 version

1	1	command_id	Message ID = 0x58H (ASCII ‘X’)
2..3	2	size	Size of message in bytes
4..7	4	start_address	Start address of software in memory (MSB first)
8..11	4	length	Size of data (MSB first)
12	1	use_run_address	Indicates (0/1) if run_address is used
13..17	4	run_address	Address in memory where software is run (MSB first)

8.3.3 MESSAGES FROM EVALUATION MODULE TO WORKSTATION

These messages are sent by Evaluation Module to the tool software. Every message is a response to the message sent by the tool software.

8.3.3.1 FLASH_VERSION_GET_RESPONSE

This message is sent as a reply to the message FLASH_VERSION_GET_COMMAND. The message informs the tool software about software and hardware versions of the Evaluation Module. ID for a flash chip can also be sent.

1	1	response_id	Message ID = 0x76H (ASCII ‘v’)
2..3	2	size	Size of message in bytes
4..5	2	sw_version	Version of boot code in Evaluation Module (MSB first)
6..7	2	hw_version	Hardware version of Evaluation Module (MSB first)
8..9	2	protocol_version	Version of protocol used
10..11	2	flash_manufact_id	Flash manufacturer ID
12,,13	2	flash_dev_id	Flash device ID
14	1	status	Status returned
15..18	4	reserved	Reserved

Values of status field:

Status	Value	Description
FLASH_VERSION_OK	0x01H	Version request OK
FLASH_VERSION_FAIL	0x02H	Unsupported protocol version

8.3.3.2 FLASH_INIT_DOWNLOAD_RESPONSE

This message is sent as a reply to the message FLASH_INIT_DOWNLOAD_COMMAND. The message returns the success/error status to the tool software.

1	1	response_id	Message ID = 0x69H (ASCII ‘i’)
2..3	2	size	Size of message in bytes
4	1	status	Status returned

Values of status field:

Status	Value	Description
FLASH_INIT_DOWNLOAD_OK	0x01H	Init OK
FLASH_INIT_DOWNLOAD_SW_TOO_BIG	0x02H	Software does not fit to memory

8.3.3.3 FLASH_DOWNLOAD_DATA_RESPONSE

This message is sent as a reply to the message FLASH_DOWNLOAD_DATA_COMMAND. The message returns the success/error status to the tool software.

1	1	Response_id	Message ID = 0x64H (ASCII ‘d’)
2..3	2	size	Size of message in bytes
4	1	status	Status returned

Values of status field:

Status	Value	Description
FLASH_DOWNLOAD_OK	0x01H	Data download successful
FLASH_DOWNLOAD_CRC_FAIL	0x02H	CRC failure in data transfer
FLASH_DOWNLOAD_BLOCK_PROTECTED	0x03H	Flash data block write protected

8.3.3.4 FLASH_EXIT_DOWNLOAD_RESPONSE

This message is sent as a reply to the message FLASH_EXIT_DOWNLOAD_COMMAND. The message returns a 16-bit CRC checksum of programmed data to the tool software. The tool software must check this CRC sum against the CRC sum tool software that it has calculated from the data itself.

1	1	response_id	Message ID = 0x78H (ASCII ‘x’)
2..3	2	size	Size of message in bytes
4..5	2	crc	CRC checksum of programmed data (MSB first)
6	1	status	Status returned

Values of status field:

Status	Value	Description
FLASH_EXIT_OK	0x01H	Download exit successful
FLASH_EXIT_FAIL	0x02H	Failure when exiting download

8.3.4 FLASH_ERROR

This message is sent in reply to every message when a critical error occurs.

1	1	response_id	Message ID = 0x65H (ASCII ‘e’)
2..3	2	size	Size of message in bytes
4	1	status	Status returned

Values of status field:

Status	Value	Description
FLASH_ERROR_UNKNOWN_MSG	0x0AH	Received message (number) unknown
FLASH_ERROR_MSG_SIZE	0x0BH	Message size wrong
FLASH_ERROR_WRITE_FAIL	0x0DH	Error writing to flash chip.
FLASH_ERROR_ERASE_FAIL	0x0eH	Flash memory could not be erased.

8.3.5 EXAMPLE SEQUENCE OF SUCCESSFUL FLASH DOWNLOADING

The sequence of programming and messages can be seen in Figure 15.

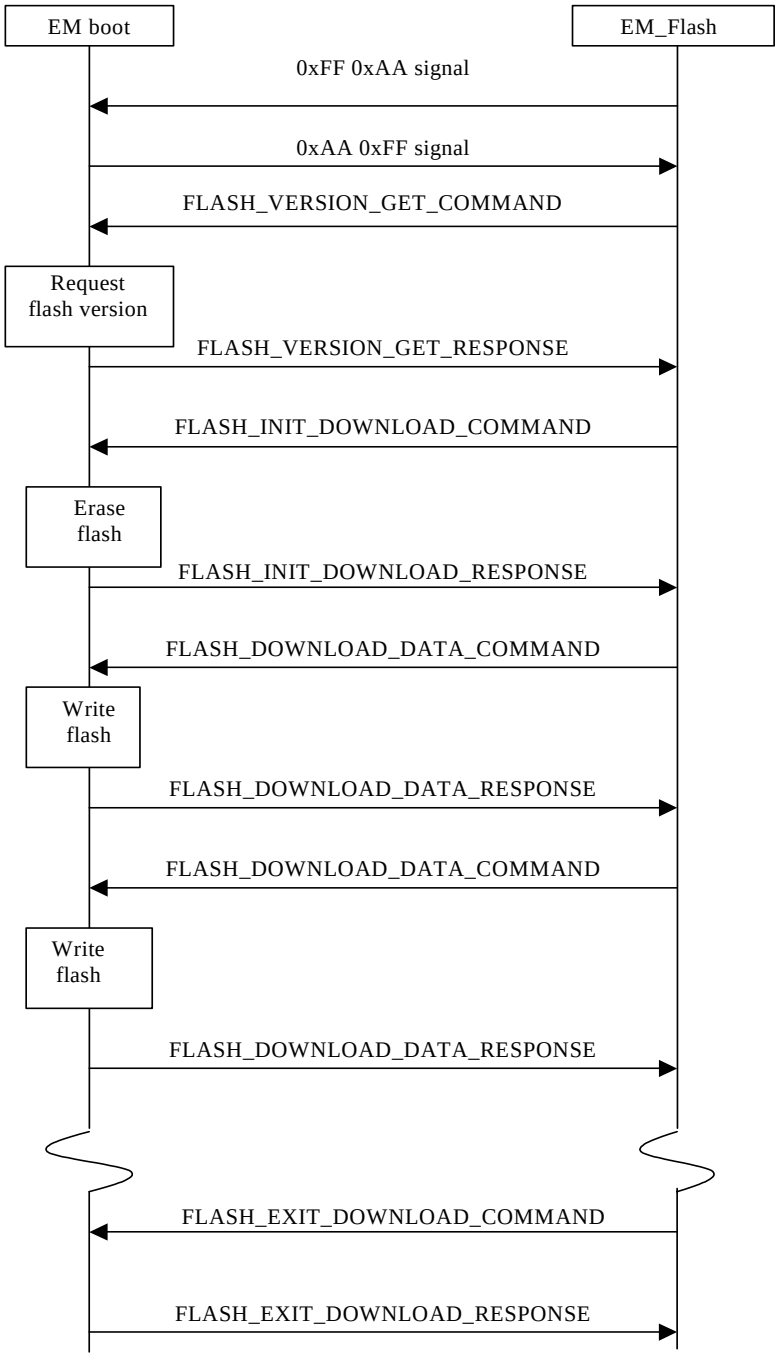


Figure 15. Flash programming sequence.

8.4 MEMORY MAPS

8.4.1 PRIMARY BOOT CODE MEMORY MAP

Segment	Address	Description
PRI_INTVEC	00000000 - 0000016F	Primary boot interrupt vector
PRI_BOOT_CODE	00000170 - 000003CF	Primary boot code
ALG_CODE_DATA	00000F00 - 00001FF3	Algorithm code data area
FLASH_HEADER_SEG	00010000 – 0001003F	Flash header segment
VERSION_ID_PTR	00010040 – 00010043	Version ID string start address
INTVEC	00010100 - 0001026F	Interrupt vector for code
CODE	00010300 – 0003FFFF	Application code segment

Table 1. Beginning of EM memory map by segments.

	**Note:** This is only an example of how memory is mapped in boot, only PRI_INTVEC, PRI_BOOT_CODE, INTVEC and CODE start addresses are always same.

8.4.2 ALGORITHM CODE MEMORY MAP

Segment	Address	Description
ALG_MAIN_SEG	00800020 - 00800E05	Algorithm code
CONST	00800E06 - 00801113	Algorithm code constant data area

Table 2. Beginning of EM memory map by segments

	**Note:** This is only an example of how memory is mapped in flash, only the ALG_MAIN_SEG start address is always the same.

8.5 CREATING ALGORITHM CODE STEP BY STEP

Step 1

Compile the algorithm code from the source files. A list of source files are in Table 3.

Table 3. Algorithm code source files

File	Description
b_boot.c	Algorithm code source file
b_boot.h	Algorithm code header file
boot.xcl	Linker control file for algorithm code
flash_dl.c	Flashing code source file
flash_dl.h	Flashing code header file

Step 2

Use the Hextransfer PC program to convert alg_code.h from the Motorola S formatted image file.

Command line parameters are:

/H Help switch, show help
/M Motorola switch, defines Motorola S format
/MAP Defines Hextransfer to use MAP file

Examples:

```
Hextransfer test.a37 /M /MAP
```

Generates header file from test.a37, start and end addresses are found in the test.map file.

```
Hextransfer test.a37 /M 800020 8001FF
```

Generates a header file from test.a37, start and end addresses are in the command line.

```
Hextransfer /h
```

Shows help for Hextransfer.

Step 3

Include the alg_code.h header file with the boot code project and compile the project.