

FreeMASTER for Embedded Applications

User Guide



Table of Contents

| Paragraph Number | | Page Number |
|--|--|-------------|
| Chapter 1 Introduction | | |
| 1.1 | Overview | 1 |
| 1.2 | Supported Platforms | 1 |
| 1.2.1 | Going Around UART and SCI..... | 1 |
| 1.3 | Where to Find the Latest Version | 1 |
| 1.4 | FreeMASTER Features | 1 |
| Chapter 2 QUESTIONS and ANSWERS | | |
| 2.1 | Why do I need it? | 3 |
| 2.2 | What does it do? | 3 |
| 2.3 | Why is it such a great demonstration tool? | 3 |
| 2.4 | What could I do with it if I follow the instructions? | 3 |
| 2.5 | How is it connected to a target development board? | 3 |
| 2.6 | What are all of these dialog boxes for? | 3 |
| 2.7 | How does a project relate to my application? | 4 |
| 2.8 | How do I set up remote control? Why would I want to? | 4 |
| 2.9 | What is the Watch-grid? | 4 |
| 2.10 | What is the Recorder? | 4 |
| 2.11 | What is the Oscilloscope? | 4 |
| Chapter 3 INSTALLATION | | |
| 3.1 | System Requirements | 5 |
| 3.2 | Enabling a FreeMASTER Connection on the Target Application | 5 |
| 3.3 | How to Install | 5 |
| Chapter 4 FreeMASTER USAGE | | |
| 4.1 | Application Window Description | 6 |
| 4.1.1 | Project Tree | 8 |
| 4.1.2 | Detail View..... | 17 |
| 4.1.3 | Watch-Grid..... | 19 |
| 4.2 | Variables | 19 |
| 4.2.1 | Generating Variables..... | 22 |
| 4.3 | Commands | 22 |
| 4.4 | Importing Project Files | 26 |
| 4.5 | Menu description | 28 |
| 4.5.1 | File Menu..... | 28 |
| 4.5.2 | Edit Menu..... | 28 |
| 4.5.3 | View Menu..... | 29 |
| 4.5.4 | Explorer Menu | 29 |
| 4.5.5 | Scope Menu..... | 29 |
| 4.5.6 | Item Menu..... | 29 |

| | | |
|-------|-------------------|----|
| 4.5.7 | Project Menu..... | 30 |
| 4.6 | Toolbars | 30 |
| 4.6.1 | Toolbar..... | 30 |
| 4.6.2 | Watch Bar | 30 |

Chapter 5 PROJECT OPTIONS

| | | |
|-------|--|----|
| 5.1 | Communication | 31 |
| 5.2 | Symbol Files | 32 |
| 5.2.1 | Regular Expression-based MAP File Parser | 33 |
| 5.3 | Packing Resource Files into Project File | 34 |
| 5.3.1 | Resource Files Manager..... | 36 |
| 5.4 | HTML Pages | 36 |
| 5.5 | Demo Mode | 37 |

Chapter 6 HTML and SCRIPTING

| | | |
|--------|---|----|
| 6.1 | Special HTML Hyperlinks | 39 |
| 6.2 | FreeMASTER ActiveX Object | 40 |
| 6.3 | FreeMASTER ActiveX Object Methods | 42 |
| 6.3.1 | GetAppVersion | 42 |
| 6.3.2 | OpenProject..... | 43 |
| 6.3.3 | StartStopComm | 44 |
| 6.3.4 | IsCommPortOpen..... | 45 |
| 6.3.5 | IsBoardDetected..... | 46 |
| 6.3.6 | StartStopComm | 47 |
| 6.3.7 | GetHtmlDocument | 48 |
| 6.3.8 | SendCommand..... | 49 |
| 6.3.9 | SendCommandDlg | 50 |
| 6.3.10 | ReadVariable | 51 |
| 6.3.11 | WriteVariable | 52 |
| 6.3.12 | ReadMemory | 53 |
| 6.3.13 | WriteMemory | 54 |
| 6.3.14 | ReadXxxArray..... | 55 |
| 6.3.15 | WriteXxxArray..... | 57 |
| 6.3.16 | ReadXxxVariable | 58 |
| 6.3.17 | WriteXxxVariable | 59 |
| 6.3.18 | GetCurrentRecorderData..... | 60 |
| 6.3.19 | GetCurrentRecorderSeries | 61 |
| 6.3.20 | StartCurrentRecorder | 62 |
| 6.3.21 | StopCurrentRecorder..... | 63 |
| 6.3.22 | GetCurrentRecorderState..... | 64 |
| 6.3.23 | RunStimulators | 65 |
| 6.3.24 | StopStimulators | 66 |
| 6.3.25 | LocalFileOpen..... | 67 |
| 6.3.26 | LocalFileClose | 68 |
| 6.3.27 | LocalFileWriteString..... | 69 |

| | | |
|--------|--|-----|
| 6.3.28 | LocalFileReadString | 70 |
| 6.3.29 | GetSymbolInfo | 71 |
| 6.3.30 | GetStructMember | 72 |
| 6.3.31 | GetAddressInfo | 73 |
| 6.3.32 | DefineSymbol | 74 |
| 6.3.33 | DeleteAllScriptSymbols | 75 |
| 6.3.34 | SubscribeVariable | 76 |
| 6.3.35 | UnSubscribeVariable | 77 |
| 6.3.36 | SelectItem | 78 |
| 6.3.37 | DefineVariable | 79 |
| 6.3.38 | DefineScope | 80 |
| 6.3.39 | DefineRecorder | 82 |
| 6.3.40 | IsBoardWithActiveContent | 83 |
| 6.3.41 | EnumHrefLinks | 84 |
| 6.3.42 | EnumProjectFiles | 85 |
| 6.3.43 | PipeOpen | 86 |
| 6.3.44 | PipeClose | 87 |
| 6.3.45 | PipeFlush | 88 |
| 6.3.46 | PipeSetDefaultTxMode | 89 |
| 6.3.47 | PipeSetDefaultRxMode | 90 |
| 6.3.48 | PipeSetDefaultStringMode | 91 |
| 6.3.49 | PipeGetRxBytes | 92 |
| 6.3.50 | PipeGetTxBytes | 93 |
| 6.3.51 | PipeGetTxFree | 94 |
| 6.3.52 | PipeGetRxBufferSize | 95 |
| 6.3.53 | PipeGetTxBufferSize | 96 |
| 6.3.54 | PipeWriteString | 97 |
| 6.3.55 | PipeWriteXxxArray | 98 |
| 6.3.56 | PipeReadString | 99 |
| 6.3.57 | PipeReadXxxArray | 100 |
| 6.3.58 | EnumVariables | 102 |
| 6.3.59 | EnumSymbols | 103 |
| 6.3.60 | GetDetectedBoardInfo | 104 |
| 6.3.61 | GetCommPortInfo | 105 |
| 6.4 | FreeMASTER ActiveX Properties | 106 |
| 6.5 | FreeMASTER ActiveX Object Events | 108 |
| 6.5.1 | OnRecorderDone | 108 |
| 6.5.2 | OnCommPortStateChanged | 108 |
| 6.5.3 | OnBoardDetected | 108 |
| 6.5.4 | OnVariableChanged | 108 |
| 6.6 | Examples | 109 |
| 6.6.1 | VB Script Embedded in HTML Page | 109 |
| 6.6.2 | JScript Embedded in HTML Page | 109 |
| 6.6.3 | VBA Script in Excel | 111 |
| 6.6.4 | Matlab m-script | 112 |

Table of Contents

| | |
|-----------------------------------|-----|
| Appendix A | 113 |
| Appendix B Revision History | 114 |

List of Figures

| Figure Number | Page Number |
|---|-------------|
| Figure 4-1. Initial Application Window | 6 |
| Figure 4-2. Application Window | 7 |
| Figure 4-3. Project Block Properties - Main Page | 8 |
| Figure 4-4. Project Block Properties - Watch Page | 9 |
| Figure 4-5. Project Block Properties - App. commands Page | 9 |
| Figure 4-6. Scope properties - Main Page | 10 |
| Figure 4-7. Scope Properties - Set-up Page | 11 |
| Figure 4-8. Basic Oscilloscope Chart | 12 |
| Figure 4-9. Third Variable Added | 12 |
| Figure 4-10. Third Variable Added - The Result | 13 |
| Figure 4-11. Joining two Y-blocks | 13 |
| Figure 4-12. Joining two Y-blocks - The Result | 14 |
| Figure 4-13. Recorder Properties - Main Page | 15 |
| Figure 4-14. Recorder Properties - Setup Page | 16 |
| Figure 4-15. Recorder Properties - Trigger Page | 16 |
| Figure 4-16. Demo Application Control Page | 17 |
| Figure 4-17. HTML Control Page Examples | 18 |
| Figure 4-18. Variables list Dialog Box | 19 |
| Figure 4-19. Variable Dialog Box - Definition Tab | 20 |
| Figure 4-20. Variable Dialog Box - Modifying Tab | 21 |
| Figure 4-21. Generating Variables | 22 |
| Figure 4-22. Project Application Commands | 23 |
| Figure 4-23. Sending Application Command | 23 |
| Figure 4-24. Application Command - Definition Tab | 24 |
| Figure 4-25. Application Command - Arguments Tab, page 1 | 24 |
| Figure 4-26. Application Command - Arguments Tab, page 2 | 25 |
| Figure 4-27. Application Command - Return Codes Tab | 26 |
| Figure 4-28. Import Project Tree Items | 27 |
| Figure 4-29. Import Project Objects | 27 |
| Figure 4-30. File Menu | 28 |
| Figure 4-31. Export graph image Dialog | 28 |
| Figure 4-32. View Menu | 29 |
| Figure 4-33. Explorer Menu | 29 |

List of Figures

| | |
|--|-----|
| Figure 4-34. Scope Menu | 29 |
| Figure 4-35. Project Menu | 30 |
| Figure 4-36. Toolbar | 30 |
| Figure 4-37. Watch Bar | 30 |
| Figure 5-1. Communication Options | 31 |
| Figure 5-2. Symbol Files Options | 32 |
| Figure 5-3. Regular Expression-based xMap File Parser | 34 |
| Figure 5-4. Testing Your Regular Expression | 34 |
| Figure 5-5. Pack Directory Options | 35 |
| Figure 5-6. Pack Directory Options, Example 2 | 35 |
| Figure 5-7. Resource Files Manager | 36 |
| Figure 5-8. HTML Pages Options | 37 |
| Figure 5-9. Demo Mode Options | 37 |
| Figure 5-10. Exit Demo Mode Confirmation Dialog | 38 |
| Figure 6-1. Registering the FreeMASTER Object in VBA Environment | 111 |

Chapter 1 Introduction

1.1 Overview

This User Manual describes the FreeMASTER application developed by Freescale engineers to allow control of an embedded application from a graphical environment running on a PC. The application was initially created for developers of hard real-time motor control applications, but many users found it very useful for their custom development.

The FreeMASTER 2.0 application is fully backward compatible with previous 1.4 and even older “PC Master” 1.0 versions.

1.2 Supported Platforms

The PC-side FreeMASTER application can be installed on any Microsoft Windows-based systems starting with Windows 2000. For the embedded side, there were several different serial communication (SCI) drivers for each supported platform before 2006. The drivers differed in a programming style, other software drivers requirements, and the way the drivers were configured.

In March 2006, Freescale released a SCI driver common for majority of Freescale microcontroller platforms. As of today, the driver supports MC56F800E, HC08, HCS08, HC12, S12, S12X, S12Z, ColdFire, Power Architecture and Kinetis families of microcontrollers. Although it still contains some platform-specific functions, the vast majority of the driver code, the documentation, and the way how the driver is configured is the same across all supported platforms.

1.2.1 Going Around UART and SCI

FreeMASTER installation comes with several plug-in modules, enabling it to access the target hardware over an alternative communication interface.

The **BDM Communication Plug-in** enables a basic memory access operations to be performed by the FreeMASTER with the HCS08, HC12/HCS12/X, ColdFire and Kinetis platforms without any target CPU intervention. In other words, no embedded-side communication driver is needed, and the FreeMASTER is still able to perform its basic tasks, which are reading and writing the target memory. This plug-in supports the P&E Multilink BDM cables and several BDM interfaces based on open source firmware.

The **Packet-driven BDM Communication plug-in** can be used as an additional layer on top of the BDM plug-in to enable high-level protocol commands like Recorder, TSA or memory protection. This plug-in uses the BDM interface to exchange communication protocol frames with the target driver, rather than just accessing the data directly. The Serial Driver is needed to run on the target in this case, and it should be configured properly for the packet-driven BDM interface.

The **JTAG/EOnCE Communication Plug-in** enables a fully-featured communication with the 56F800E hybrid microcontrollers over the JTAG interface cable. The Serial Driver is needed to run on the target in this case, and it should be configured properly for the JTAG interface. The plug-in uses the Real-time Data Exchange feature of JTAG/EOnCE, which is very similar to the SCI communication.

The **FreeMASTER-over-CAN Plug-in** enables to use FreeMASTER services over a CAN interface. Also use this plug-in with the Serial Driver running on the target, configured for msCAN or FlexCAN module.

See more details about the communication plug-in modules in the read-me documents installed together with the latest FreeMASTER application.

1.3 Where to Find the Latest Version

There are two software setup packs available on the Freescale web pages. One is for the PC-side application (FMSTRSW), and one is for the new SCI driver for the embedded-side (FMSTRSCIDRV). Both packages are available at the FreeMASTER home page at <http://www.freescale.com/freemaster>.

1.4 FreeMASTER Features

- Graphical environment
- Easy to understand navigation
- Simple RS232 native connection and other options possible on selected platforms (BDM, JTAG, CAN,...)
- Real-time access to embedded-side C variables
- Visualization of real-time data in the Scope window
- Acquisition of fast data changes using the on-target Recorder
- Built-in support for standard variable types (integer, floating point, bit fields)

Introduction

- Value interpretation using custom defined text messages
- Several built-in transformations for real type variables
- Automatic C-application variable extraction from compiler output files (ELF/DWARF1/2, Text-based map files,...)
- Demo mode with password protection support
- HTML-based description or navigation pages
- ActiveX interface to enable VBScript or JScript control over embedded applications
- Remote Communication Server enabling a connection to target board over a network, including the Internet

Chapter 2 QUESTIONS and ANSWERS

First question: why place this topic immediately after an introduction? The reason is really quite practical. While writing this User Manual, the following questions were raised. Since the answers to these questions clarified terms and topics described further in the manual, it was decided to put this topic before those that are more detailed and perhaps less easily understood.

2.1 Why do I need it?

The primary goal of developing FreeMASTER software was to deliver a tool for debugging and demonstrating Motor Control algorithms and applications. The result was a tool with the versatility to be used for multipurpose algorithms and applications. Some real-world uses include:

- Real-time debugging - FreeMASTER allows users to debug applications in true real-time through its ability to watch variables. Moreover, it allows debugging at the algorithm level, which helps to shorten the development phase.
- Diagnostic tool - FreeMASTER remote control capability allows it to be used as a diagnostic tool for debugging customer applications remotely across a network.
- Demonstrations - FreeMASTER is an outstanding tool for demonstrating algorithm or application execution and variable outputs.
- Education - FreeMASTER may be used for educational purposes. Its application control features allow students to play with the application in demonstration mode, learning how to control program execution.

2.2 What does it do?

FreeMASTER communicates with the target system application via serial communication to read and write application internal variables. FreeMASTER provides the following visualization features for displaying variable information in a friendly format:

- Oscilloscope - provides monitoring/visualization of application variables in the same manner as a classical oscilloscope with a CRT. In this case, monitoring rates are limited by the serial communication speed.
- Recorder - provides monitoring/visualization of application variables that are changing at a rate faster than the sampling rate of the oscilloscope. While the Scope periodically reads FreeMASTER variable values and plots them in real-time, the Recorder is running on the target board. Variable values are sampled into a memory buffer on the board, then the sampled data is downloaded from board to FreeMASTER. This mechanism allows a much shorter sampling period and enables sampling and plotting of very quick actions.

2.3 Why is it such a great demonstration tool?

The embedded-side algorithm can be demonstrated in one block, or divided into several blocks, depending on which possibility better reflects the algorithm structure. Each block's input parameters may be explored to observe how they affect output parameters. Each block has a description tab for explaining algorithm details using multimedia-capable and scriptable HTML format.

2.4 What could I do with it if I follow the instructions?

Using the demo project included with the embedded-side implementation, it is easy to learn how to use FreeMASTER by playing with the project's defined blocks and parameters. The demo project allows you to understand how to control the application as well. You can go into details of each item, check its properties, change parameters, and determine how each can be used in your application. For a detailed explanation of the parameters, see [Chapter 4, "FreeMASTER USAGE"](#).

2.5 How is it connected to a target development board?

FreeMASTER requires a serial communication port on the target development hardware. Connection is made using a standard RS-232 serial cable. On one side, the cable is plugged into the PC serial port (COM1, COM2 or other), and on the opposite side, into the target development board's serial connector.

In addition to RS232 link, the custom communication plug-in modules can be written and used by FreeMASTER. There are communication plug-ins available for CAN Calibration Protocol, JTAG Real-time Data Exchange port on 56F800E, BDM interface on HCS08/12 devices, etc.

2.6 What are all of these dialog boxes for?

In [Chapter 4, "FreeMASTER USAGE"](#), you will see pictures with dialog boxes. These dialog boxes are used as a questionnaire, where you will enter parameters describing, for example, one algorithm block or application variable and its visualization.

2.7 How does a project relate to my application?

There can be many FreeMASTER projects related to a single target-board application. For example, three specific FreeMASTER projects can work with the same board application to provide three different purposes:

- to provide information used during debug process
- to provide service maintenance capabilities
- may be used for learning about your application during operator training phase

2.8 How do I set up remote control? Why would I want to?

For remote control, you need at least two computers connected via a network, one running the standalone mini-application called FreeMASTER Remote Communication Server, and the second running the standard FreeMASTER application. The target development board is then connected to the computer running FreeMASTER Server.

Remote control operation is valuable for performing remote debugging or diagnostics. An application may be diagnosed remotely by connecting the target development board to the remote PC, and then running the FreeMASTER locally with a service project for this customer's application.

2.9 What is the Watch-grid?

The Watch-grid is one of the panes in the FreeMASTER application window. It shows selected application variables and their content in human readable format. The application variables displayed are selected separately in the block property settings of each project block.

2.10 What is the Recorder?

The Recorder is created in software on the target development board, and stores changes of variables in real-time. You can define the list of variables which will be recorded by the embedded-side timer periodic interrupt service routine. After the requested number of variable samples are stored within the Recorder buffer on the target board, it is downloaded from the board and displayed in FreeMASTER Recorder pane as a graph. The main advantage of the Recorder is the ability to sample very fast actions.

2.11 What is the Oscilloscope?

FreeMASTER Oscilloscope is similar to the classical hardware oscilloscope. It shows graphically selected variables in real-time. The variable values are read from the board application in real-time through the serial communication line. The oscilloscope GUI looks similar to the Recorder, except that the sampling speed of variables is limited by the communication data link.

Chapter 3 INSTALLATION

3.1 System Requirements

The FreeMASTER application requirements can be easily satisfied by almost any Windows-based host PC available today. It was tested with the latest version of Microsoft Windows 7, 8 and Windows 10, however it was originally designed for and should run smoothly even with Microsoft Windows 98 and Internet Explorer 5.5.

Operating system: Microsoft Windows 7 or later

Required software: Internet Explorer 10 or later installed.

Hard drive space: 20MB

Other hardware requirements: Mouse, serial RS-232 or USB port for local control, network access for remote control

3.2 Enabling a FreeMASTER Connection on the Target Application

To enable the FreeMASTER connection to the target board application, follow the instructions provided with the embedded-side driver, the FMASTERSCIDRV. The recommended and fastest way to start using FreeMASTER is by trying the sample application. Note that the sample application name may still refer to the "PC Master" software, which is the previous name of the FreeMASTER tool. FreeMASTER is fully backward compatible with PC Master.

3.3 How to Install

The FreeMASTER application is distributed as a standalone, single file, self-extracting, executable file. Download the installer file from web site, run it and proceed according to instructions given on the screen.

Chapter 4 FreeMASTER USAGE

4.1 Application Window Description

When the application is started, the main window is displayed on the screen. When there is no project loaded, the welcome page is displayed in the main pane of the window. The initial look of the main window is shown in Figure 4-1.

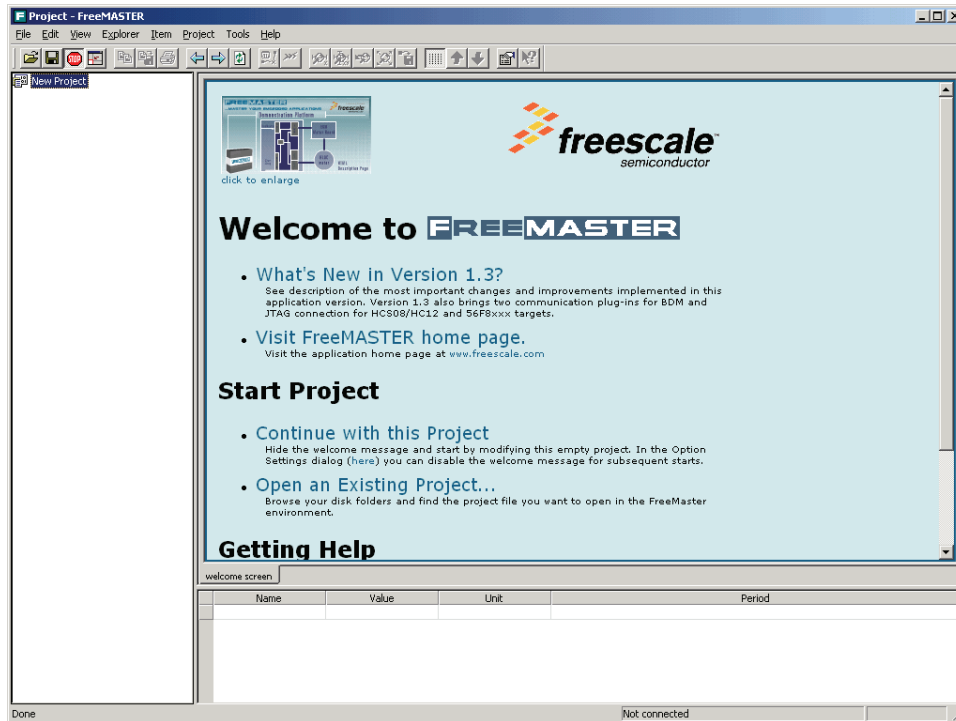


Figure 4-1. Initial Application Window

The welcome page contains links to the documentation and to the application help. There are also several other links corresponding to the standard menu commands (for example, the *Open project* command).

In the remaining part of this chapter, an application usage is demonstrated on an example of a simple demo application, which is a part of the SCI Driver installation (FMMASTERSCIDRV).

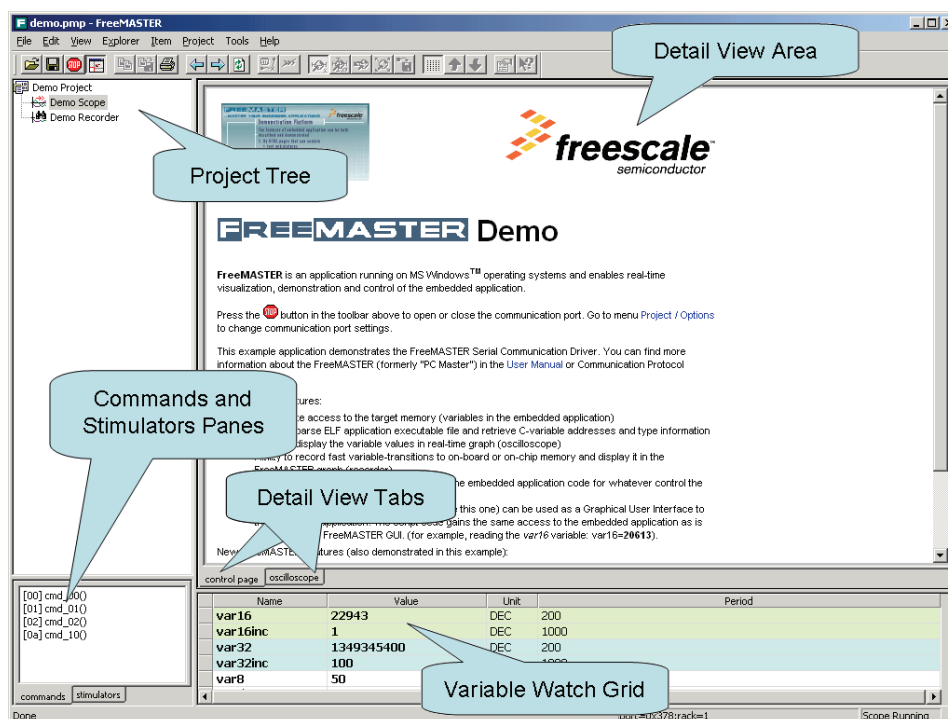


Figure 4-2. Application Window

As shown in Figure 4-2, the application window consists of four panes. Three panes are always displayed: the *Project Tree* pane, the *Detail View* pane and the *Variable Watch* pane. The *Commands/Stimulators* “fast access” pane may be shown or hidden as described later in Section 4.5.3, “View Menu.”

The *Project Tree* pane contains a logical tree structure of the application being monitored/controlled. Users can add project sub-blocks, Scope, and Recorder definitions to the project block in a logical structure to form a *Project Tree*. This pane provides point and click selection of defined *Project Tree* elements.

The *Detail View* pane dynamically changes its content depending on the item selected in the *Project Tree*. Depending on the type of the item selected in the tree, this pane also provides several tabs with sub-pages of additional information associated to the item.

- *control page* = An HTML page created for controlling the target system
- *algorithm block description* = An HTML page or another document whose URL is defined in the selected *Project Tree* item’s properties
- *current item help* = Another HTML document whose URL is defined in the Scope or Recorder properties
- *oscilloscope* = A real-time graph displaying application variables as defined in the Scope properties
- *recorder* = A graph displaying recorded application variables as defined in the Recorder properties
- The *control page*, when defined, is available for all *Project Tree* items to allow the user to control the board at any time. The content of the *algorithm block description* page changes with the *Project Tree* item selected. When a Scope or Recorder is selected from the *Project Tree*, the *current item help* and *oscilloscope/recorder* chart pages are also available.

The *Variable Watch* pane contains the list of variables assigned to the watch. The pane displays the immediate variable values, and also enables you to change them (if this is enabled in the variable definition).

All the information related to one application is stored in a single project file with the extension “.pmp”. This information includes the project settings and options, the *Project Tree*, *Detail View* HTML pages, real-time chart definitions, watch interface settings, variables and commands, stimulators and more.

4.1.1 Project Tree

When a new project is created, the *Project Tree* window contains an empty structure with just one root project block called “*New Project*”. You can then change properties of this block or add sub-blocks, Scopes or Recorders to the structure.

Property changes and *Project Tree* additions can be done in two ways:

- Select an item in the *Project Tree* and right mouse click to use the local menu
- Select an item in the *Project Tree* and select main menu “*Item*” pull-down

4.1.1.1 Project Block and Sub-block

The *Project block* typically covers an integral component of the application or algorithm being demonstrated with FreeMASTER. *Sub-blocks* may be added, should the user care to break the algorithm into multiple blocks. Each block has its own *algorithm block description page*, *watch variables* and *commands*. All of these can be defined in the *Project block properties* dialog, as shown in [Figure 4-3](#).

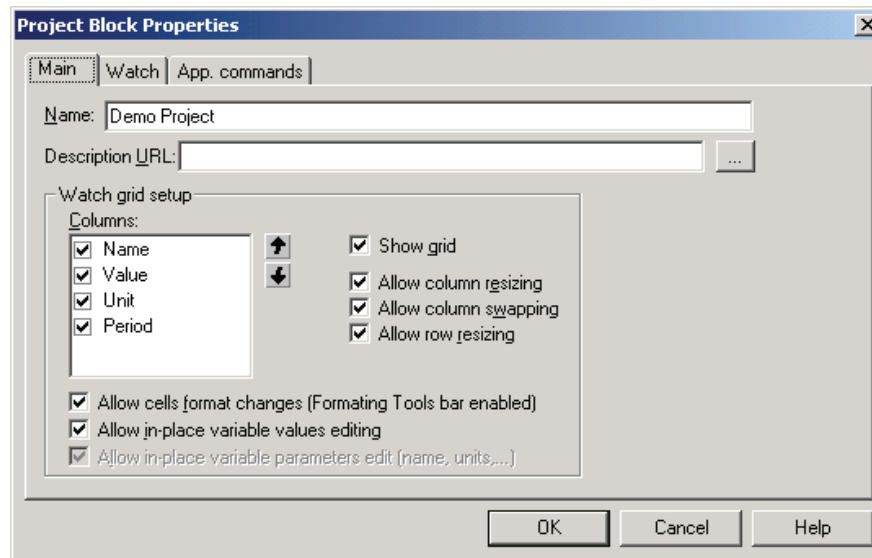


Figure 4-3. Project Block Properties - Main Page

The *Main* page contains the following user configuration items.

- *Name* = Name of Project block that will be displayed in the *Project Tree*
- *Description URL* = Select a description URL or a path to *.htm* or *.html* file to be shown in the *Detail View* pane under the *algorithm block description* tab. This file may be created with any HTML editor such as MS Front Page Express or Netscape Composer. In the demo application used as our example, the description page is left empty, causing the “Algorithm Block Description” tab to be hidden. See [Section 4.1.2, “Detail View](#) for more details.
- *Watch-grid setup* = You can select columns to display in the Watch-grid (Name, Value, Unit, Period); specify the column order using the Up and Down arrow buttons; check the grid behavior options (column resizing/swapping, row resizing); allow format changes to grid cells with Toolbar (see [Section 4.6.2, “Watch Bar](#)), and edit in-place variable values by checking the next option boxes.

The *Watch* page shown in [Figure 4-4](#) selects which FreeMASTER project variables are to be watched in the context of this project block.

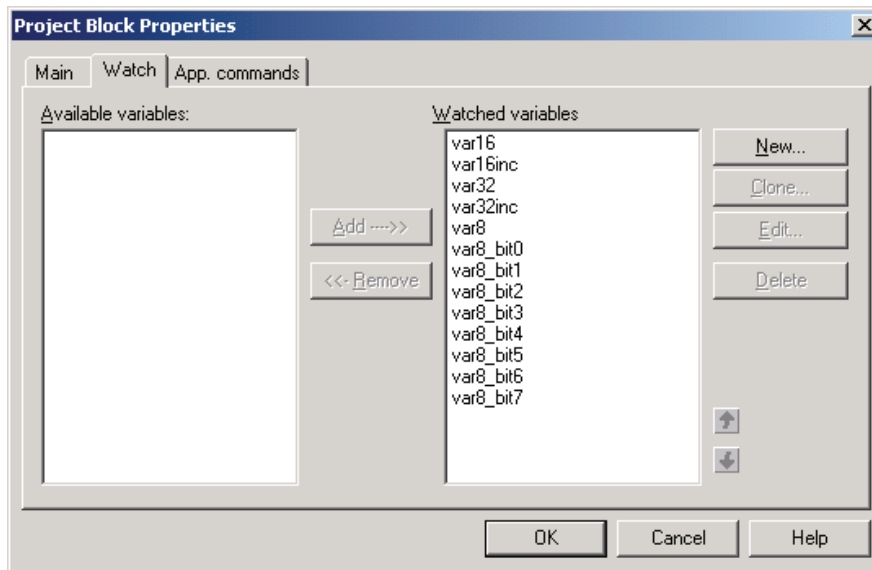


Figure 4-4. Project Block Properties - Watch Page

The variables in the *Watched variables* list are the project variables which are currently selected for watching in the Watch-grid. The *Available variables* list contains the remaining available project variables not selected for watching with the current block item selected in the tree. You can use the following buttons:

- *Add/Remove* = Moves variables into and out of the *Watched variables* window
- *New* = Creates a new variable (see [Section 4.2, “Variables”](#))
- *Clone* = Creates a new variable based on a copy of the selected variable
- *Edit* = Changes the selected variable properties
- *Delete* = Deletes the selected variable from the project
- *Up/Down arrows* = Sets the display order of the watched variables in the Watch-grid

FreeMASTER communicates with the board application by reading/writing variables and/or sending so called “application commands” (see [Section 4.3, “Commands”](#)). As the variable appearance in the Watch-grid can be dependent on the block selected in the *Project Tree* pane, the availability of application commands can also be dependent on the selected block. The *App. commands* page, shown in [Figure 4-5](#), sets which commands will be available in the *Fast Access* pane in the context of this project block and also enables the management of application commands.

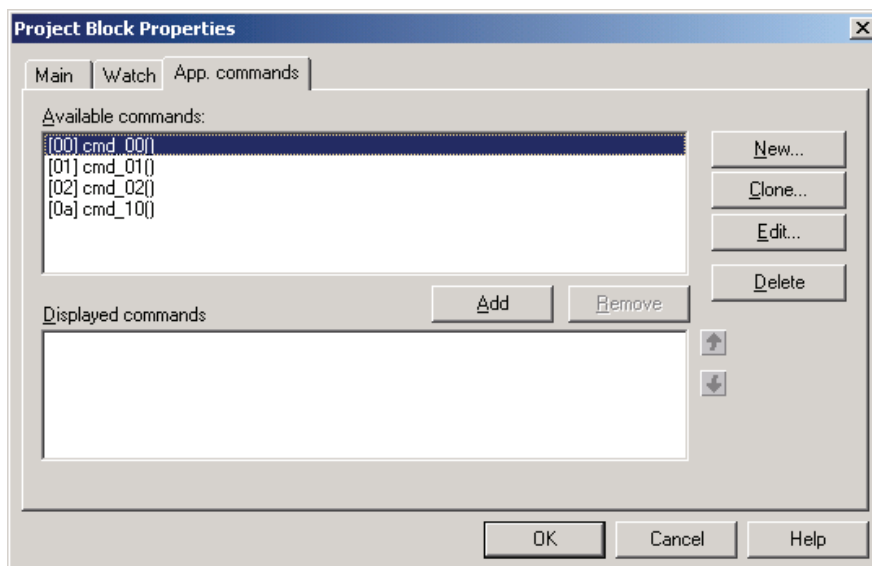


Figure 4-5. Project Block Properties - App. commands Page

The commands are listed in the *Available commands* window. Use the *Add* button to move a command into the *Displayed commands* window and to make it available in the *Fast Access* pane. Use the *Remove* button for reverse operation. You can use the following buttons:

- *New* = Creates a new command (see Section 4.3, "Commands")
- *Clone* = Creates a new command based on a copy of the selected command
- *Edit* = Changes the selected command properties
- *Delete* = Deletes the selected command
- *Up/Down arrows* = Sets the display order of commands in the *Fast Access* pane

4.1.1.2 Scope

The *Scope* item in the *Project Tree* structure defines a real-time oscilloscope chart to be shown in the *Detail View* pane. The Scope properties window, shown in Figure 4-6, allows you to configure the appearance and characteristics of the scope chart.

The *Main* page contains these user configuration items:

- *Name* = The name of the Scope item that will be displayed in the *Project Tree*
- *Description URL* = Specify the URL of the document or local path to a file to be shown in the *Detail View* pane under the *current item help* tab. This file may be created with any HTML editor, such as MS Front Page Express or Netscape Composer, and should explain the chart variables and settings to the user.
- *Scope global properties* = Common properties for all scope variables
- *Period* = Oscilloscope sampling period
- *Buffer* = The number of samples in one data subset in the chart
- *Legend location* = Set the visibility and location of the chart legend

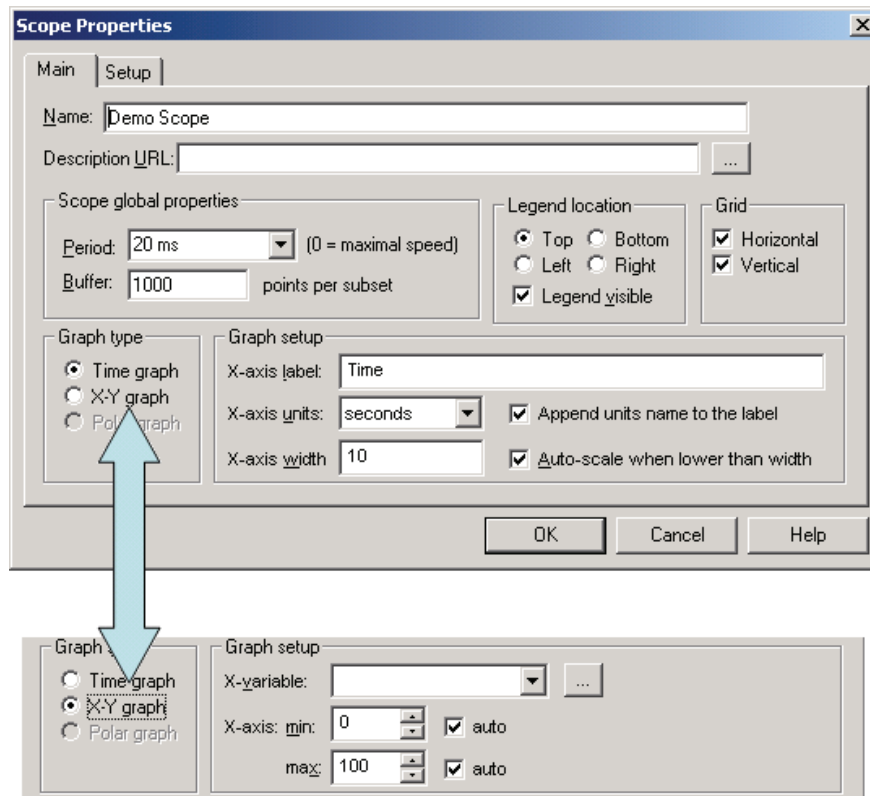


Figure 4-6. Scope properties - Main Page

- *Grid* = Choose the horizontal and/or vertical grid lines to be displayed in the chart
- *Graph type* = Select the mode of oscilloscope operation
- *Time graph* = A variable (*values* versus *time*) will be displayed in the chart
- *X-Y graph* = Inter-variable dependencies (*value* versus *value*) will be displayed in the chart
- *Graph setup* (for *Time graph*):

- *X-axis label* = Specify the name displayed for the X axis
- *X-axis units* = Select the axis units
- *X-axis width* = Specify the range of the X axis
- *Auto-scale X axis until width is reached* = Scales the axis width after Scope start when the length of subset is shorter than the *X axis width*
- *Graph setup* (for X-Y graph):
 - *X-variable* = Selects the variable whose values will be used for X axis values
 - *X-axis min* = Sets the X axis lower limit value (checks the *auto* box to enable X axis auto-scaling)
 - *X-axis max* = Sets the X axis upper limit value (checks the *auto* box to enable X axis auto-scaling)

The *Setup* page (Figure 4-7) is used to assign variables to be displayed on the oscilloscope chart. Up to eight *Chart vars* (seven in X-Y mode) may be selected for display in the chart and assigned to a maximum of five Y-blocks. Select one of eight positions and browse for a variable in the drop-down list below the list to set or change a variable at this position. Select the first (empty) item in the drop-down list to clear the selected position in the list. Each chart variable is assigned to a Y block.

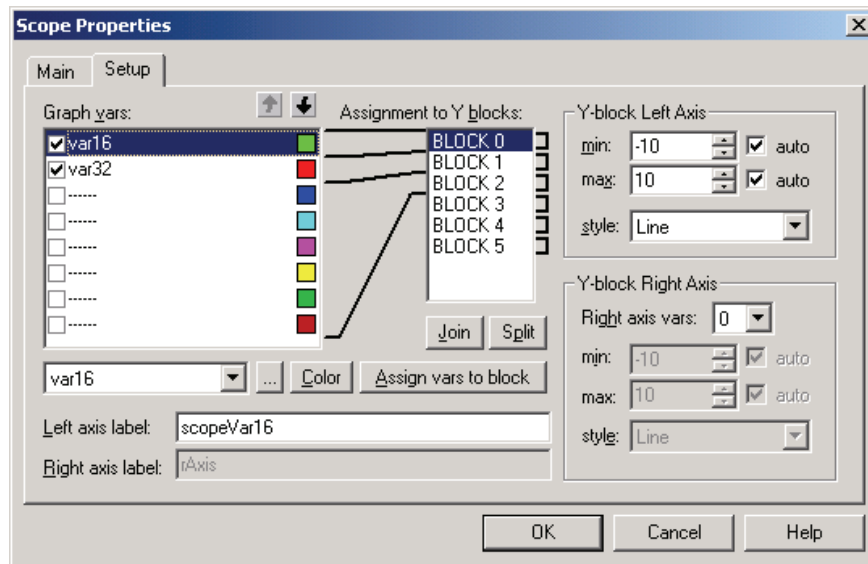


Figure 4-7. Scope Properties - Set-up Page

The Y block is a graph element represented by one left Y axis and, optionally, one right Y axis also. The Y blocks can be drawn separately, or overlapped in the graph.

- *Assign vars to block* button = Assigns successive selected *Chart vars* to a Y block. Select the chart variables you want to group into one Y block and press this button. A simple assignment of two variables into two separate Y blocks is shown in Figure 4-7.
- In the *Y-block Left axis* frame, set the axis range by specifying the *min* and *max* axis value, or check *auto* box to enable automatic minimum and/or maximum tracking. Select a *Style* of drawing the data subsets from the drop-down list box.
- Type the *Left axis label*, which will be assigned to a selected Y-block.

The resulting oscilloscope chart is displayed in Figure 4-8.

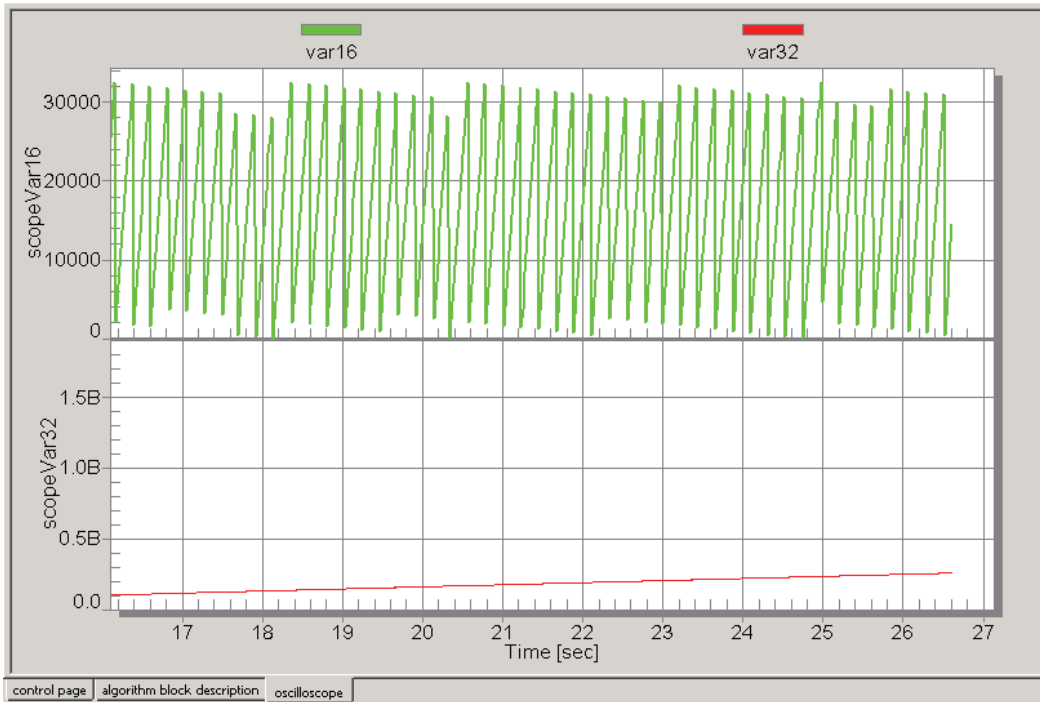


Figure 4-8. Basic Oscilloscope Chart

- When more than one variable are displayed in a single Y-block, a separate *Right axis* may be shown. For the selected Y-block, select the number of *Right axis vars* that will use the right axis within the Y-block. The value of *Right axis vars* specifies the number of variables (counting from the bottom of the *Chart vars* list) that are assigned to the right axis within the Y-block.
- As with *Left axis*, specify *min*, *max*, *style* and *axis label* for the right axis in the selected Y block.

In Figure 4-9, we added one variable to the second Y-block and set *Right axis vars* to one. This means that the added variable (*var32inc*) will be assigned to the Right axis labeled *var32 Increment*.

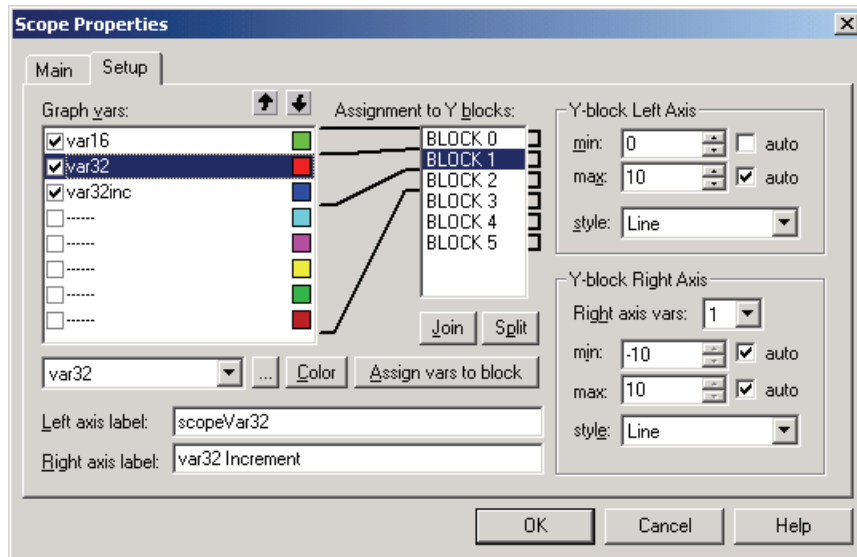


Figure 4-9. Third Variable Added

The *var32inc* variable value was changed manually from 100 to 1000 to achieve the waveform shown in Figure 4-10.

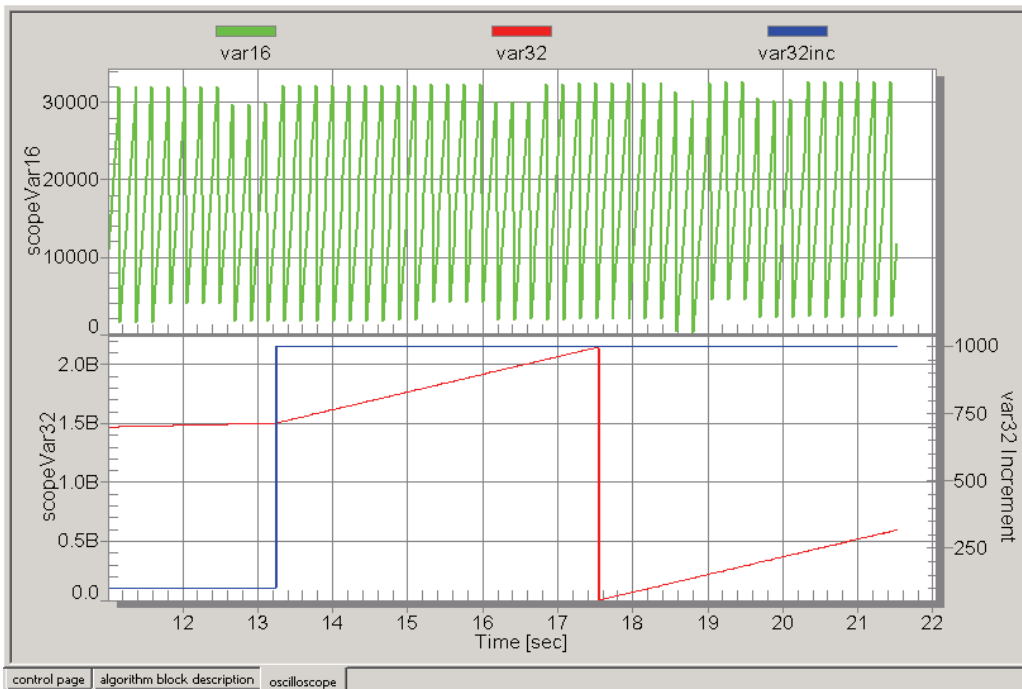


Figure 4-10. Third Variable Added - The Result

Figure 4-11 shows the two Y-blocks joined, using the “Join” button. Both Y-blocks will be plotted “overlapped”, causing multiple X axes to be drawn (two in our case).

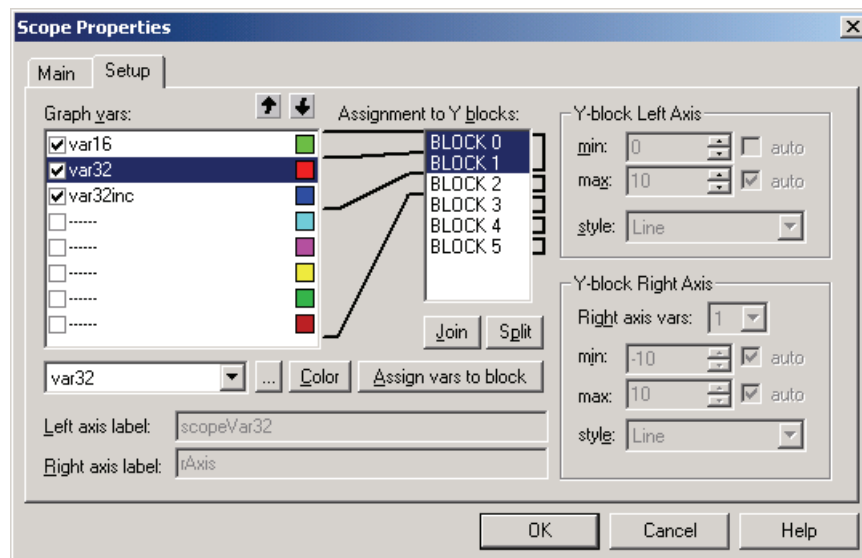


Figure 4-11. Joining two Y-blocks

The resulting chart is displayed in Figure 4-12

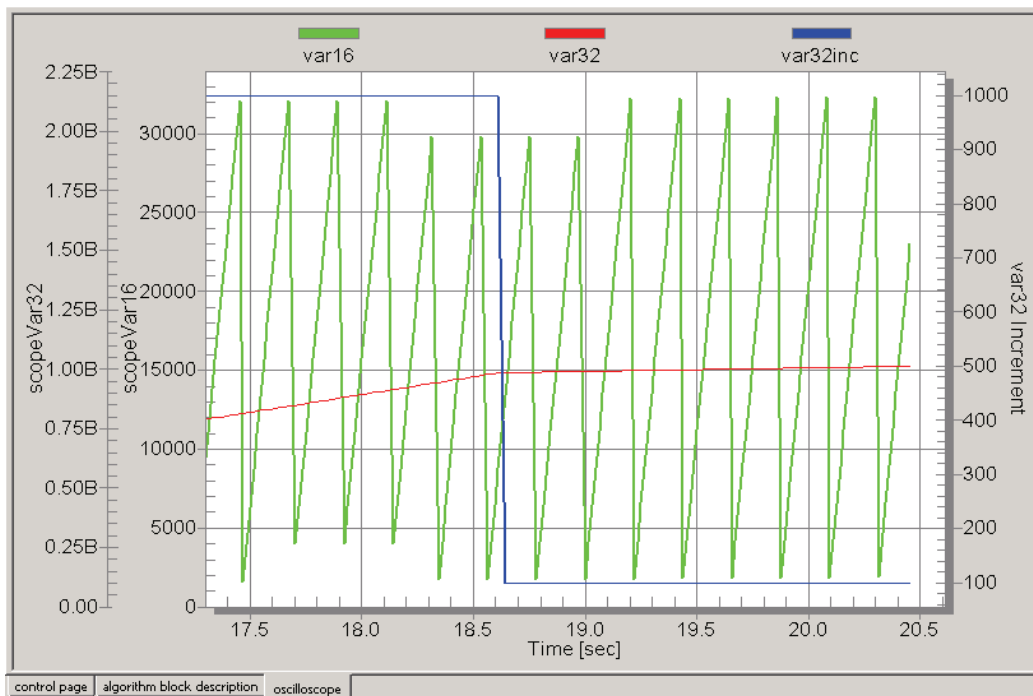


Figure 4-12. Joining two Y-blocks - The Result

4.1.1.3 Recorder

The *Recorder* item in the *Project Tree* structure defines a real-time recorder chart to be shown in the *Detail View* pane. While the *Scope* periodically reads variable values and plots them in real time, the *Recorder* is running on the target board, reads application variables, and sends them to the FreeMASTER tool in a burst mode fashion. The recorder variables are continually sampled and stored into a circular buffer in the target board application. When the trigger event is detected by the target, data samples are counted until the number of *Recorder samples* is reached. At this point, data is sent to the FreeMASTER application. This mechanism enables use of much shorter sampling period and enables sampling and plotting very quick actions.

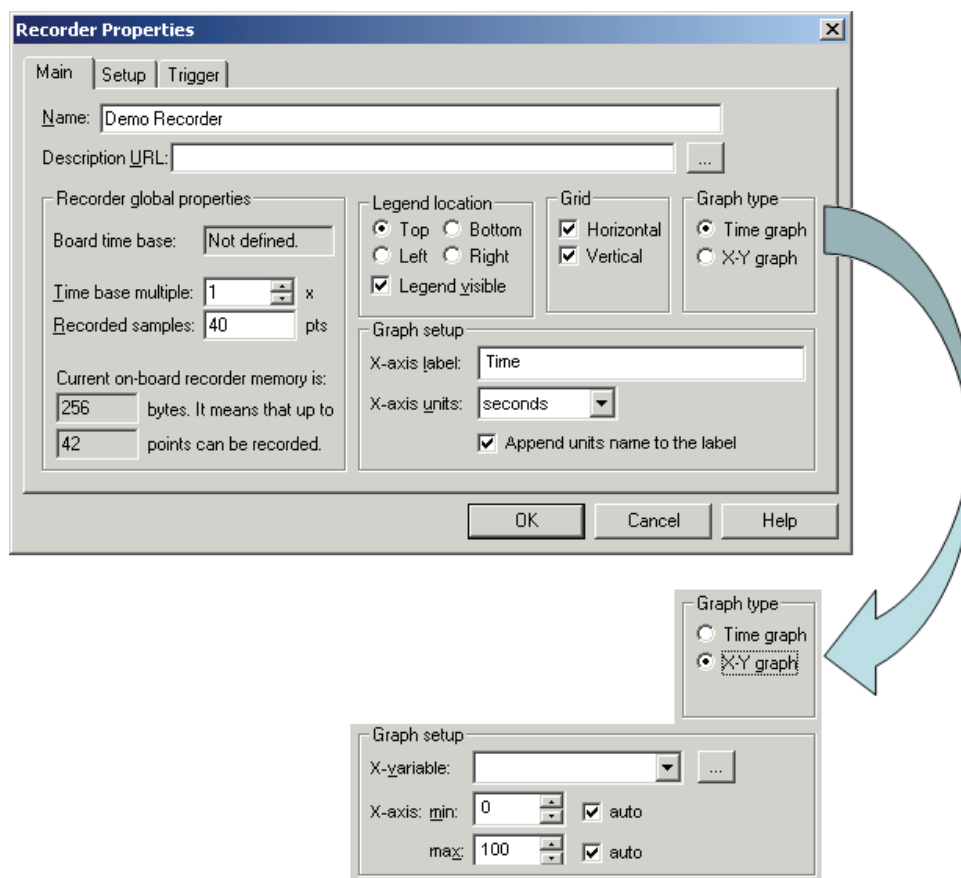


Figure 4-13. Recorder Properties - Main Page

The *Main* page contains the following user configuration items:

- *Name* = The name of the Recorder item that will be displayed in the *Project Tree*
- *Description URL* = Specify the document's URL or local path to a file to be shown in the *Detail View* pane under the *current item help* tab. This file may be created with any HTML editor, such as MS Front Page Express or Netscape Composer, and should explain the chart variables and settings to the user.
- *Recorder global properties* = Common properties for all Recorder variables
 - *Board time base* = A sampling period preset by the board application. In the SDK, the time base value can be adjusted by setting `PC_MASTER_RECORDER_TIME_BASE` in the `appconfig.h` file during the board application development.
 - *Time base multiple* = Sets an integer multiple of the time base to extend the sampling period used for recorder operation
 - *Recorded samples* = The number of samples buffered for one recorded subset
 - *On board recorder memory* = Displays the amount of on-board application memory allocated for recorder operation. Based on the memory size, recorded variables format, and the number of recorded variables, the maximum number of points which fit in the recorder's memory is calculated and displayed. The *Recorded samples* value set should be lower than this result.
- *Legend location* = Sets the visibility and location of the chart legend
- *Grid* = Chooses the horizontal and/or vertical grid lines to be displayed in the chart
- *Graph type* = Selects the mode of recorder operation
 - *Time graph* = A variable (*values versus time*) will be displayed in the chart
 - *X-Y graph* = Inter-variable dependencies (*value versus value*) will be displayed in the chart
- *Graph setup* (for *Time graph*):
 - *X-axis label* = Specify the name displayed for X axis
 - *X-axis units* = Selects the axis units
- *Graph setup* (for *X-Y graph*):

- *X-variable* = Selects the variable whose values will be used for X axis values
- *X-axis min* = Sets the lower limit value of the X axis (check the *auto* box to enable auto-scaling of the X axis)
- *X-axis max* = Sets the upper limit value of the X axis (check the *auto* box to enable auto-scaling of the X axis)

The *Setup* page of the *Recorder properties* dialog, shown in Figure 4-14, looks exactly the same as the appropriate page of the *Scope properties* dialog. For more information about how to add variables to the recorder chart and how to set up the chart itself, see Section 4.1.1.2, “Scope.”

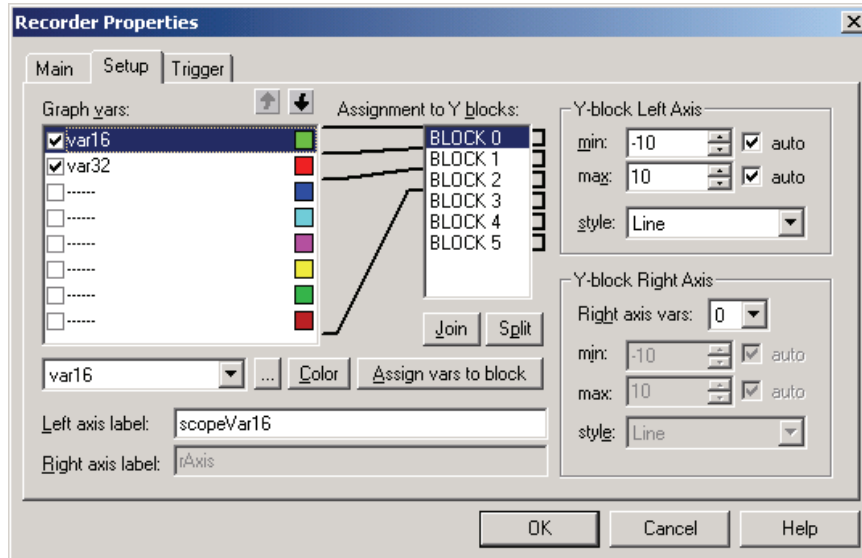


Figure 4-14. Recorder Properties - Setup Page

The *Trigger* page, shown in Figure 4-15, defines the Recorder start conditions. The trigger starts the Recorder when the *Trigger variable* exceeds the specified *Threshold value* with the selected type of slope, (positive = rising edge or negative = falling edge).

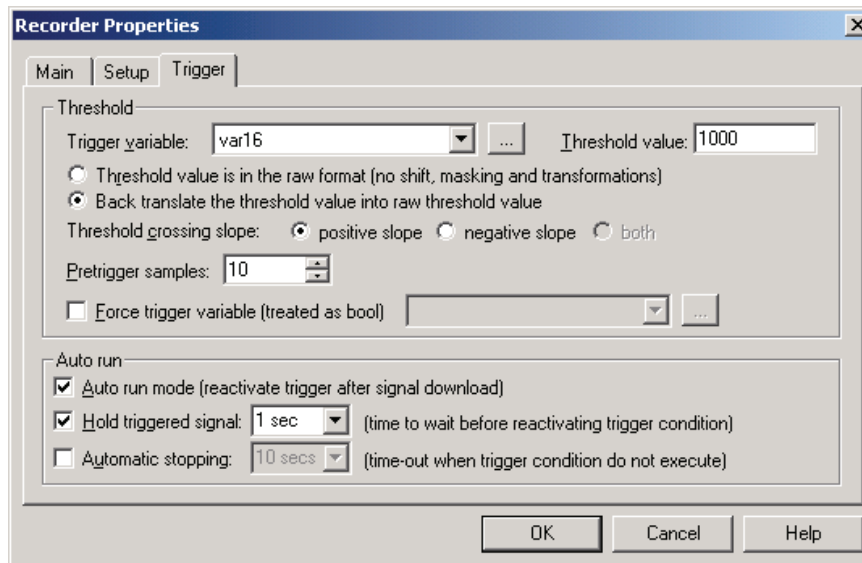


Figure 4-15. Recorder Properties - Trigger Page

- *Threshold* = Specify conditions for the trigger event
 - *Trigger variable* = Selects a variable to be tracked for the trigger event
 - *Threshold value* = The value of the variable being recorded that, when crossed, causes the trigger event. Specify a threshold value and select whether the value is in raw format (as in the board application) or whether it must be translated back into raw format (e.g., when a real-type transformation is defined for the variable and you want to apply an inverse transformation on the trigger value before it is set as threshold in the embedded application).
 - *Threshold crossing slope* = Selects the slope on which the threshold crossing is monitored

- *Pretrigger samples* = Specify the number of samples to save and display prior to the trigger event
- *Auto run* = Specifies conditions for reactivating the trigger for repeated recording
 - *Auto run mode* = Select to enable repeated recording. After detecting the trigger event, filling the buffer and downloading the buffer data to FreeMASTER, the trigger is automatically reactivated and new data is downloaded immediately after the next trigger event occurs.
 - *Hold triggered signal* = Check this box and specify how long to wait after one signal is displayed in the chart and before reactivating the trigger.
 - *Automatic stopping* = Check this box and specify the maximum time period for detecting the trigger event. If the event is not detected within the specified time, the sampling is unconditionally stopped, and actual buffer data is downloaded.

4.1.2 Detail View

The *Detail View* is a multipage pane. Availability of various pages in the *Detail View* depends on the type of item selected in the *Project Tree*.

The *control page*, when defined in project *Options* (5.4, “HTML Pages”), is available for all *Project Tree* items to allow the user to control the board at any time. The content of the *algorithm block description* page changes with the *Project Tree* item selected. When a Scope or Recorder item is selected in the *Project Tree*, the *current item help* and *oscilloscope/recorder* chart pages are also available.

4.1.2.1 Control Page

The *Control Page* is an HTML page created for board application control. It typically contains the scripts-enhanced form or forms which enable user-friendly control of the embedded application. The URL of the page or the path to the HTML file with a page source code can be specified in project *Options* dialog described in Section 5.4, “HTML Pages”.

The control page for the demo application used as an example is shown in Figure 4-16. Despite of its name, there are no “control” features utilized in this simple application. Still, it demonstrates a JScript scripting technique to display a variable value directly in the HTML-coded page. See more details on HTML and scripting in Section 6.2, “FreeMASTER ActiveX Object”.



Figure 4-16. Demo Application Control Page

A few more sophisticated control page screen shots are shown below. The applications shown here use some 3rd party Instrumentation components, inserted into the HTML code as embedded ActiveX objects. More screen shots are available on the FreeMASTER Home Page on the www.freescale.com (see Section 1.3, “Where to Find the Latest Version” for more details).

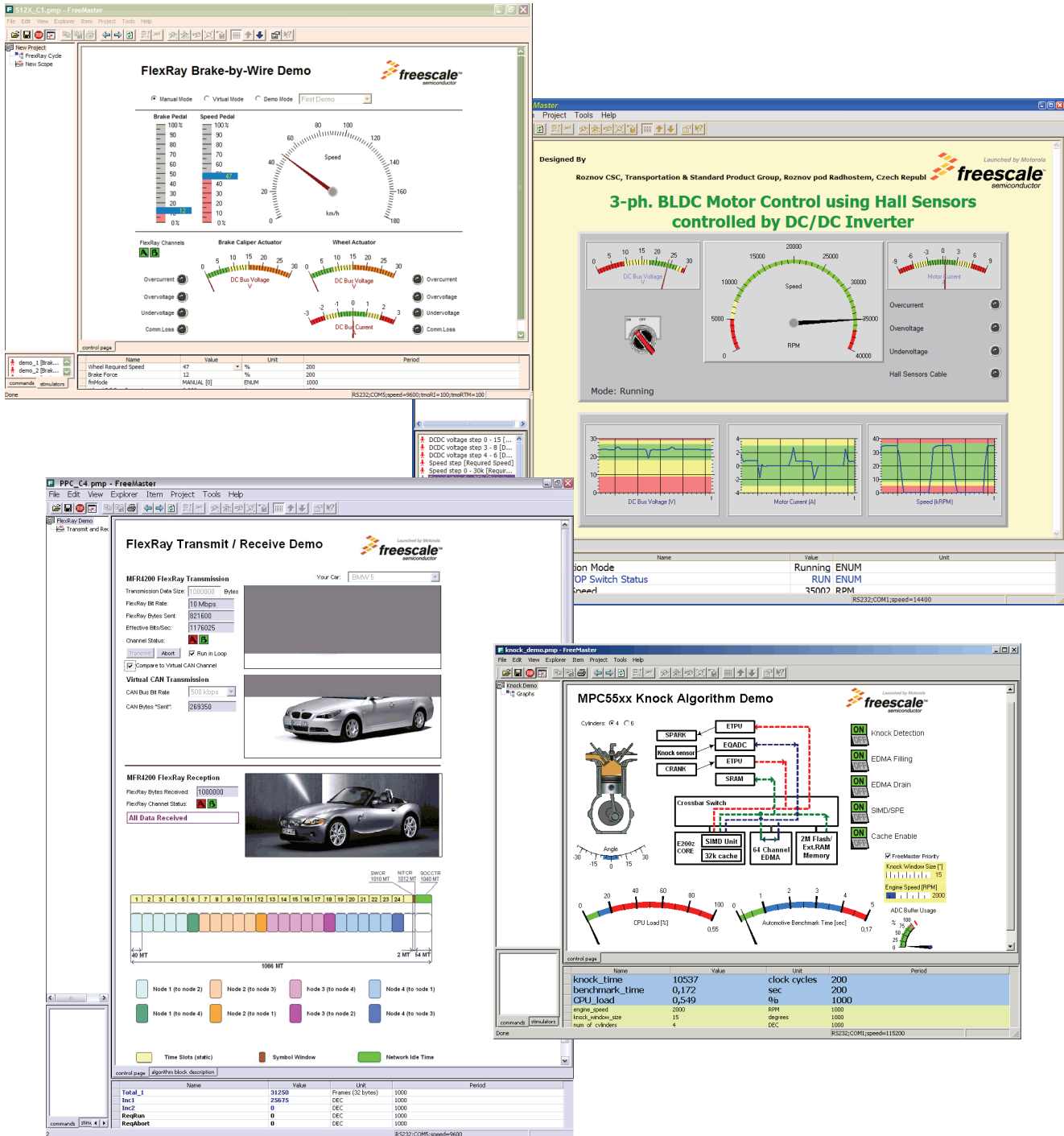


Figure 4-17. HTML Control Page Examples

4.1.2.2 Algorithm Block Description

The *Algorithm Block Description* page in the *Detail View* pane is designated for placing an HTML page describing the selected block functionality. The URL or local path to the source file is specified in block item properties dialog, described in Section 4.1.1.1, “Project Block and Sub-block. This page is displayed when it is defined and when the user selects the appropriate block item or any of its child scope or recorder items.

As the standard HTML page, this page can also contain the scripts or other controls, but it is not a common practice.

4.1.2.3 Current Item Help

The *Current item help* tab is designated for placing an HTML page describing the selected *Scope* or *Recorder*. This page should contain such information as definitions or use instructions and is specified as the Description URL.

4.1.2.4 Oscilloscope / Recorder

The *Oscilloscope page* in the *Detail View* pane contains the real-time chart representing tracked variables, as already shown above in [Section 4.1.1.2, "Scope](#). Similarly, the *Recorder page* contains the chart created from the recorded data as described in [Section 4.1.1.3, "Recorder](#).

4.1.3 Watch-Grid

The *Watch-Grid* pane in the bottom of the application window contains the list of watch variables. The selection of watch variables and their graphical properties are defined separately for each project block. As a result, the *Watch-Grid* pane changes its contents each time a different project block is selected.

During the definition of the variable, the variable name, units, and number format are specified. Moreover, the *Watch bar* can be used to change the graphical look of the variable, including font type and size, foreground and background color, and alignment. Refer to [Section 4.6.2, "Watch Bar](#) for details.

Read-only variables can only be monitored. Variables with changes allowed (modifying enabled in the variable definition) can be altered from the *Watch-Grid* pane. Details about variables can be found below in [Section 4.2, "Variables](#).

4.2 Variables

FreeMASTER communicates with the board application via a well-defined communication protocol. This protocol supports sending commands from the PC application to the target board application and reading or writing its variables. All commands and variables used in the FreeMASTER project must be specified within the project.

The *Variables list* dialog box, shown in [Figure 4-18](#), can be opened by selecting the *Variables* item from the *Project* menu. It can also be opened from other project-development points, where it could be used to manage variables (e.g. *Scope Setup*).

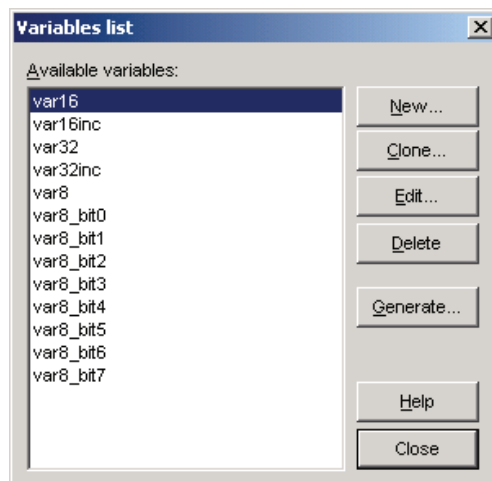


Figure 4-18. Variables list Dialog Box

To define a new variable, use the button *New*. This will open the *Variable* dialog box, where you can set the variable properties described in [Section , "Variable Settings](#). When you need to create a copy of an existing variable, select the original variable and press the *Clone* button.

The *Edit* button opens the same *Variable* dialog box for changing the selected variable properties. After pressing the *Delete* button, the user is asked for confirmation of deletion and the selected variable will be deleted.

Generate button opens the interface for mass creation of variable objects based on the symbols loaded from an embedded application executable file. It is described in [Section 4.2.1, "Generating Variables](#); loading of symbol files is described in [Section 5.2, "Symbol Files](#).

Variable Settings

The *Variable* definition dialog has two tabs: *Definition*, shown in [Figure 4-19](#), and *Modifying*, shown in [Figure 4-20](#).

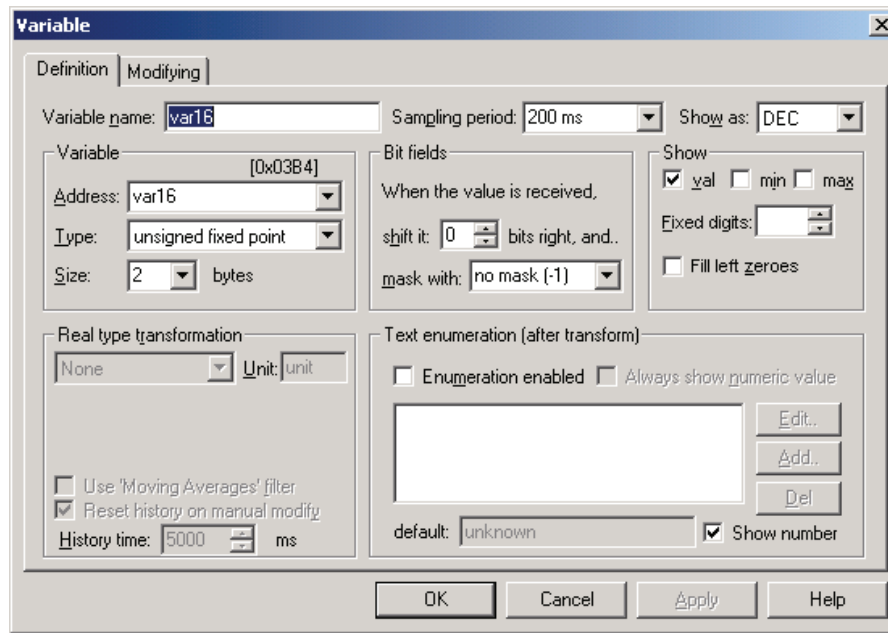


Figure 4-19. Variable Dialog Box - Definition Tab

On the *Definition* tab, you specify a general variable properties.

- *Variable name* = Specify the variable name as the variable identifier in the project
- *Sampling period* = The time period of reading the variable value from the board when the variable is displayed in the variable watch
- *Shows as* = A format in which the variable value is printed in the watch window. Select the proper format from the drop-down list (DEC, HEX, BIN, ASCII or REAL).
- *Variable panel* = Information about the variable as it is defined in the embedded application
 - *Address* = The physical address of the variable in the target application memory. Although you can type the direct hexadecimal value, it is recommended that you select a symbol name of the application variable from the drop-down list. A symbol table can be loaded directly from the embedded application executable (if it is in standard ELF/Dwarf1 format) or from the text-based MAP file generated by the linker. See [Section 5.2, "Symbol Files"](#) for more information about loading the symbol tables from selected files.
 - *Type* = Select the variable type as it is defined in the target application (*unsigned fixed point*, *signed fixed point*, *floating point IEEE*, *fractional <-1,1>*, *unsigned fractional <0,2>* or *string*)
 - *Size* = Specify the size of the variable as it is defined in target application
- *Bit fields* = The parameters for extracting the single bit or bit groups from a given variable
 - *Shift* = Specify the number of bits the received value is right-shifted before it is masked
 - *Mask* = Select or specify the mask value which is AND-ed with the shifted value
 - Using the *Shift* and *Mask* fields, you can extract any bit field from the received variable. For example: to extract the most significant bit from the 16-bit integer value, you would specify 15-bit shifting and one-bit mask (0x1).
- *Show* = According to the value display format selected in the *Show as* field, this set of parameters controls how the variable value is actually printed
 - *val*, *min*, *max* = Check the boxes if you want the variable watch to display the immediate variable value and/or the detected peak values. The peak values can be reset by right-clicking the variable entry in the watch window and selecting the menu command *Reset MIN/MAX*.
 - *Fixed digits* (for *Show as* set to DEC, HEX or BIN) = Prints numeric values left-padded by zeroes or spaces to a given number of digits
 - *Fixed digits* (for *Show as* set to REAL) = Prints floating-point numeric values with a constant number of digits after the decimal point
 - *Zero terminated* (for *Show as* set to string) = The string values are printed only to the first occurrence of zero character. For string values, you can also select whether to display unprintable characters as HEX numbers (or question marks) and a few other string-specific settings.
- *Real type transformation* = When the *Show as* format is set to REAL, you can define further post-processing numeric transformation, which is applied to the variable value.

- Transformation type
 - linear: $ax + b$* : Specify the a and b constants of the linear transformation $y = ax + b$. The 'a' and 'b' parameters can be specified as a numeric values or by the name of the project variables whose immediate value (last valid value) is then used as the parameter.
 - linear two points*: If it is more convenient for you to specify the linear transformation by two points, rather than by the parameters a and b , fill in the two coordinate points $(x1, y1)$ and $(x2, y2)$. As the parameter values, you can again specify the numeric values or variable names.
 - hyp: $d/(ax+b) + c$* : Specify the parameters a , b , c and d of a hyperbolic transformation function.
- *Unit* = The name of unit displayed in the variable watch
- *Use "Moving Averages" filter* = When monitoring a noisy action, you might want to display the average value instead of the immediate one
- *History time* = The time interval from which the average value is computed
- *Text enumeration* = This allows you to describe the meaning of certain variable values and assign the text label to each of them, which is then displayed in the variable watch together with or instead of the numeric value. Use *Edit*, *Add* and *Del* buttons to manage the look-up table with value to text label assignment.
 - *Default* = Specify the default text label, which is displayed when no matching text is found in the look-up table.

The *Modifying* page, shown in Figure 4-20, contains settings and restrictions for variable value modifications.

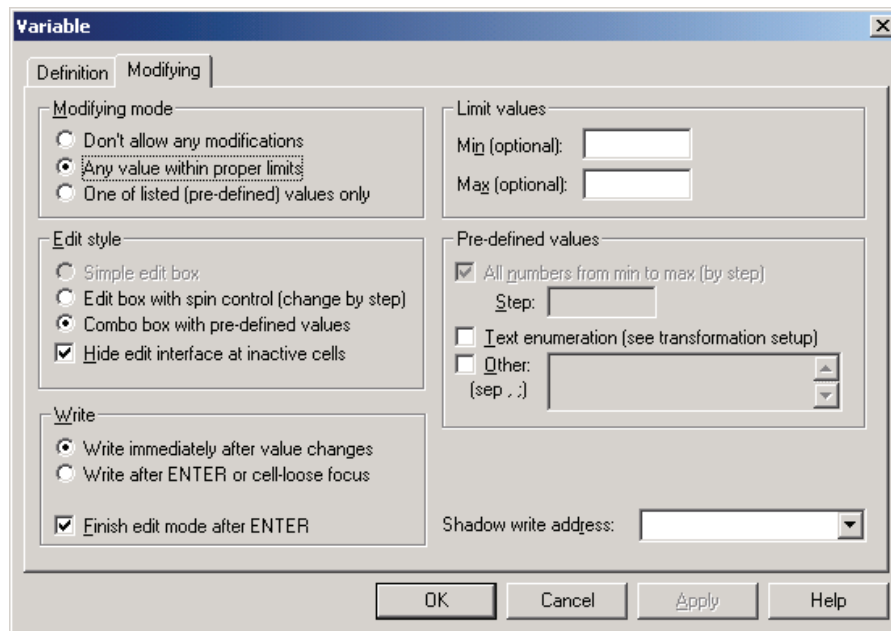


Figure 4-20. Variable Dialog Box - Modifying Tab

- *Modifying mode*
 - *Don't allow any modifications* = The variable is read-only; all other settings on this page are disabled
 - *Any value within proper limits* = You can specify the *Min* and/or *Max* value. The value the user enters in the watch window is then validated with the specified limits.
 - *One of listed values only* = Once you have specified a list of values, only those values will be accepted in the watch window to be written. The acceptable values can be specified in the *Pre-defined values* group.
 - All numbers from min to max* = All the numbers from "min" to "max" by "step" will be treated as predefined values
 - Text enumeration* = Treat all values from the text enumeration look-up table as predefined
 - Other* = Specify any other predefined values (comma or semicolon separated)
- *Edit style* = Select the look of the edit interface for a given variable, which is displayed in the appropriate cell in the watch window grid
 - *Edit box with spin control* = The variable value edit interface will be displayed in the form of an edit box with two spin arrows for incrementing and decrementing the value
 - *Combo box with pre-defined values* = The variable value edit interface will be displayed in the form of a drop-down list box. The predefined values will be available in the list.
 - *Hide edit interface at inactive cells* = The variable edit interface will be hidden when the appropriate cell in the watch grid loses a keyboard focus

- *Write style* = Specify exactly when the new variable value is actually sent to the board application.
 - *Write immediately after each value changes* = The modified variable value will be sent to the embedded application each time the user presses the spin arrow button or selects a new value in the drop-down list box.
 - *Write after ENTER or kill focus only* = The modified variable value will not be sent to the embedded application until the user does not press the Enter key

4.2.1 Generating Variables

The *Generate* button in the variable list dialog (Figure 4-18) opens the *Generate variable* dialog box shown in Figure 4-21.

In this dialog, the user can automatically generate the variable objects for the symbols loaded from an embedded application executable (ELF) or a linker MAP file (see Section 5.2, “Symbol Files for more information about symbol tables).

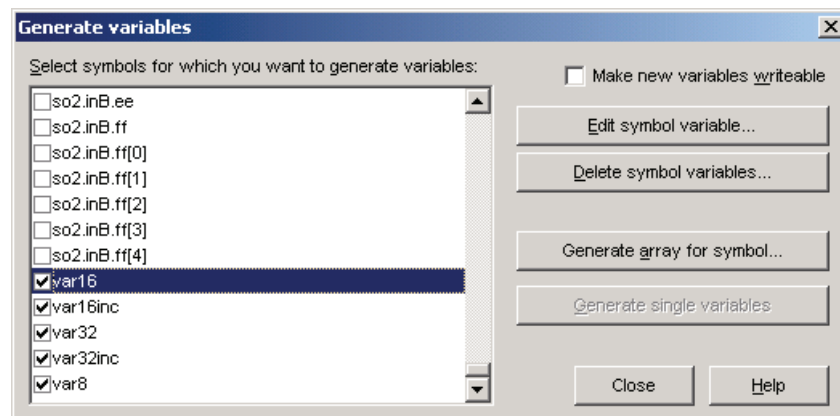


Figure 4-21. Generating Variables

The list in the dialog shows all the symbols available in the project as they were read from the current symbol file. The symbols for which the variables are already defined are marked with a check mark.

- *Edit symbol variable* = Press this button to edit a variable bound to the selected symbol, if any
- *Delete symbol variable* = Press to delete a variable bound to the selected symbol(s)
- *Generate single variables* = Generates a new variable with the same name as the symbol and with proper address, type and size settings. After creation, you can push the *Edit symbol variable* button to see and change other settings in the *Variable* dialog box
- *Generate array for symbol* = Enables you to generate a set of variables encapsulating the selected symbol and its successive locations (offsets)

4.3 Commands

The list of *Application commands* defined in the project can be opened by selecting the *Project / Commands* menu and is shown in Figure 4-22. You can use the buttons *New*, *Clone*, *Edit* and *Delete* to manage the list. It is very similar to variables management:

- *New* button = Creates a new application command
- *Edit* button = Edits properties of the selected application command
- *Clone* button = Creates a new command as a copy of the selected command
- *Delete* button = Deletes the selected command
- *Send* button = Opens the interface which enables the command to be sent to the embedded application

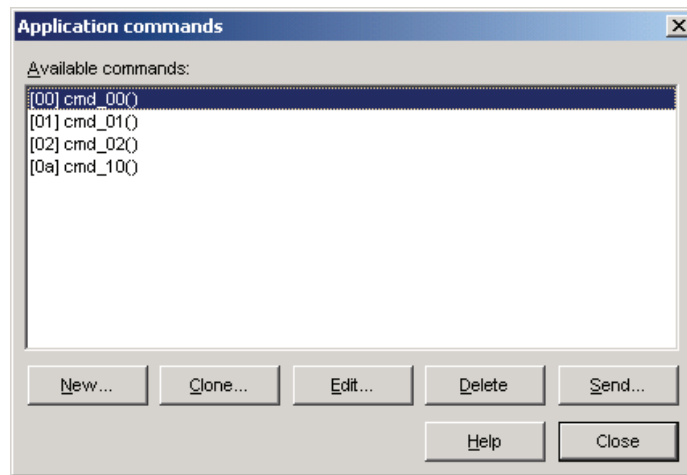


Figure 4-22. Project Application Commands

In the *Send application command* dialog box which follows after pressing the *Send* button, specify the command parameters (if any) and you can send the command to the embedded application. The dialog is shown in [Figure 4-23](#). For each argument, you can define the help message, which is displayed in this dialog when typing the argument value as shown in [Figure 4-24](#).

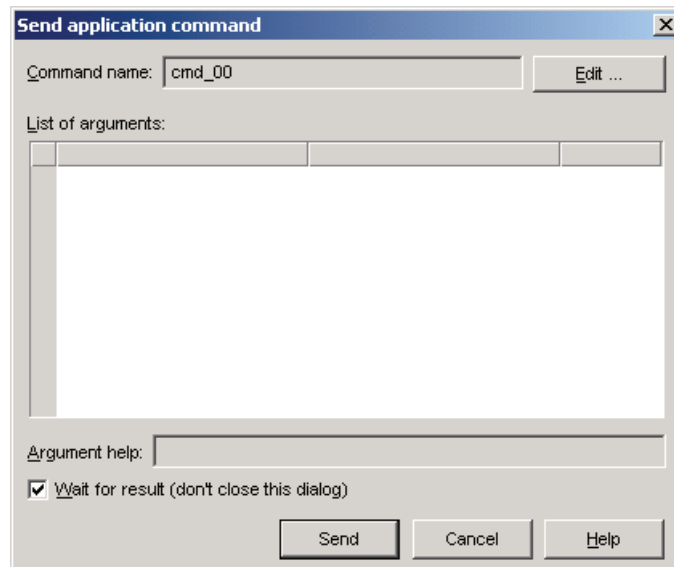


Figure 4-23. Sending Application Command

If you wish to wait for data to be returned from the board (a command result) without closing the dialog, check the *Wait for result* box. Before sending the command, you can review or edit the command definition.

When defining or editing the command, the *Application command* dialog box is opened. The first of three pages of the dialog is shown in [Figure 4-24](#).

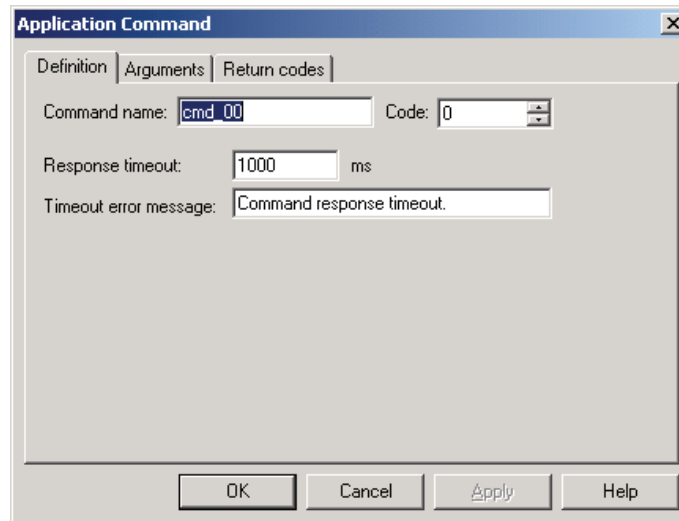


Figure 4-24. Application Command - Definition Tab

On the *Definition* tab, enter the *Command name* used in the project and specify the one-byte command *Code* which identifies the command in the target board application. The command codes and their purposes, as well as the command return codes and their purposes, come from the board application developer.

The *Response time-out* is the maximum time interval in milliseconds that FreeMASTER waits for response from the board application. If the embedded application does not acknowledge the command and respond to it before this time-out occurs, the text entered into the *Timeout error message* field appears in alert window.

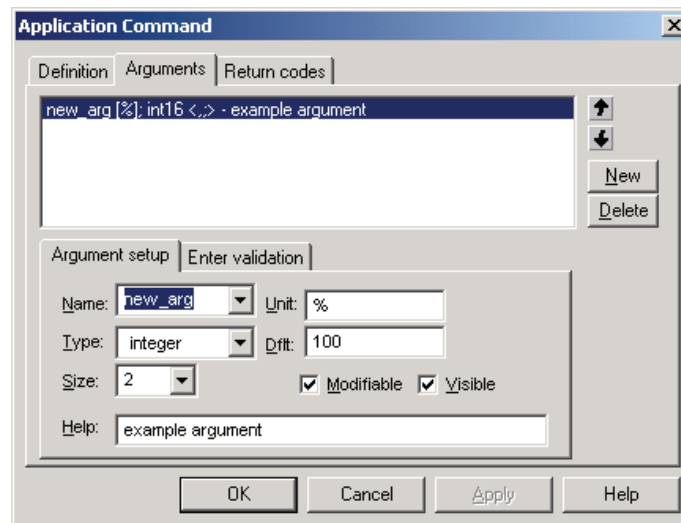


Figure 4-25. Application Command - Arguments Tab, page 1

The *Arguments* tab, shown in [Figure 4-25](#), is used for definition of command arguments. Commands which do not have arguments will have an empty argument list. Commands can have arguments to pass a value to the target board application together with the command code.

Use the *New* button to create a new argument, the *Delete* button to delete an argument selected in the list, and the up and down arrows to change the arguments' order.

On the *Argument setup* sub-page, you define the selected argument parameters:

- *Name* = Specify the argument name as it should appear in the list and in the *Send application command* dialog when the user is prompted for argument values. You can also select the existing argument name from the drop-down list box.
- *Type* = Specify the argument's numeric value (*integer* or *floating point*)
- *Size* = Specify the argument's value size in bytes
- *Unit* = Specify any text which will be displayed as argument units. This text is not sent to the target application.

- *Dflt* = Enter the default value of the argument. This value will be set in the argument list of the *Send application command* dialog. If empty, the user must type the value any time he sends the command.
- *Modifiable* = Unless this box is checked, the user is not allowed to change the default argument value in the argument list of *Send application command* dialog.
- *Visible* = If this box is not checked, the argument will not be displayed in the argument list and its default value will always be sent to the target application.
- *Help* = Write any text information which will be shown in the *Send application command* dialog when the user will be prompted for an argument value.

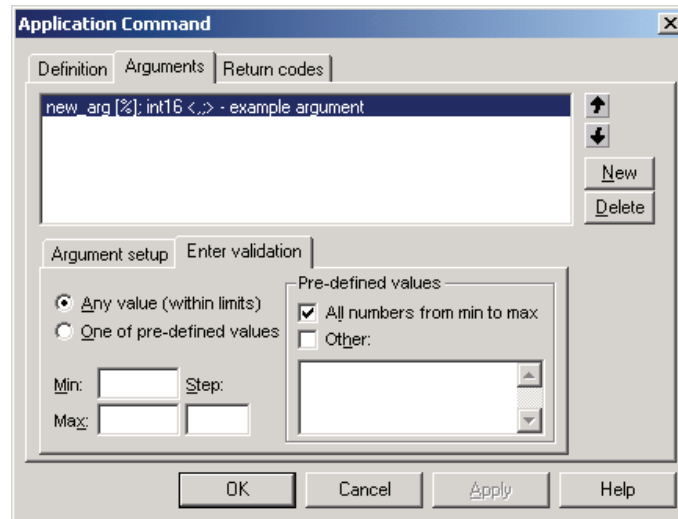


Figure 4-26. Application Command - Arguments Tab, page 2

On the *Enter validation* sub-page, shown in [Figure 4-26](#), you define the validation criteria for the argument value:

- Specify which values are allowed for the argument
 - *Any value* = Any numeric value is allowed as the argument value. The value must be between *Min* and *Max* limits, if these are set.
 - *One of predefined values* = Only one of the values defined in the *Pre-defined values* fields can be supplied as an argument value
- Pre-defined values
 - All numbers from *min* to *max* = When this box is checked, all numbers between *Min* and *Max* limits, incremented by *Step*, are considered to be valid for the argument value
 - *Other* = Check this box and specify the list of other values (comma separated) which are valid as argument value

The *Return codes* page, shown in [Figure 4-27](#), is used for specifying the command return code messages. To create a return code, enter the return code value in hexadecimal (0x00) or decimal form in the *code* field at the lower left of the page; enter the return code message in the next field; and click the *New* button. The return code item will appear in the list. Repeat to create all desired return codes. A *Message icon* may be assigned to each return code message from the panel at the lower right of the page. It will then appear in the message dialog, together with the text of message.

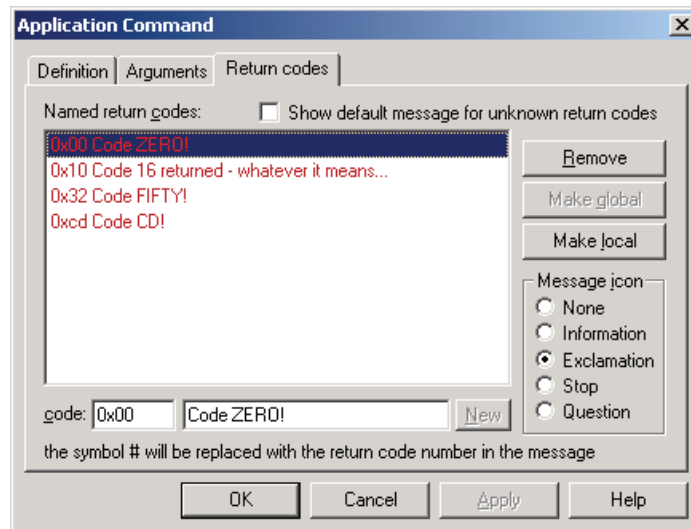


Figure 4-27. Application Command - Return Codes Tab

Return codes can be local or global. Local return codes apply to a single command, while global return codes are valid for all commands of the project. To switch between local and global validity, use the *Make local* and *Make global* buttons.

Check the *Show default messages for unknown return codes* box at the top of the page to pop up a standard message box with return code when an unlisted code returns from the board application.

4.4 Importing Project Files

When preparing your project, you may want to reuse the variables, commands, scope and recorder definitions or watch definitions you created in previous projects. Selecting the *File / Import* menu command opens a dialog in which you can select objects defined in different projects and import them to the current project.

The first dialog of the *Import* procedure is shown in Figure 4-28. After specifying the name of the original project file, you select and check the project tree items you wish to have in your current project. You can also select the target block item under which you want the imported items to be created.

Together with imported tree items, all referenced objects, such as variables or application commands, are also automatically imported. Using the switch radio-buttons below the lists, you can specify how the referenced objects are created:

- *Overwrite existing* = When there is an object, such as a variable, imported with a tree item, for example, an oscilloscope, the current project is searched for an object of the same type (variable) and with the same name. If found, it is overwritten with the one imported.
- *Bind to existing* = If an object with the same name is found, it is not overwritten, but the imported tree item binds to it
- *Always create new* = All referenced objects are created, even if they already exist in the current document; in such a case, the name will be duplicated
- *Merge imported root item* = When importing the root item, it is possible to merge its variable watch definition with the watch of the root item in the current project. When this option is not checked, the root item will be imported and inserted as a standard block item.

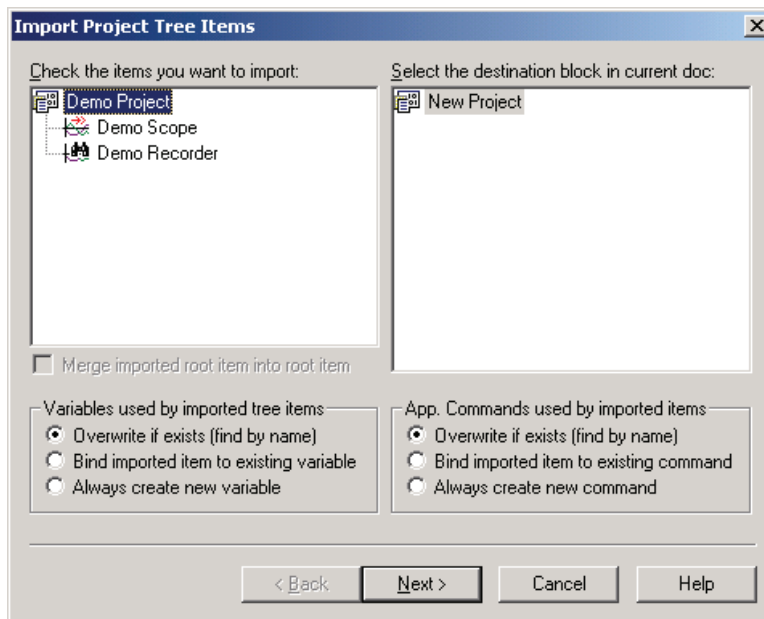


Figure 4-28. Import Project Tree Items

Pressing the *Next* button opens the second part of the *Import* procedure, where you can select additional objects to be imported. The second dialog is shown in Figure 4-29. The three check-box lists contain the objects found in the source project file:

- *Variables* = Put a check mark by each variable you want to import. You don't need to import variables which are referenced from the tree items selected in previous dialog from Figure 4-28; such variables are always unconditionally imported.
- *App. commands* = Put a check mark by each application command definitions you want to import. As with variable objects, you don't need to check commands which are referenced from the block tree items selected in previous dialog.
 - *Global return messages* = If this box is checked, the application commands return codes and messages are imported from the source document.
 - *Overwrite existing* = The existing return codes are overwritten with those being imported.
- *Stimulators* = Put a check mark by each variable stimulator you want to import.

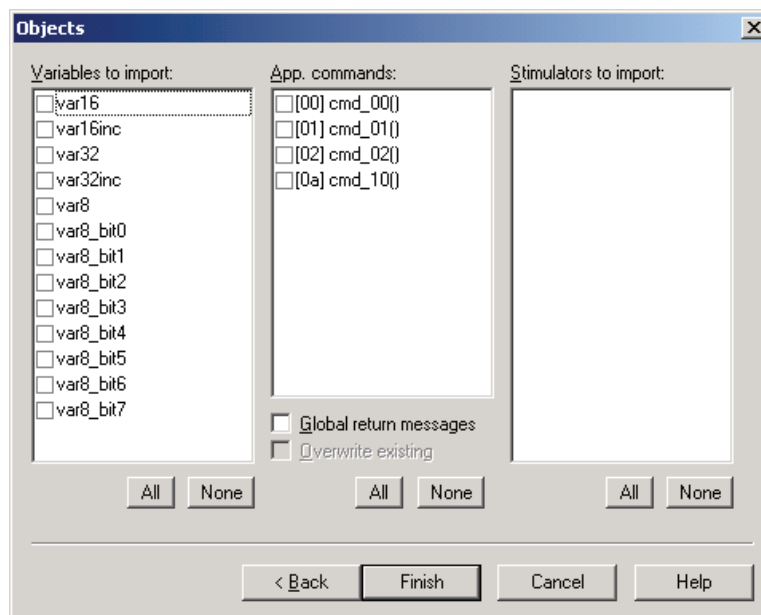


Figure 4-29. Import Project Objects

4.5 Menu description

4.5.1 File Menu

Selecting *New Project* creates a new empty project, while *Open Project* opens an existing project file, which has the extension *.pmp*.

Import Wizard enables you to import selected objects (*Project Tree* items, variables, commands, stimulators) from an existing project (*.pmp* file) into the current project.

Save Project saves the open project into the current file, while *Save As* allows users to specify a new filename for the current project.

Stop Communication pauses the communication with the board and unlocks the communication port. When the communication is released from the paused state, the symbol file is automatically checked for changes. If a difference is found, the user can choose to reload the new version of the file.

Print prints the content of current window, when possible. Currently, printing is supported only for HTML description and control pages. *Print Setup* opens the standard *Print Setup* dialog.

The list of the most recently loaded projects: Selecting one of these items loads the specified project, just like using the *Open Project* command.

Demo Mode is a switch to enter or leave the application demonstration mode. While in *Demo Mode*, you cannot modify any important project settings.

Exit exits the application.

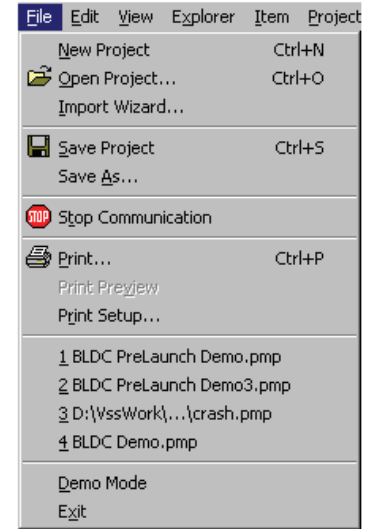


Figure 4-30. File Menu

4.5.2 Edit Menu

Edit menu contains standard clipboard manipulation commands (*Cut*, *Copy* and *Paste*).

Copy Special is enabled when the *Oscilloscope* or *Recorder* graph is active. The command enables saving the graph image to the clipboard or to a file in a different format or size. The set-up dialog is shown in [Figure 4-31](#).

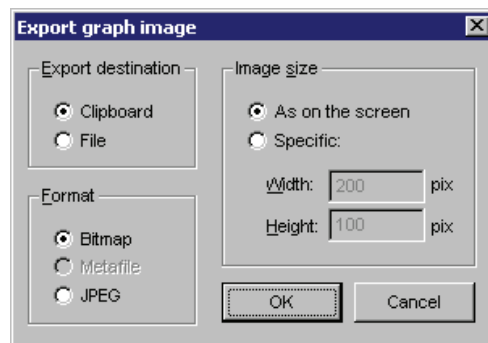


Figure 4-31. Export graph image Dialog

4.5.3 View Menu

In the *View* menu, you can select whether to show or hide the *Toolbar*, the *Watch Bar* or the *Status Line*.

Another feature of the *View* menu is the ability to adjust the size of individual panes without using a mouse. Use *Adjust Left Splitter* for changing the height of the *Fast Access* pane in the *Tree* pane. *Adjust Right Splitter* can be used for changing the height of the *Detail View* pane (HTML pages and charts). Finally, *Adjust Vertical Splitter* changes the width of the *Project Tree* pane.

Show Fast Access Pane switches the display of the *Fast Access* pane, a small helper window located at the bottom-left part of the application window. It contains two pages: the *Commands Page*, which lists currently-defined application commands and the *Stimulators Page*, which lists all stimulator objects. When the menu item *Show all project commands* is checked, FreeMASTER is forced to show all application commands defined in the project. The content of the *commands* page is then independent of which block is currently selected from the *Project Tree*. Otherwise, only commands selected for displaying in the current tree context are displayed; see Figure 4-5 and Section 4.1.1.1, “Project Block and Sub-block for more information.

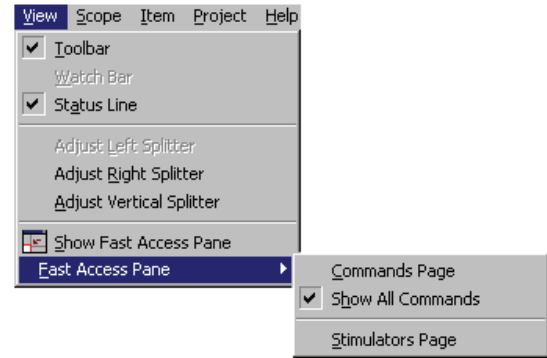


Figure 4-32. View Menu

4.5.4 Explorer Menu

The *Explorer* sub-menu is available when an HTML page (*Control* page, *Block Description* page or *Chart Variables Info*) is displayed in the *Detail View*. When a *Chart View* page is displayed in the *Detail View*, then the *Explorer* menu item is replaced by the *Scope* or *Recorder* sub-menu.

Items *Back*, *Forward* and *Refresh* represent commands for the Internet Explorer window embedded in the FreeMASTER. They are used to move through previously-visited pages and to refresh the page contents.

Fonts sets the font size for the current Internet Explorer window.

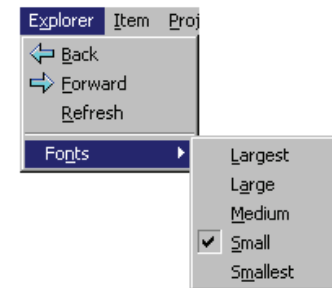


Figure 4-33. Explorer Menu

4.5.5 Scope Menu

The *Stop Scrolling* and *Stop Data* commands cause the FreeMASTER to stop moving (rolling) the oscilloscope chart and to enter a mode in which the chart can be zoomed and the data series can be examined with the data cursor.

The difference between these two commands is that *Stop Scrolling* stops the picture, but allows incoming data to be appended to the end of the chart, while *Stop Data* also stops the incoming data.

Export Picture opens the *Export graph* image dialog, where you can specify the picture format and export the picture to the clipboard or to a file.

Data cursor enables you to select the style of data cursor for examining the chart values. The *Next Subset* sub-item selects the next chart line to be examined.

The *Zoom* submenu consists of zooming modes and commands. The *Horizontal Zoom Only* mode allows “x-axis only” zooming; *Horz and Vert Zoom* allows free rectangle zooming.

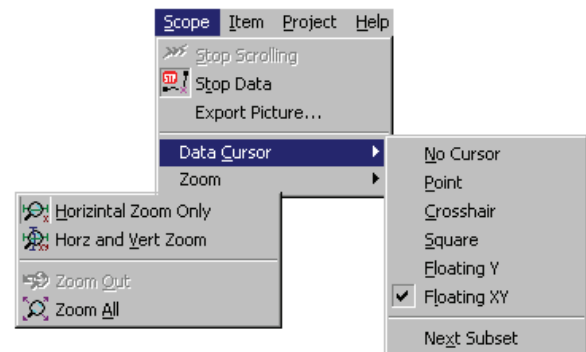


Figure 4-34. Scope Menu

4.5.6 Item Menu

The content of this sub-menu depends on the selected object within the FreeMASTER application window. The same menu is displayed when you click the right mouse button on this object. The menu usually consists of the *Delete* item for deleting the selected object and the *Properties* item for opening an object properties dialog.

4.5.7 Project Menu

Variables opens the Variables list dialog box, where you can manage all project variables at once.

Commands opens the Application commands dialog box, where you can manage all project commands.

The *Reload Map File* command updates the physical addresses of board application variables from the *.map* file.

Options opens the multipage dialog box for all project option settings.

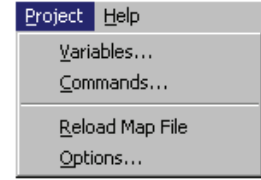


Figure 4-35. Project Menu

4.6 Toolbars

4.6.1 Toolbar

The *toolbar* allows quick access to some of the menu commands.

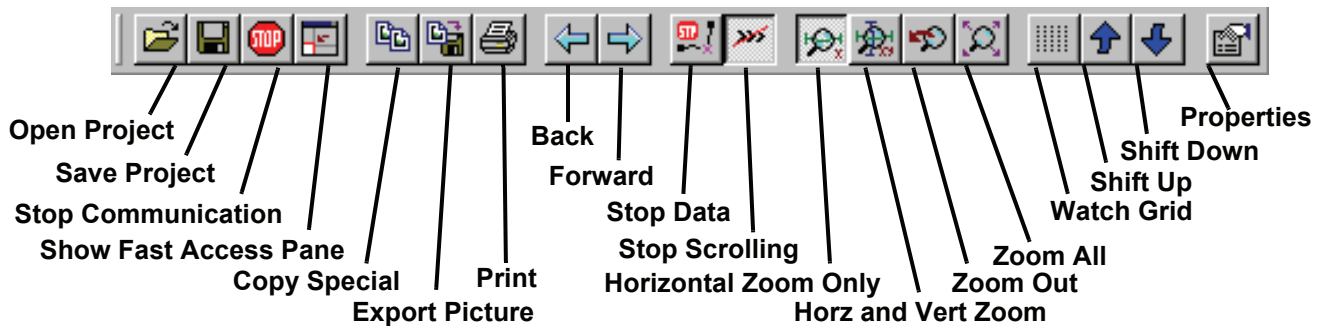


Figure 4-36. Toolbar

4.6.2 Watch Bar

The *Watch Bar* is available when the *Watch-Grid* pane has the focus and *Watch Pane* is selected from the *View* menu. It contains buttons with which you can change various graphical attributes of variables.

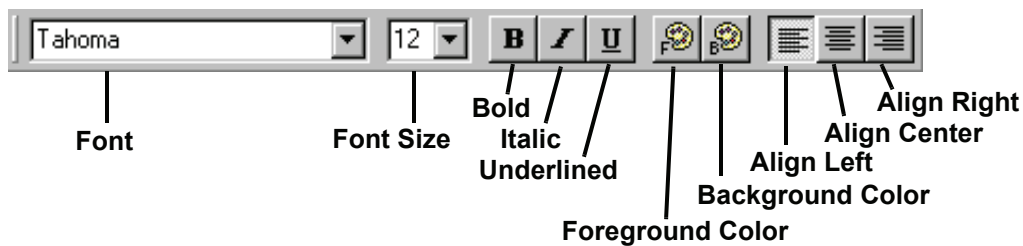


Figure 4-37. Watch Bar

Chapter 5 PROJECT OPTIONS

To set application and project options, use the *Options* dialog, which can be activated by selecting *Options* from the *Project* menu. The dialog consists of several pages - each dedicated to a different group of settings.

5.1 Communication

To set up the parameters related to communication between FreeMASTER application and target board, select the first tab, as shown in Figure 5-1.

- Communication
 - *Direct RS232 connection* = The standard serial interface (3-wire RS232 cable) will be used to connect to the target application
 - Port* = Select the serial port on which the board is accessed
 - Speed* = Select or specify the communication baud rate. This speed must correspond with the embedded application settings.
 - *Plug-in Module Interface* = The communication with an embedded device will be handled by a separate (custom) plug-in module. The module must conform to the Microsoft COM specification and must be registered in the system registry database. See protocol and communication library documentation for more information. The drop-down list contains all plug-in modules known (registered) in the local system.
 - Connect string* = The plug-in module configuration is saved in string form, internally called a “connect string”. You can directly specify this string, or you can press the *Build* button to open the module-specific configuration dialog.

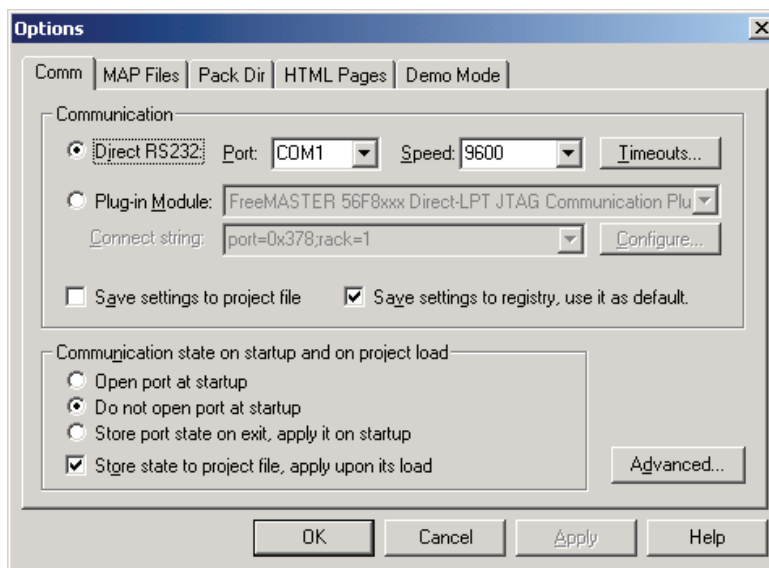


Figure 5-1. Communication Options

- *Save settings to project file* = If checked, the communication parameters will be stored within the project file during the next *Save Project* operation. When the project is next loaded, the communication settings will be restored and used instead of the defaults.
- *Save settings to registry* = If checked, the settings are stored in the local computer registry database. These settings will be used as default when the application is started next time.

NOTE

There are two communication plug-in modules distributed within the FreeMASTER installation pack which handle a communication with the FreeMASTER Remote Server application. The difference between the two is a protocol used to connect to the server. One uses Microsoft DCOM remote procedure call interface while the other uses standard HTTP text-based protocol.

When using the DCOM-based communication to remote server, the security issues must be considered when connecting over the network. Both sides of the communication must have the DCOM properly set up on the system level by running the DCOMCNFG utility. Also, the remote server must be properly installed on the remote side. If there are problems connecting to the remote computer, contact your local network administrator.

- *Communication state* = By selecting one of three options, you can set whether the communication port will be opened or not when the application is started.
 - Open port at start-up
 - Do not open port at start-up
 - Store port status on exit, apply it on start-up
- *Store status to project file, apply it on its load* = If checked, the state of the communication port will be saved to a project file during the next *Save Project* operation. The state will be then restored the next time the project is loaded.

5.2 Symbol Files

When defining project variables, it is often useful to specify the physical address of the target memory location as the symbol name instead of direct hexadecimal value. The symbol table can be loaded directly from the embedded application executable if it is the standard ELF or Dwarf1 debugging format. For other cases, the text MAP file generated by the linker may be loaded and parsed for symbol information.

In the project, you can specify multiple files which contain the symbol table. Later, you can switch between different symbol tables from different files by selecting the menu command *Project / Select Symbol File*. For example, with the DSP embedded application tested on an EVM board, you can have two symbol files specified in the project—one for code running from RAM memory and the second for code running from Flash.

Figure 5-2 shows the *MAP File* tab in the project *Options* dialog.

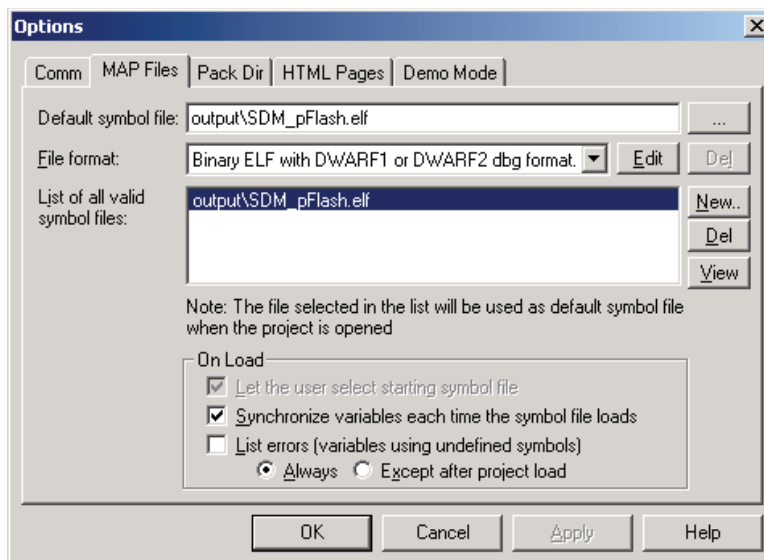


Figure 5-2. Symbol Files Options

- *Default symbol file* = Specify the name of the symbol file, which will be loaded by default. You can press the browse button (...) to locate the file in the standard open file dialog.
- *File format* = Select the format of the file
 - *Standard binary ELF* = Choose this option for an ELF file
 - *Hiware MAP File 509* = Choose this option for HiWare Smart Linker v5.0.9
 - *Define new Regular Expression-based parser* = Choose this to define a new text file parser based on regular expression pattern matching; see [Section 5.2.1, “Regular Expression-based MAP File Parser](#)
- *List of valid symbol files* = The list of symbol files defined in the project. Use *New* and *Del* buttons to manage the list.
- *View* = View the symbol table parsed from the selected file

NOTE

The paths to symbol files may be specified in several forms. It can be the absolute path on the local disk, the path relative to the directory where the project file is stored or the path relative to the current “pack” directory. The latter is the most suitable for project deployment to other computers; see [Section 5.3, “Packing Resource Files into Project File](#) for more information.

- *On Load* panel options control the actions performed with symbol files after the project is loaded:
 - *Let the user select the symbol file* = When checked, the user is prompted to select the initial symbol file when the project is loaded.
 - *Synchronize variables each time the map file loads* = When checked, the variables are automatically updated by new symbol addresses after the project is loaded
 - *List errors* = When this box is checked, all the variables using the symbols missing in the loaded symbol table are listed in a special dialog box. The user is able to edit the corrupted variables or select another symbol file.

5.2.1 Regular Expression-based MAP File Parser

When your compiler or linker does not support ELF output format and the MAP file cannot be parsed by the built-in HiWare 5.0.9 parser, you must describe the internal MAP file structure by using the regular expression pattern.

It would be out of the scope of this document to describe the theory of regular expression pattern matching, but a simple example might help to understand the strength of this technology. In the example, we will define the parser of *xMap* files generated by CodeWarrior Compiler and Linker for the Freescale DSP56800 processor family.

The example of the *xMap* line, which describes the global symbol `length` might look like the following:

```
00002320 00000001 .bss    Flength(bsp.lib pcmasterdrv.o )
00002321 00000001 .bss    Fpos(bsp.lib pcmasterdrv.o )
```

First, we must describe the line format by the regular expression pattern. The following pattern

```
[0-9a-fA-F]+\s+[0-9a-fA-F]+\s+\S+\s+F\w+
```

could be read as follows:

- first, there are one or more hexadecimal digits: `[0-9a-fA-F]+`
- followed by one or more spaces (or another white characters): `\s+`
- followed again by one or more hexadecimal digits: `[0-9a-fA-F]+`
- followed again by one or more white characters: `\s+`
- followed by a set of any non-white characters: `\S+`
- followed by one or more white characters again: `\s+`
- followed by the character `F`
- followed by one or more alphanumeric (word) characters: `\w+`

To identify the sub-patterns which describe the symbol parameters, we put them in the round parentheses:

```
([0-9a-fA-F]+\s+)([0-9a-fA-F]+\s+\S+\s+F)(\w+)
```

Now we must identify the sub-pattern indexes for individual symbol values:

- The first sub-pattern corresponds with the symbol address (index 1)
- The next sub-pattern corresponds with the symbol size (index 2)
- The next sub-pattern corresponds with the symbol name (index 3)

Supply all the parameters prepared above in the regular expression dialog as shown in Figure 5-3.

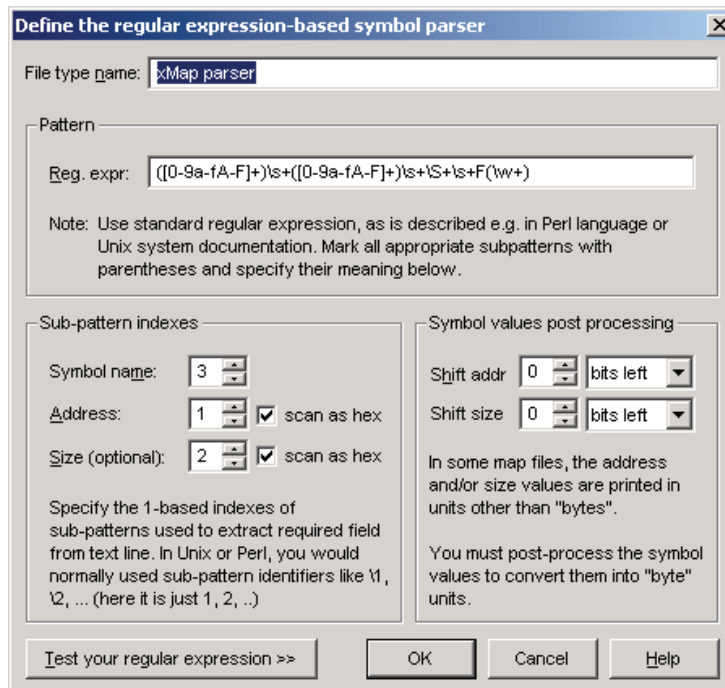


Figure 5-3. Regular Expression-based xMap File Parser

By pressing *Test your regular expression* button, the dialog vertically expands and the test panel is displayed as shown in [Figure 5-4](#). Cut and paste a single line from the *xMap* file to *Input line* field and press the button *Execute reg. expression*:

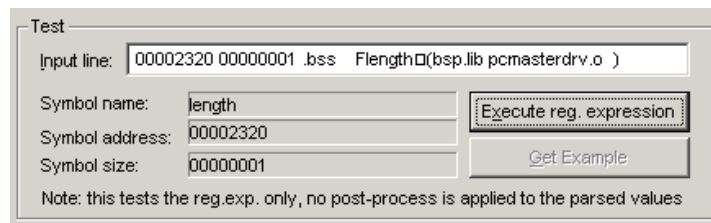


Figure 5-4. Testing Your Regular Expression

The Symbol name, Symbol address and Symbol size output fields should be displayed as shown in [Figure 5-4](#). If you have errors, please be sure that you have Internet Explorer 5.5 or higher installed on your machine for the regular expression functionality.

To finish our example, you must set a one bit *shifting* of the "size" value in the *Symbol post-processing* parameters. This is because the symbol size is printed in word units (16-bit) in the *xMap* file, but we need this value in bytes.

NOTE

All created regular-expression parsers will automatically be saved to the project file and to the local computer registry database for future use.

5.3 Packing Resource Files into Project File

The project often uses data from many different files. For example, each HTML page displayed for a project tree item or each image used on such a page is stored in a separate file. This may cause problems when you want to deploy or distribute a project to different computers. Using the *Pack* options shown in [Figure 5-3](#), you can choose to pack the resource files into the project file when it is saved.

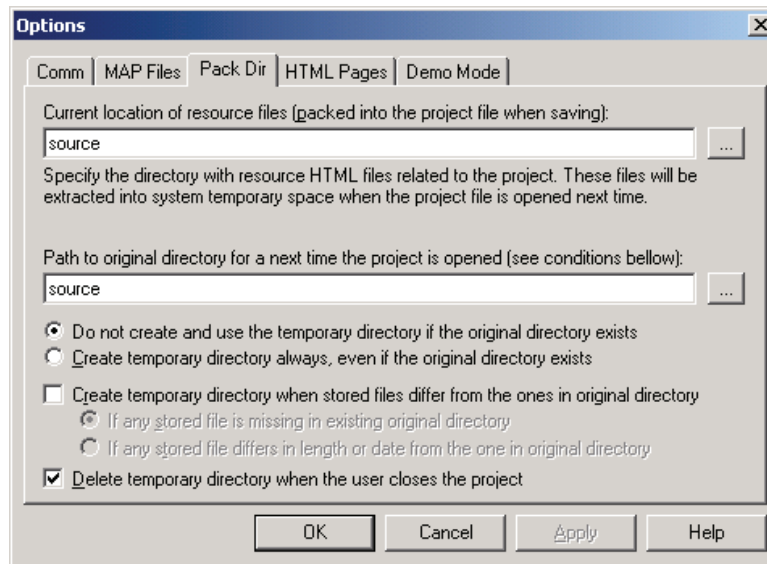


Figure 5-5. Pack Directory Options

To pack resource files into the project file, put all these files into one directory or a directory structure and specify the path to that directory in both fields of the *Pack Dir* page. By pressing the browse button (...), you can find and select the directory in the standard *open directory* dialog. The path to the directory can be specified by a path relative to the directory where the project file is saved.

When the project with the packed files is loaded next time by the application, the *original* path, entered in the second entry of the *Pack Dir* page, is checked. If it is missing, or if one of the files in it differs from the files originally packed in the project, a temporary directory is created in the system temporary space and the packed files are extracted into that directory. The temporary directory is then set as a default for the rest of resource files. It is thus very important to specify the FreeMASTER paths to HTML files or to the symbol file relative to the directory specified in *Pack Dir* dialog.

The *Pack Dir* page displays the path to the temporary directory if it is currently in use, but still remembers the path to the original resource directory.

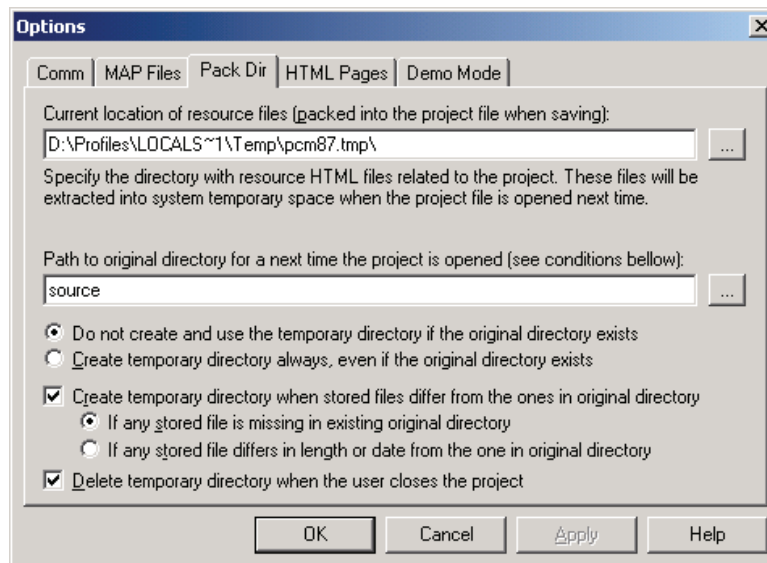


Figure 5-6. Pack Directory Options, Example 2

Using the other Pack Directory options shown in [Figure 5-6](#), you can set the conditions under which the temporary directory is created.

5.3.1 Resource Files Manager

Before deploying a complex project which uses many external resources or symbol files, it is worth verifying that all file paths are correct and in the proper relative format. The path relative format must in turn be properly evaluated when the files are extracted to a temporary directory on different hosts.

The Resource Files Manager can be activated in the menu *Project / Resource Files*. The typical look of the window is shown in [Figure 5-7](#)

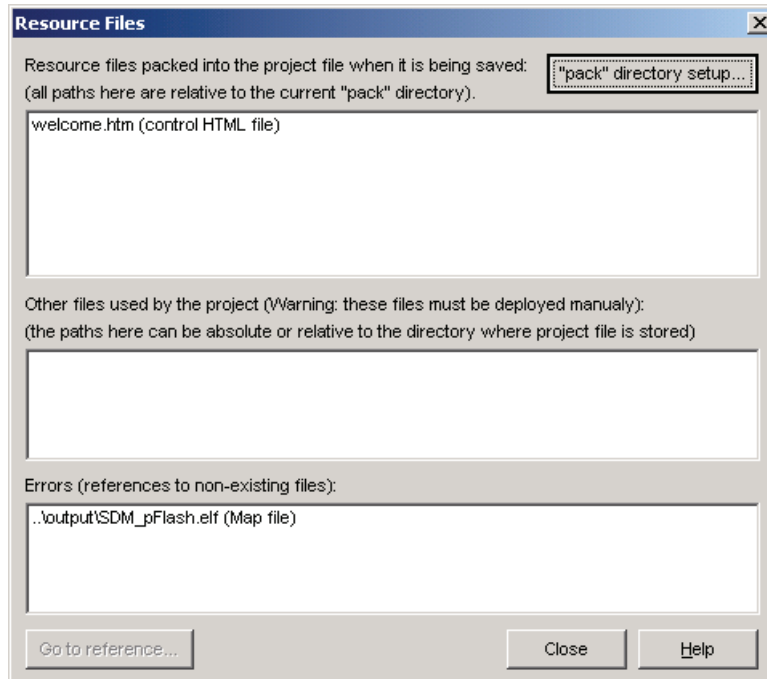


Figure 5-7. Resource Files Manager

The dialog displays all external files directly referenced by the project. Note that there can also be other files (e.g., images) referenced indirectly from the HTML pages. You should verify whether the files lie in the pack directory and whether the HTML pages point to them using a relative path.

- The first list in the dialog contains the files located in the current pack directory. These files will be properly stored in the project file when it is saved. On other hosts, the files will be extracted in a temporary space if needed.
- The second list in dialog contains the files which are successfully used by the project, but are located outside the pack directory. Their file names are specified either by absolute path or by a path relative to the directory where project is stored. However, the files will not automatically be stored in the project file and you will have to deploy them manually.
- The third list contains references to nonexistent files.
- *“Pack” directory setup* = Pressing this button opens the project *Options* dialog, where you can redefine the pack directory location and other settings.
- *Go to reference* = Opens the dialog in which the selected file is referenced. This can be either a *Properties* dialog for a project tree item or a project *Options* dialog for shared HTML or symbol files.

5.4 HTML Pages

The project uses HTML pages to display the description and controls related to the selected item in the project tree. The path or URL of the pages are specified in the property dialog for each tree item. On the *HTML Pages* options tab, shown in [Figure 5-8](#), you can specify the paths to common HTML files, displayed in the application main window.

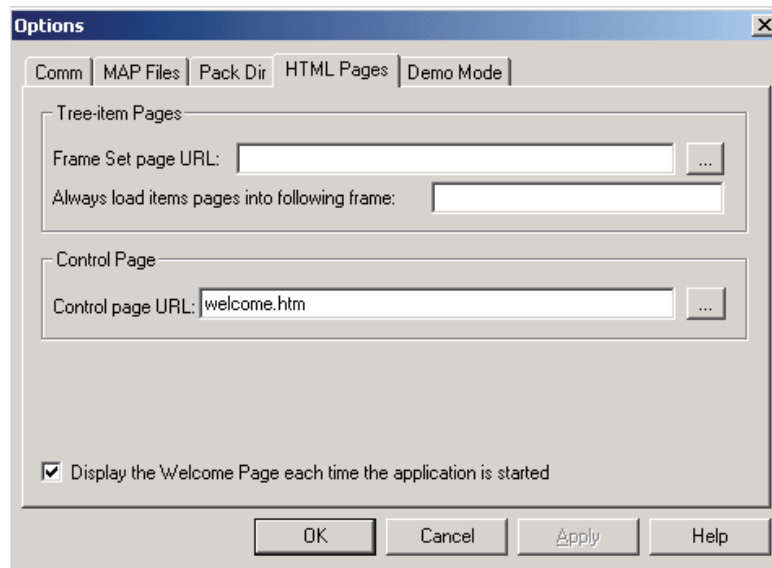


Figure 5-8. HTML Pages Options

Specify the *Frame Set page URL* if you want to divide the pages displayed for project tree items into the frames. The page specified in this field must contain the `<FRAMSET>` declaration, with one frame dedicated as a target for the description pages. The name of the target frame must be specified exactly.

The HTML code can contain frames, controls and scripts (i.e., Visual Basic Script) which can be used to access the attached target board through defined variables and commands. A special HTML page dedicated to the control task can be defined and can be displayed in the separated tab in the detail view of the application main window.

5.5 Demo Mode

An important part of the FreeMASTER's capabilities is the demonstration and description of the target board application. It is essential that the demonstration project, once prepared, is not accidentally modified. To prevent modification, the project's author can lock the project against changes by switching it into the *Demo Mode*. See [Figure 5-9](#) for details of the *Demo Mode* tab.

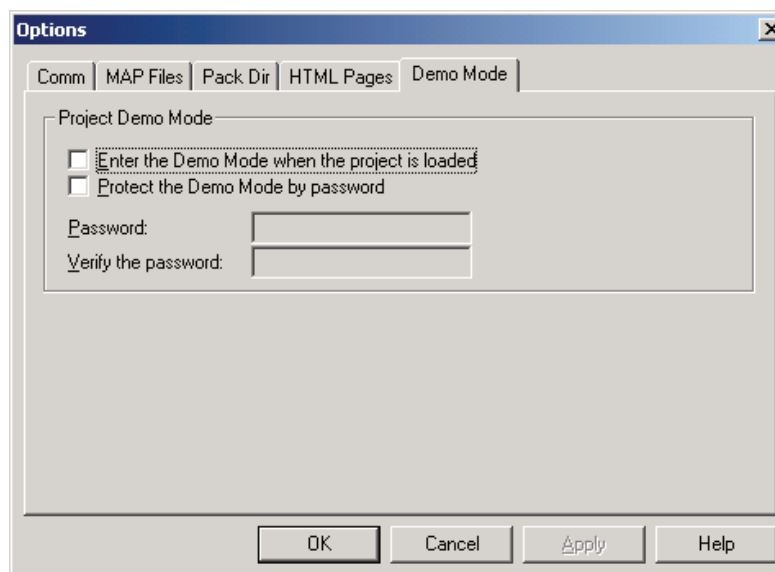


Figure 5-9. Demo Mode Options

When the *Enter the Demo Mode...* box is checked, the demo mode is activated automatically after the project is loaded. The Demo Mode can be started manually by selecting *Demo Mode* from the *File* menu. Exit from Demo Mode can be protected by a password.

PROJECT OPTIONS

In the Demo Mode, the user cannot change the *Project Tree* item properties, cannot add or remove the tree items, and cannot change any project options, except those on the communication page.

When the user asks to leave the Demo Mode, the warning message shown in [Figure 5-10](#) appears, and the user is prompted for the password if the Demo Mode is password-protected.

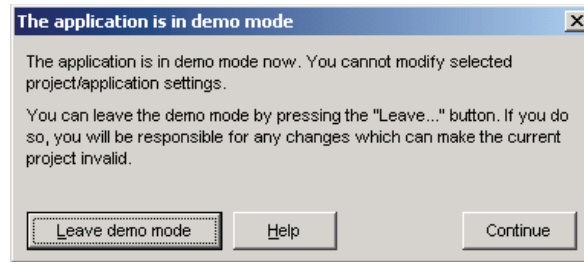


Figure 5-10. Exit Demo Mode Confirmation Dialog

Chapter 6 HTML and SCRIPTING

All HTML pages used in FreeMASTER are rendered using the standard Microsoft Internet Explorer component.

The advantage of using the HTML and Internet Explorer component is that it fully supports scripting languages and enables scripts to embed and access third-party ActiveX controls. The FreeMASTER application itself implements a (non-visual) ActiveX component to let script-based code to access and control the target board application.

The following text is recommended primarily for HTML page authors experienced with scripting and interfacing of the ActiveX controls.

6.1 Special HTML Hyperlinks

When creating HTML pages which are to be displayed in the FreeMASTER environment, the author can use special hyperlinks to let users navigate in the project tree or to invoke selected application commands (see 4.3, “Commands”).

Application command invocation hyperlinks include:

- *HREF=pcmaster:cmd:cmdname(arguments)* sends an application command *cmdname*. Command argument values can be specified in round brackets. An argument with a defined default value may be omitted.
- *HREF=pcmaster:cmdw:cmdname(arguments)* sends an application command *cmdname* and waits until the target application processes it
- *HREF=pcmaster:cmddlg:cmdname(arguments)* displays “Send Application Command” dialog for the application command *cmdname*. Any specified argument values are filled into appropriate fields in the dialog.
- *HREF=pcmaster:selitem:itemname:tabname* selects a project tree item named *itemname* and displays detail view and watch view contents exactly as if the user had selected the item manually. It then selects the specified tab in detail view.

Valid *tabname* identifiers are:

- *ctl,ctltab,control,controltab* = Control Page tab
- *blk,blktab,blkinfo,blkinfotab* = Algorithm Block Description tab
- *info,infotab,iteminfo,iteminfotab* = Current Item Help tab
- *scope,osc,oscilloscope,osctab,scopetab* = Oscilloscope tab
- *recorder,rec,rectab,recordertab* = Recorder tab

6.2 FreeMASTER ActiveX Object

The FreeMASTER object is registered in the system registry during each start of the FreeMASTER application. Its class ID (CLSID) is:

```
{48A185F1-FFDB-11D3-80E3-00C04F176153}
```

The registry name is "MCB.PCM.1"; version independent name is "MCB.PCM".

FreeMASTER functions can be called from any HTML code via the FreeMASTER ActiveX control. Insert the FreeMASTER ActiveX control into your HTML code by the Class ID number (see the example below) and set the dimensions (height and width) to zero to make the object invisible.

```
<object name="freemaster" width="0" height="0"
classid="clsid:48A185F1-FFDB-11D3-80E3-00C04F176153">
```

The FreeMASTER ActiveX control provides the following functions:

- *GetAppVersion* retrieves the application version
- *OpenProject* opens a specified project file
- *StartStopComm* opens or closes the communication port
- *IsCommPortOpen*, *IsBoardDetected* can be used to determine connection status
- *GetHtmlDocument* retrieves the HTML DOM object from the FreeMASTER window
- *SendCommand* sends FreeMASTER-defined command
- *SendCommandDlg* opens command dialog and send a FreeMASTER-defined command
- *ReadVariable* reads value from a FreeMASTER-defined variable
- *WriteVariable* writes value to a FreeMASTER-defined variable
- *ReadMemory*, *ReadMemoryHex* reads a block of memory from a specified location
- *ReadIntArray*, *ReadUIntArray*, *ReadFloatArray*, *ReadDoubleArray* read numeric arrays from a specified location
- *WriteIntArray*, *WriteUIntArray*, *WriteFloatArray*, *WriteDoubleArray* write numeric arrays to a specified location
- *ReadIntVariable*, *ReadUIntVariable*, *ReadFloatVariable*, *ReadDoubleVariable* read single value from a specified location
- *WriteIntVariable*, *WriteUIntVariable*, *WriteFloatVariable*, *WriteDoubleVariable* write single value to a specified location
- *GetCurrentRecorderData* retrieves data currently displayed in the recorder chart
- *GetCurrentRecorderSeries* retrieves one data series from the currently-displayed recorder chart
- *StartCurrentRecorder* starts the currently-displayed recorder
- *StopCurrentRecorder* stops the currently-displayed recorder
- *GetCurrentRecorderState* retrieves the current recorder status code and text
- *LocalFileOpen*, *LocalFileClose* enables to open and close local file for temporary text-based storage
- *LocalFileWriteString* writes text-based data to file
- *LocalFileReadString* reads text-based data from file
- *GetSymbolInfo*, *GetStructMember*, *GetAddressInfo* retrieve information about symbols in the target application
- *DefineSymbol*, *DeleteAllScriptSymbols* manipulate the symbolic information on top of the ones provided by the target application
- *SubscribeVariable*, *UnSubscribeVariable* enable the script to be notified when a variable value changes
- *SelectItem* select the project item in the FreeMASTER project tree view
- *DefineVariable*, *DefineScope*, *DefineRecorder* are complex functions which can be used to dynamically create or modify FreeMASTER variable objects or Scope/Recorder graphs
- *IsBoardWithActiveContent*, *EnumHrefLinks*, *EnumProjectFiles* can be used to enumerate TSA Active Content items. Refer to FreeMASTER Serial Driver documentation for more information about TSA and Active Content.
- *PipeOpen*, *PipeClose*, *PipeWrite* and many other *Pipe* functions to be used for loss-less communication with the target board.
- *EnumVariables*, *EnumSymbols* to enumerate variables and symbols loaded in the current project.
- *GetDetectedBoardInfo* to retrieve information about connected board.
- *GetCommPortInfo* to retrieve information about current communication port settings.

Callback events implemented:

- *OnRecorderDone* called when the currently-selected recorder finishes downloading new recorder data
- *OnCommPortStateChanged* called when communication port status changes
- *OnBoardDetected* called when valid connection to a target board is established
- *OnVariableChanged* called when any of subscribed variables changes

6.3 FreeMASTER ActiveX Object Methods

6.3.1 GetAppVersion

GetAppVersion (bsRetMsg)

6.3.1.1 Description

This function get the FreeMASTER application version.

6.3.1.2 Arguments:

| | |
|------------------------------------|---|
| bsRetMsg [out, optional] | Application version as a string (e.g. 1.4.1.6). Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |
|------------------------------------|---|

6.3.1.3 Return Value:

| | |
|---------------|--|
| Numeric Value | Numeric value representing the FreeMASTER version. For example, for version 1.4.1-build 6, the return value is 0x01040106. |
|---------------|--|

6.3.2 OpenProject

OpenProject (bsProject)

6.3.2.1 Description

This function opens the specified project file.

6.3.2.2 Arguments:

| | |
|--------------------------|--------------------------------------|
| bsProject [in] | String containing project file path. |
|--------------------------|--------------------------------------|

6.3.2.3 Return Value:

| | |
|---------------|-------------------------------------|
| Boolean Value | <i>True</i> if open was successful. |
|---------------|-------------------------------------|

6.3.3 StartStopComm

StartStopComm (bStart)

6.3.3.1 Description

This function opens or closes the FreeMASTER communication interface selected by the currently loaded project.

6.3.3.2 Arguments:

| | |
|-----------------------|---|
| bStart [in] | Set as <i>“True”</i> to start communication. Set as <i>“False”</i> to close communication |
|-----------------------|---|

6.3.3.3 Return Value:

| | |
|---------------|---|
| Boolean Value | <i>“True”</i> is returned if the command has been successfully performed. <i>“False”</i> is returned if the command failed to open or close the communication interface. |
|---------------|---|

6.3.4 IsCommPortOpen

IsCommPortOpen ()

6.3.4.1 Description

This function returns state of the communication port.

6.3.4.2 Arguments:

None.

6.3.4.3 Return Value:

| | |
|---------------|---|
| Boolean Value | <i>True</i> if the communication port is currently open. <i>False</i> if port is closed. |
|---------------|---|

6.3.5 IsBoardDetected

IsBoardDetected ()

6.3.5.1 Description

This function returns state of a connection to the target board.

6.3.5.2 Arguments:

None.

6.3.5.3 Return Value:

| | |
|---------------|---|
| Boolean Value | <i>True</i> if the communication port is currently open and the board is online. <i>False</i> if port is closed or if board could not be enumerated. |
|---------------|---|

6.3.6 StartStopComm

StartStopComm (bStart)

6.3.6.1 Description

This function opens or closes the FreeMASTER communication interface selected by the currently loaded project.

6.3.6.2 Arguments:

| | |
|-----------------------|---|
| bStart [in] | Set as <i>“True”</i> to start communication. Set as <i>“False”</i> to close communication |
|-----------------------|---|

6.3.6.3 Return Value:

| | |
|---------------|---|
| Boolean Value | <i>“True”</i> is returned if the command has been successfully performed. <i>“False”</i> is returned if the command failed to open or close the communication interface. |
|---------------|---|

6.3.7 GetHtmlDocument

GetHtmlDocument (nWinId, bActivate)

6.3.7.1 Description

This function returns a pointer to DOM object of the HTML renderer.

6.3.7.2 Arguments:

| | |
|--------------------------|--|
| nWinId [in] | Number identifying the HTML pane to access. Use one of the following values: 0 - Control page 1 - Block description page 2 - Detailed description page (for Scope and Recorder items) |
| bActivate [in] | Boolean value which selects whether activate the selected HTML view. |

6.3.7.3 Return Value:

| | |
|------------------------|---|
| Object dispatch handle | The returned handle can be used in the script to access the HTML Document object. |
|------------------------|---|

6.3.8 SendCommand

SendCommand (bsCmd, bsRetMsg)

6.3.8.1 Description

This function sends the FreeMASTER Application Command.

6.3.8.2 Arguments:

| | |
|------------------------------------|---|
| bsCmd [in] | String value with the command name and arguments which should be sent. The command must be defined in the currently-open FreeMASTER project. Use the function call-like syntax when sending a command. For example if a command has three parameters, use "command_name(1, 2, 3)" |
| bsRetMsg [out, optional] | Text returned after the command invocation. When an error occurs, this value contains an error message Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.8.3 Return Value:

| | |
|---------------|--|
| Boolean value | " <i>True</i> " is returned if the command has been sent without errors. " <i>False</i> " is returned if the command could not be found in FreeMASTER project. |
|---------------|--|

6.3.9 SendCommandDlg

SendCommandDlg (bsCmd, bsRetMsg)

6.3.9.1 Description

This function invokes the FreeMASTER *Send Application Command* dialog.

6.3.9.2 Arguments:

| | |
|------------------------------------|--|
| bsCmd [in] | String value with the command name and arguments which should be displayed in the dialog. The command must be defined in the currently-open FreeMASTER project |
| bsRetMsg [out, optional] | Text returned after the command invocation. When an error occurs, this value contains an error message Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.9.3 Return Value:

| | |
|---------------|--|
| Boolean value | “ <i>True</i> ” is returned if the command has been sent without errors. “ <i>False</i> ” is returned if the command could not be found in FreeMASTER project. |
|---------------|--|

6.3.10 ReadVariable

ReadVariable (bsVar, vValue, tValue, bsRetMsg)

6.3.10.1 Description

This function reads value from a FreeMASTER-defined variable.

Note: This method may return a “cached” value of the variable if it is younger than the sampling time defined. For example if the variable sampling time is set to one second, a script calling the ReadVariable function more often will not retrieve the live value from target. To disable this caching mechanism, use an exclamation mark ‘!’ in front of the variable name in the bsVar parameter.

6.3.10.2 Arguments:

| | |
|------------------------------------|---|
| bsVar [in] | String value with the name of the variable to be read. The variable must be defined in the FreeMASTER project which is currently opened. |
| vValue [out, optional] | Returns numeric representation of the variable <i>bsVar</i> . Value of this output parameter is mirrored in LastVariable_vValue data member. Use it when the scripting language does not support [out] arguments. |
| tValue [out, optional] | Returns string value which represents the variable format and units necessary for displaying the variable value. Value of this output parameter is mirrored in LastVariable_tValue data member. Use it when the scripting language does not support [out] arguments. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.10.3 Return Value:

| | |
|---------------|--|
| Boolean value | “True” is returned if the variable has been read without errors. “False” is returned if the specified variable was not found in FreeMASTER project or if a communication error occurred. |
|---------------|--|

6.3.11 WriteVariable

```
WriteVariable (bsVar, vValue, bsRetMsg)
```

6.3.11.1 Description

This function writes a value to a FreeMASTER variable.

6.3.11.2 Arguments:

| | |
|------------------------------------|---|
| bsVar [in] | A string value with the name of the variable to be written. The variable must be defined in the currently-open FreeMASTER project |
| vValue [in] | Value to write to the variable |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.11.3 Return Value:

| | |
|---------------|---|
| Boolean value | “ <i>True</i> ” is returned if the variable has been written without errors. “ <i>False</i> ” is returned if the specified variable was not found in FreeMASTER project, if the value passed was invalid, or if a communication error occurred. |
|---------------|---|

6.3.12 ReadMemory

```
ReadMemory (vAddr, vSize, arrData, bsRetMsg)
```

```
ReadMemory_t (vAddr, vSize, arrData, bsRetMsg)
```

```
ReadMemoryHex (vAddr, vSize, bsData, bsRetMsg)
```

6.3.12.1 Description

These functions read a block of memory from the target application.

- *ReadMemory* returns data in the safearray of variants, which can be used in both scripting languages like VBScript and in compiled languages like Visual Basic.
- *ReadMemory_t* returns data in the safearray of type “unsigned 1 byte integer”, which can be used in compiled languages like Visual Basic. Using typed arrays is faster than using arrays of variants.
- *ReadMemoryHex* returns data in the string value. Each byte is represented in hexadecimal form by two characters.

6.3.12.2 Arguments:

| | |
|------------------------------------|---|
| vAddr [in] | The address of the memory block to be read. This can be either an absolute numeric address, a symbol name valid in the current FreeMASTER project, or a symbol name plus offset value |
| vSize [in] | The size of memory block to be read |
| arrData [out, optional] | The return array of the values. Value of this output parameter is mirrored in LastMemory_data data member. Use it when the scripting language does not support [out] arguments. |
| bsData [out, optional] | The return data in the string format. Value of this output parameter is mirrored in LastMemory_hexstr data member. Use it when the scripting language does not support [out] arguments. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.12.3 Return Value:

| | |
|---------------|--|
| Boolean value | “True” is returned if the memory block has been read without errors. “False” is returned if the specified address was invalid or if a communication error occurred. |
|---------------|--|

6.3.13 WriteMemory

```
WriteMemory (vAddr, vSize, arrData, bsRetMsg)
```

6.3.13.1 Description

This function writes a block of bytes to the target application's memory.

6.3.13.2 Arguments:

| | |
|------------------------------------|---|
| vAddr [in] | The address of the memory area to be written. This can be either an absolute numeric address, a symbol name valid in the current FreeMASTER project, or a symbol name plus offset value |
| vSize [in] | The size of memory block to be written |
| arrData [in] | Safe array of bytes to be written. It must contain at least vSize elements. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.13.3 Return Value:

| | |
|---------------|--|
| Boolean value | " <i>True</i> " is returned if the memory block has been read without errors. " <i>False</i> " is returned if the specified address was invalid or if a communication error occurred. |
|---------------|--|

6.3.14 ReadXxxArray

To access array of 1,2, or 4-byte signed integers:

```
ReadIntArray (vAddr, vSize, vElemSize, arrData, bsRetMsg)
ReadIntArray_t (vAddr, vSize, vElemSize, arrData, bsRetMsg)
```

To access array of 1,2, or 4-byte unsigned integers:

```
ReadUIntArray (vAddr, vSize, vElemSize, arrData, bsRetMsg)
ReadUIntArray_v (vAddr, vSize, vElemSize, arrData, bsRetMsg)
ReadUIntArray_t (vAddr, vSize, vElemSize, arrData, bsRetMsg)
```

To access array of 4-byte floating point numbers:

```
ReadFloatArray (vAddr, vSize, arrData, bsRetMsg)
ReadFloatArray_t (vAddr, vSize, arrData, bsRetMsg)
```

To access array of 8-byte floating point numbers:

```
ReadDoubleArray (vAddr, vSize, arrData, bsRetMsg)
ReadDoubleArray_t (vAddr, vSize, arrData, bsRetMsg)
```

6.3.14.1 Description

These functions read a block of memory from the target application and return it to the caller in the form of integer or floating point numbers. For each call, there are one or two optional functions which can be used in different scripting environments:

- *ReadXxxArray* returns data as a safearray of variants, which can be used in both scripting languages like VBScript and in compiled languages like Visual Basic. As VBScript engine does not handle variants encapsulating 2 and 4-byte unsigned integer types, this method converts such a values to floating point format before returning.
- *ReadXxxArray_v* is the same as *ReadXxxArray* except it does not perform the VBScript translation for 2 and 4-byte unsigned integer types.
- *ReadXxxArray_t* returns data in the safearray of strictly typed values, which can be used in compiled languages like Visual Basic. Using typed arrays is significantly faster than using arrays of variants.

6.3.14.2 Arguments:

| | |
|------------------------------------|---|
| vAddr [in] | The address of the memory block to be read. This can be either an absolute numeric address, a symbol name valid in the current FreeMASTER project, or a symbol name plus offset value |
| vSize [in] | The number of elements to be read from the array |
| vElemSize [in] | The size of an array element in bytes. The total number of bytes read from the target can be calculated as $vSize * vElemSize$. |
| arrData [out, optional] | The return array of the values. Value of this output parameter is mirrored in LastMemory_data data member. Use it when the scripting language does not support [out] arguments. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.14.3 Return Value:

| | |
|---------------|--|
| Boolean value | <i>True</i> is returned if the memory block has been read without errors. <i>False</i> is returned if the specified address was invalid or if a communication error occurred. |
|---------------|--|

6.3.15 WriteXxxArray

To access array of 1,2, or 4-byte signed integers:

```
WriteIntArray (vAddr, vSize, vElemSize, arrData, bsRetMsg)
```

To access array of 1,2, or 4-byte unsigned integers:

```
WriteUIntArray (vAddr, vSize, vElemSize, arrData, bsRetMsg)
```

To access array of 4-byte floating point numbers:

```
WriteFloatArray (vAddr, vSize, arrData, bsRetMsg)
```

To access array of 8-byte floating point numbers:

```
WriteDoubleArray (vAddr, vSize, arrData, bsRetMsg)
```

6.3.15.1 Description

These functions translate an array of numbers passed by the caller into a block of bytes suitable for the target CPU and write it into the target memory.

6.3.15.2 Arguments:

| | |
|------------------------------------|---|
| vAddr [in] | The address of the memory block to be written. This can be either an absolute numeric address, a symbol name valid in the current FreeMASTER project, or a symbol name plus offset value |
| vSize [in] | The number of elements to be written to the target memory. |
| vElemSize [in] | The size of an array element in bytes. The total number of bytes to be written to the target can be calculated as $vSize * vElemSize$. |
| arrData [in] | The array of the values to be written. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.15.3 Return Value:

| | |
|---------------|---|
| Boolean value | "True" is returned if the memory block has been Write without errors. "False" is returned if the specified address was invalid or if a communication error occurred. |
|---------------|---|

6.3.16 ReadXxxVariable

To access 1,2, or 4-byte signed integers:

```
ReadIntVariable (vAddr, vSize, vValue, bsRetMsg)
ReadIntVariable_v (vAddr, vSize, vValue, bsRetMsg)
```

To access 1,2, or 4-byte unsigned variable:

```
ReadUIntVariable (vAddr, vSize, vValue, bsRetMsg)
ReadUIntArray_v (vAddr, vSize, vValue, bsRetMsg)
```

To access 4-byte floating point variable:

```
ReadFloatVariable (vAddr, vValue, bsRetMsg)
```

To access 8-byte floating point variable:

```
ReadDoubleVariable (vAddr, vValue, bsRetMsg)
```

6.3.16.1 Description

These functions reads a memory cell from the target application and return it to the caller in the form of integer or floating point number. Unlike the ReadVariable function, these functions access the memory directly, without a FreeMASTER variable object defined. For each call, there is an optional function which can be used in different scripting environments:

- *ReadXxxVariable* returns data as a variant suitable for both scripting languages like VBScript and for compiled languages like Visual Basic. As VBScript engine does not handle variants encapsulating 2 and 4-byte unsigned integer types, this method converts such a values to floating point format before returning.
- *ReadXxxVariable_v* is the same as *ReadXxxVariable* except it does not perform the VBScript translation for 2 and 4-byte unsigned integer types.

6.3.16.2 Arguments:

| | |
|------------------------------------|---|
| vAddr [in] | The address of the memory block to be read. This can be either an absolute numeric address, a symbol name valid in the current FreeMASTER project, or a symbol name plus offset value |
| vSize [in] | The size of variable to read. |
| vValue [out, optional] | The return value. Value of this output parameter is mirrored in LastVariable_vValue data member. Use it when the scripting language does not support [out] arguments. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.16.3 Return Value:

| | |
|---------------|---|
| Boolean value | “True” is returned if the memory has been read without errors. “False” is returned if the specified address was invalid or if a communication error occurred. |
|---------------|---|

6.3.17 WriteXxxVariable

To write 1,2, or 4-byte signed integer:

```
WriteIntVariable (vAddr, vSize, vValue, bsRetMsg)
```

To write 1,2, or 4-byte unsigned integer:

```
WriteUIntVariable (vAddr, vSize, vValue, bsRetMsg)
```

To write 4-byte floating point number:

```
WriteFloatVariable (vAddr, vSize, vValue, bsRetMsg)
```

To write 8-byte floating point number:

```
WriteDoubleVariable (vAddr, vSize, vValue, bsRetMsg)
```

6.3.17.1 Description

These functions write a single variable to target memory. Unlike the WriteVariable function, these functions access the memory directly, without a FreeMASTER variable object defined.

6.3.17.2 Arguments:

| | |
|------------------------------------|---|
| vAddr [in] | The address of the memory block to be written. This can be either an absolute numeric address, a symbol name valid in the current FreeMASTER project, or a symbol name plus offset value |
| vSize [in] | The size of the target variable. |
| vValue [in] | The value to be written. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.17.3 Return Value:

| | |
|---------------|---|
| Boolean value | “True” is returned if the memory block has been Write without errors. “False” is returned if the specified address was invalid or if a communication error occurred. |
|---------------|---|

6.3.18 GetCurrentRecorderData

```
GetCurrentRecorderData (arrData, arrSerieNames, timeBaseSec, bsRetMsg)
```

```
GetCurrentRecorderData_t (arrData, arrSerieNames, timeBaseSec, bsRetMsg)
```

6.3.18.1 Description

These functions retrieve data currently displayed in the recorder chart. They differ in how the data is returned to the caller.

- *GetCurrentRecorderData* returns data in the safearray of variants, which can be used in both scripting languages like VBScript and in compiled languages like Visual Basic
- *GetCurrentRecorderData_t* returns data in the safearray of type “double floating point”, which can be used in compiled languages like Visual Basic. Using typed arrays is faster than using arrays of variants.

6.3.18.2 Arguments:

| | |
|--|---|
| arrData [out, optional] | Return, two-dimensional array of values; the first dimension is series index, the second dimension is value index. Value of this output parameter is mirrored in LastRecorder_data data member. Use it when the scripting language does not support [out] arguments. |
| arrSerieNames [out, optional] | Return array with names of series on appropriate indexes. Value of this output parameter is mirrored in LastRecorder_serieNames data member. Use it when the scripting language does not support [out] arguments. |
| timeBaseSec [out, optional] | Returned time between individual recorded samples in seconds; this value can be 0 if the target does not supply such a value. Value of this output parameter is mirrored in LastRecorder_timeBaseSec data member. Use it when the scripting language does not support [out] arguments. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.18.3 Return Value:

| | |
|---------------|--|
| Boolean value | “True” is returned if the data has been retrieved successfully from the recorder item. “False” is returned if there is no valid data in the recorder or there is no currently-active recorder. |
|---------------|--|

6.3.19 GetCurrentRecorderSeries

```
GetCurrentRecorderSeries (bsSerieName, arrData, timeBaseSec, bsRetMsg)
```

```
GetCurrentRecorderSeries_t (bsSerieName, arrData, timeBaseSec, bsRetMsg)
```

6.3.19.1 Description

These functions retrieve one data series from the currently-displayed recorder chart. They differ in how the data is returned to the caller.

- *GetCurrentRecorderSeries* returns data in the safearray of variants, which can be used in both scripting languages like VBScript and in compiled languages like Visual Basic.
- *GetCurrentRecorderSeries_t* - returns data in the safearray of type “double floating point”, which can be used in compiled languages like Visual Basic. Using typed arrays is faster than using arrays of variants.

6.3.19.2 Arguments:

| | |
|---------------------------------------|---|
| bsSerieName [in] | String with the name of the series whose data is to be retrieved |
| arrData [out, optional] | Return array of the values. Value of this output parameter is mirrored in LastRecorder_data data member. Use it when the scripting language does not support [out] arguments. |
| timeBaseSec [out, optional] | Returned time between individual recorded samples in seconds; this value can be 0 if the target does not supply such a value. Value of this output parameter is mirrored in LastRecorder_timeBaseSec data member. Use it when the scripting language does not support [out] arguments. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.19.3 Return Value:

| | |
|---------------|---|
| Boolean value | “ <i>True</i> ” is returned if the data has been retrieved successfully from the recorder item. “ <i>False</i> ” is returned if there is no valid data in the recorder or if there is no recorder currently active. |
|---------------|---|

6.3.20 StartCurrentRecorder

StartCurrentRecorder (bsRetMsg)

6.3.20.1 Description

This function starts the recorder which is currently-displayed in the FreeMASTER window.

6.3.20.2 Argument:

| | |
|------------------------------------|---|
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |
|------------------------------------|---|

6.3.20.3 Return value:

| | |
|---------------|---|
| Boolean value | “ <i>True</i> ” is returned if the recorder has been started successfully. “ <i>False</i> ” is returned if there is no recorder currently active. |
|---------------|---|

6.3.21 StopCurrentRecorder

StopCurrentRecorder (bsRetMsg)

6.3.21.1 Description

This function manually stops the Recorder which is currently-displayed in the FreeMASTER window.

6.3.21.2 Argument:

| | |
|---|--|
| bsRetMsg [out, optional] | <p>When an error occurs, this value contains an error message; otherwise, it is empty.</p> <p>Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments.</p> |
|---|--|

6.3.21.3 Return Value:

| | |
|---------------|--|
| Boolean value | <p><i>“True”</i> is returned if the recorder has been successfully stopped. <i>“False”</i> is returned if there is no recorder currently active.</p> |
|---------------|--|

6.3.22 GetCurrentRecorderState

GetCurrentRecorderState (nState, bsRetMsg)

6.3.22.1 Description

This function retrieves current recorder status code and assigned status text.

6.3.22.2 Arguments:

| | |
|---|---|
| nState [out, optional] | <p>Return numeric value identifying current recorder state. Valid states codes are:</p> <ul style="list-style-type: none"> 0=idle; 1=starting; 2=running; 3=downloading results; 4=holding received signal; 5=error; 6=manually stopping; 7=not initialized; 8=holding in error; 9=ready to download; <p>Value of this output parameter is mirrored in LastRecorder_state data member. Use it when the scripting language does not support [out] arguments.</p> |
| bsRetMsg [out, optional] | <p>When an error occurs, this value contains an error message; otherwise, it contains text description of current recorder state.</p> <p>Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments.</p> |

6.3.22.3 Return Value:

| | |
|---------------|--|
| Boolean value | <p>“True” is returned if the recorder state has been read successfully. “False” is returned if there is no recorder currently active.</p> |
|---------------|--|

6.3.23 RunStimulators

RunStimulators (bsStimNames)

6.3.23.1 Description

This function starts one or more FreeMASTER variable stimulators.

6.3.23.2 Arguments:

| | |
|----------------------------|--|
| bsStimNames [in] | A semicolon-delimited list of stimulators to be started. |
|----------------------------|--|

6.3.23.3 Return Value:

| | |
|---------------|---|
| Integer Value | Number of stimulators actually started. This number may be equal or less than the number of semicolon-delimited stimulator names passed in bsStimNames parameter. The return count does not include stimulators not found by name and stimulators which were already running. |
|---------------|---|

6.3.24 StopStimulators

StopStimulators (bsStimNames)

6.3.24.1 Description

This function halts one or more FreeMASTER variable stimulators.

6.3.24.2 Arguments:

| | |
|----------------------------|---|
| bsStimNames [in] | A semicolon-delimited list of stimulators to be halted. |
|----------------------------|---|

6.3.24.3 Return Value:

| | |
|---------------|---|
| Integer Value | Number of stimulators actually stopped. This number may be equal or less than the number of semicolon-delimited stimulator names passed in bsStimNames parameter. The return count does not include stimulators not found by name and stimulators which were not running. |
|---------------|---|

6.3.25 LocalFileOpen

LocalFileOpen (bsFileName, bsOpenMode)

6.3.25.1 Description

This function opens file locally stored in the project area and returns handle to the file for subsequent operations.

6.3.25.2 Arguments:

| | |
|-------------------------------------|--|
| bsFileName [in] | Name of file to be opened, optionally with a relative path. Due to security reasons, this function denies to open files on absolute path or files with unsafe extensions. The permitted extensions are: .txt, .xml, .htm, .html, .c, .cpp, .asm, .h, .hpp. The relative path can start with one of the virtual folder names: FMSTR_PROJECT_PATH - location of current project file FMSTR_PACKDIR_PATH - location of current resource module folder |
| bsOpenMode [in, optional] | Specifies access mode of the open file: “r” – open file for reading (default). “w” – create or open file for writing, original file gets truncated to zero length if it already exists “a” – create or open file for writing in append mode, original file is not truncated if it already exists |

6.3.25.3 Return Value:

| | |
|--------------|--|
| Handle Value | <i>Numeric Handle to file or a false Boolean value in case the file could not be opened.</i> |
|--------------|--|

6.3.26 LocalFileClose

LocalFileClose (nHandle)

6.3.26.1 Description

This function closes the file identified by the nHandle parameter.

6.3.26.2 Arguments:

| | |
|-----------------|------------------------------|
| nHandle [in] | Handle to file to be closed. |
|-----------------|------------------------------|

6.3.26.3 Return Value:

| | |
|---------------|---|
| Boolean Value | <i>“True”</i> is returned if the file has been successfully closed. <i>“False”</i> is returned in case the file handle is not valid. |
|---------------|---|

6.3.27 LocalFileWriteString

LocalFileWriteString (nHandle, bsData, nCharsToWrite, bUnicode)

6.3.27.1 Description

This function writes data to file.

6.3.27.2 Arguments:

| | |
|--|---|
| nHandle [in] | Handle to open file. |
| bsData [in] | Text to be written to the file. |
| nCharsToWrite [in, optional] | Number of characters to be actually written. If this parameter is omitted or is set 0, the whole text passed in vData parameter is written. |
| bUnicode [in, optional] | “ <i>True</i> ” - write data in unicode format (not supported). “ <i>False</i> ” - write data in ASCII format. |

6.3.27.3 Return Value:

| | |
|---------------|---|
| Integer Value | Returns number of characters written to the file. |
|---------------|---|

6.3.28 LocalFileReadString

LocalFileReadString (nHandle, nCharsToRead, bUnicode, bsRetString)

6.3.28.1 Description

This function reads data from file.

6.3.28.2 Arguments:

| | |
|---------------------------------------|---|
| nHandle [in] | Handle to open file. |
| nCharsToRead [in] | Number of characters to read. |
| bUnicode [in, optional] | " <i>True</i> " - read data in unicode format (not supported). " <i>False</i> " - read data in ASCII format. |
| bsRetString [out, optional] | The return data containing the characters read from the file. Value of this output parameter is mirrored in LastLocalFile_string data member. Use it when the scripting language does not support [out] arguments. |

6.3.28.3 Return Value:

| | |
|---------------|---|
| Integer Value | Returns number of characters actually read from the file. |
|---------------|---|

6.3.29 GetSymbolInfo

GetSymbolInfo (bsSymbol, vSymAddr, vSymSize, bsRetMsg, bUseVBScriptTypes)

6.3.29.1 Description

This function returns information about symbol in the target application executable (ELF or MAP file).

6.3.29.2 Arguments:

| | |
|--|---|
| bsSymbol [in] | Name of the symbol. |
| vSymAddr [out, optional] | Address of the symbol. Value of this output parameter is mirrored in LastSymbolInfo_addr data member. Use it when the scripting language does not support [out] arguments. |
| vSymSize [out, optional] | Size of the symbol. Value of this output parameter is mirrored in LastSymbolInfo_size data member. Use it when the scripting language does not support [out] arguments. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |
| bUseVBScriptTypes [in, optional] | Use True to return address and size as floating point numbers compatible with VBScript scripting language. |

6.3.29.3 Return Value:

| | |
|---------------|--|
| Boolean Value | <p>"True" is returned if the symbol exists and information was retrieved. "False" is returned when symbol was not found.</p> |
|---------------|--|

6.3.30 GetStructMember

GetStructMember (bsTypeName, bsMember, vMembOff, vMembSize, bsRetMsg, bUseVBScriptTypes)

6.3.30.1 Description

This function returns information about structure type in the target application executable (ELF).

6.3.30.2 Arguments:

| | |
|--|---|
| bsTypeName [in] | Name of the structure type. |
| bsMember [in] | Name of the structure member. |
| vMembOff [out, optional] | Returns offset of the structure member. Value of this output parameter is mirrored in LastMemberInfo_offset data member. Use it when the scripting language does not support [out] arguments. |
| vMembSize [out, optional] | Returns size of the structure member. Value of this output parameter is mirrored in LastMemberInfo_size data member. Use it when the scripting language does not support [out] arguments. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |
| bUseVBScriptTypes [in, optional] | Use True to return address and size as floating point numbers compatible with VBScript scripting language. |

6.3.30.3 Return Value:

| | |
|---------------|---|
| Boolean Value | <p>"True" is returned if the type exists and information was retrieved. "False" is returned when structure type was not found.</p> |
|---------------|---|

6.3.31 GetAddressInfo

```
GetAddressInfo (vAddr, vSize, bIsExactMatch, bsSymbolName);
```

6.3.31.1 Description

This function returns information about memory location the target application executable (ELF). All global and static target symbols are evaluated to see if they overlap with the memory area specified. The algorithm tries to find the best matching name of the memory area, also including the structure members and array elements.

6.3.31.2 Arguments:

| | |
|---|--|
| vAddr [in] | Memory address. |
| vSize [in] | Size of the memory area to lookup. |
| bIsExactMatch [out, optional] | Returns True if the returned symbol matches exactly the memory address and area size. Value of this output parameter is mirrored in LastAddressInfo_exact data member. Use it when the scripting language does not support [out] arguments. |
| bsSymbolName [out, optional] | Returns the best-matching name of the memory area. Value of this output parameter is mirrored in LastAddressInfo_name data member. Use it when the scripting language does not support [out] arguments. |

6.3.31.3 Return Value:

| | |
|---------------|---|
| Boolean Value | <p>"True" is returned if the memory area was mapped to any global or static symbol.</p> <p>"False" is returned when the memory area was not mapped to any symbol.</p> |
|---------------|---|

6.3.32 DefineSymbol

```
DefineSymbol (bsSymbol, vAddr, bsTypeName, vSize, bsRetMsg);
```

6.3.32.1 Description

This function can be used to extend the symbol information normally obtained by parsing the target application executable file (ELF or MAP files). This may be useful to define symbols for dynamically allocated variables or other non-global objects for which you know the address and type. Once the symbol is defined, a FreeMASTER variable can be defined in the GUI (or again by script method *DefineVariable*).

6.3.32.2 Arguments:

| | |
|------------------------------------|---|
| bsSymbol [in] | Symbol name. It can also contain special characters like ., -> or [] when defining complex symbols mapping structure members. |
| vAddr [in] | Symbol address. |
| bsTypeName [in] | Name of the type behind the symbol. Typically this is a name of existing structure type loaded from the debugging information of the application ELF file. When null or an empty string is passed, no type is assigned to the symbol. When a structure type name is passed, the FreeMASTER defines also symbols for each structure member. The separator between symbol name and member name is a dot '.' by default. It can be changed to "->" if a "pointer" type name is specified, for example "TYPE*". This feature enables to define symbols with valid C language syntax. |
| vSize [in] | Symbol size. You can also use special expressions like "sizeof(TYPE)" or mathematical expressions. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.32.3 Return Value:

| | |
|---------------|---|
| Boolean Value | "True" is returned if the symbol was defined with the type information as specified. "False" is returned when the type name was specified, but such type was not found in the type information loaded from application executable file. |
|---------------|---|

6.3.33 DeleteAllScriptSymbols

```
DefineSymbol();
```

6.3.33.1 Description

This function deletes all symbols defined previously with *DefineSymbol* call.

6.3.33.2 Arguments:

None.

6.3.33.3 Return Value:

| | |
|---------------|--|
| Boolean Value | <i>This method returns "True" value.</i> |
|---------------|--|

6.3.34 SubscribeVariable

```
SubscribeVariable (bsVarName, vInterval, bsRetMsg);
```

6.3.34.1 Description

This function may be used by script to enable event-driven processing of variable value changes. Once a variable is “subscribed” the script receives *OnVariableChanged* notification events whenever the FreeMASTER detects that the variable value has changed. All subscribed variables are periodically sampled by FreeMASTER at a rate defined in related FreeMASTER variable object (or faster when the variable is also displayed in a scope graph).

6.3.34.2 Arguments:

| | |
|------------------------------------|---|
| bsVarName [in] | Variable name, this must be a valid FreeMASTER variable name (not just a symbol name). |
| vInterval [in] | A “dead time” interval which may be used to prevent the script to be overloaded with events if variable changes too fast. An <i>OnVariableChanged</i> event is only called once per this interval value even if the variable changes more frequently. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.34.3 Return Value:

| | |
|---------------|---|
| Numeric Value | Value other than zero is a Subscription ID which can be used to unsubscribe the variable. This ID also identifies the variable in the <i>OnVariableChanged</i> event callback. Zero when variable could not be subscribed. |
|---------------|---|

6.3.35 UnSubscribeVariable

```
UnSubscribeVariable (vNameOrId, bsRetMsg);
```

6.3.35.1 Description

This function cancels the variable subscription and stops the *OnVariableChanged* event calls for the variable identified by the first parameter.

6.3.35.2 Arguments:

| | |
|------------------------------------|---|
| vNameOrId [in] | Variable name used to subscribe the variable or subscription ID which was returned from the <i>SubscribeVariable</i> call. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.35.3 Return Value:

| | |
|---------------|---|
| Boolean Value | " <i>True</i> " is returned if the variable was un-subscribed. " <i>False</i> " if an error occurred. |
|---------------|---|

6.3.36 SelectItem

```
SelectItem (bsName, vTabPage);
```

6.3.36.1 Description

This function selects the project tree item in the FreeMASTER window and activates one of the view tabs assigned to the item.

6.3.36.2 Arguments:

| | |
|-------------------------|---|
| bsName [in] | Name of the item as it appears in the FreeMASTER project tree view. |
| vTabPage [in] | Name of the FreeMASTER view tab which is to be activated after the tree item is selected. Use one of the following values: - <i>control</i> ... Control Page Tab - <i>blkinfo</i> ... Parent Block description page - <i>iteminfo</i> ... Item description page - <i>scope</i> ... Scope graph tab - <i>recorder</i> ... Recorder graph tab |

6.3.36.3 Return Value:

| | |
|---------------|---|
| Boolean Value | " <i>True</i> " is returned when the item was found and selected. " <i>False</i> " otherwise. |
|---------------|---|

6.3.37 DefineVariable

```
DefineVariable (bsName, bsDefString, bsRetMsg);
```

6.3.37.1 Description

This function enables script to define FreeMASTER variable object with all its properties. The script needs to use a JSON notation to describe all properties of the object. The variable object is created or is modified if it already exists. All properties which are not defined in JSON text are assigned a default values.

It is out of scope of this document to describe how to generate the JSON text notation. For JScript users it is easy to map the object properties, lists and arrays to this format.

To get list of valid JSON parameter names, save existing FreeMASTER project which contains any variable as an XML format and see the XML tags describing the variable items. The JSON notation uses the same property names.

6.3.37.2 Arguments:

| | |
|------------------------------------|---|
| bsName [in] | Name of the FreeMASTER variable to be created or modified. |
| bsDefString [in] | The JSON definition of the variable object. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.37.3 Return Value:

| | |
|---------------|---|
| Boolean Value | " <i>True</i> " is returned when the object was created or modified. " <i>False</i> " if an error occurred. |
|---------------|---|

6.3.37.4 JScript Example:

```
function define_variable(name, symbol, size, period)
{
    var def = {
        "address" : symbol,
        "byte_size" : size,
        "period" : period,
    };

    ok = pcmaster.DefineVariable(name, JSON.stringify(def));
}
```

6.3.38 DefineScope

```
DefineScope (bsName, bsDefString, bsRetMsg);
```

6.3.38.1 Description

This function enables script to define FreeMASTER Oscilloscope object with all its properties. The script needs to use a JSON notation to describe all properties of the object. The scope object is created or is modified if it already exists. All properties which are not defined in JSON text are assigned a default values.

It is out of scope of this document to describe how to generate the JSON text notation. For JScript users it is easy to map the object properties, lists and arrays to this format.

To get list of valid JSON parameter names, save existing FreeMASTER project which contains a scope as an XML format and see the XML tags describing the scope item. The JSON notation uses the same property names.

6.3.38.2 Arguments:

| | |
|------------------------------------|---|
| bsName [in] | Name of the FreeMASTER oscilloscope item to be created or modified in project tree. |
| bsDefString [in] | The JSON definition of the oscilloscope object. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.38.3 Return Value:

| | |
|---------------|---|
| Boolean Value | " <i>True</i> " is returned when the object was created or modified. " <i>False</i> " if an error occurred. |
|---------------|---|

6.3.38.4 JScript Example:

```
function define_my_scope()
{
    // array of scope variables, my_variable_x should already be valid FreeMASTER
    // variable objects, first two variables will share the Y graph block
    var vars = [
        { "variable":"my_variable_1", "visible":true, "y_block":0 },
        { "variable":"my_variable_2", "visible":true, "y_block":0 },
        { "variable":"my_variable_3", "visible":true, "y_block":1 },
    ];

    // array of scope Y-blocks, we have two, not joined
    var yblocks = [
        { "laxis_label":"raw signal", "join_class":0,
          "laxis_min_auto":true, "laxis_max_auto":true },
        { "laxis_label":"touch status", "join_class":1,
          "laxis_min_auto":true, "laxis_max_auto":true },
    ];

    // scope definition
    var def = {};
    def["var_info"] = vars;
}
```



```
def["yblock_info"] = yblocks;  
def["scope_period"] = 0.025; // 25ms  
def["href"] = "my_description_page.htm";  
  
pcmaster.DefineScope("my scope", JSON.stringify(def));  
}
```

6.3.39 DefineRecorder

```
DefineRecorder (bsName, bsDefString, bsRetMsg);
```

6.3.39.1 Description

This function matches exactly the *DefineScope* method described above. Use it to create or modify the Recorder object in the FreeMASTER project tree.

6.3.40 IsBoardWithActiveContent

```
IsBoardWithActiveContent ();
```

6.3.40.1 Description

This function can be used to determine if an Active Content is present in the currently connected target MCU board. The Active Content includes HTML pages, project files and hyperlinks. Refer to FreeMASTER Serial Driver documentation for more details about embedding of the Active Content into a target MCU application.

6.3.40.2 Arguments

None.

6.3.40.3 Return Value:

| | |
|---------------|---|
| Boolean Value | "True" is returned when the Active Content is present. "False" otherwise. |
|---------------|---|

6.3.40.4 JScript Example

See JScript example of using this function in the source code of FreeMASTER Welcome page which is displayed as soon as the FreeMASTER starts. The Welcome page detects and displays the hyper links contained by the target MCU's Active Content feature.

6.3.41 EnumHrefLinks

```
EnumHrefLinks (vIndex, bsHrefName);
```

6.3.41.1 Description

This function can be used to enumerate hyperlinks in the target MCU board's Active Content. User script is expected to call this function with increasing vIndex value until an invalid (false) value is returned.

6.3.41.2 Arguments

| | |
|--------------------------------------|---|
| vIndex [in] | Index of hyperlink to be retrieved. User is expected to call this function with vIndex increasing from 0 until the function returns false value. |
| bsHrefName [out, optional] | Hyperlink text name - the string to be displayed to user. Value of this output parameter is mirrored in LastLinkName data member. Use it when the scripting language does not support [out] arguments. |

6.3.41.3 Return Value:

| | |
|--------------|---|
| String value | Hyperlink URL value in string form when hyperlink at given vIndex exists. Otherwise an empty false value is returned. |
|--------------|---|

6.3.41.4 JScript Example

See JScript example of using this function in the source code of FreeMASTER Welcome page which is displayed as soon as the FreeMASTER starts. The Welcome page detects and displays the hyper links contained by the target MCU's Active Content feature.

6.3.42 EnumProjectFiles

```
EnumProjectFiles (vIndex, bsPrjName);
```

6.3.42.1 Description

This function can be used to enumerate project files in the target MCU board's Active Content. User script is expected to call this function with increasing vIndex value until an invalid (false) value is returned.

6.3.42.2 Arguments

| | |
|--------------------------------------|--|
| vIndex [in] | Index of hyperlink to be retrieved. User is expected to call this function with vIndex increasing from 0 until the function returns false value. |
| bsHrefName [out, optional] | Project name - the string to be displayed to user. Value of this output parameter is mirrored in LastLinkName data member. Use it when the scripting language does not support [out] arguments. |

6.3.42.3 Return Value:

| | |
|--------------|--|
| String value | Project file URL in string form when project at given vIndex exists. Otherwise an empty false value is returned. |
|--------------|--|

6.3.42.4 JScript Example

See JScript example of using this function in the source code of FreeMASTER Welcome page which is displayed as soon as the FreeMASTER starts. The Welcome page detects and displays the hyper links contained by the target MCU's Active Content feature.

6.3.43 PipeOpen

```
PipeOpen (vPort, vTxBufferSize, vRxBufferSize, bsRetMsg);
```

6.3.43.1 Description

This function initializes the pipe object used to communicate with the target MCU using a loss-less stream I/O channel. Each pipe is fully identified by a port number. The pipe with the same port must also be initialized and periodically processed on the target MCU side in order the communication runs correctly. Refer to FreeMASTER Serial Driver documentation for more details.

6.3.43.2 Arguments

| | |
|--|---|
| vPort [in] | The port number which identifies the pipe channel. Only lower 16bits of the value are used. |
| vTxBufferSize [in, optional] | Local transmit buffer size. This buffer is used to accumulate data being transmitted to target MCU by calling any of PipeWrite functions until the MCU is ready to accept it. Optional, defaults to 0x1000. |
| vRxBufferSize [in, optional] | Local receive buffer size. This buffer is used to accumulate data being received from target MCU until the script reads them by calling any of PipeRead functions. Optional, defaults to 0x1000. |
| bsRetMsg [out, optional] | When an error occurs, this value contains an error message; otherwise, it is empty. Value of this output parameter is mirrored in LastRetMsg data member. Use it when the scripting language does not support [out] arguments. |

6.3.43.3 Return Value:

| | |
|---------------|---|
| Boolean Value | "True" is returned when pipe was successfully initialized. "False" is returned otherwise. |
|---------------|---|

6.3.44 PipeClose

```
PipeOpen (vPort);
```

6.3.44.1 Description

This function de-initializes the pipe object.

6.3.44.2 Arguments

| | |
|----------------------|---|
| vPort [in] | The port number which identifies the pipe channel. Only lower 16bits of the value are used. |
|----------------------|---|

6.3.44.3 Return Value:

| | |
|---------------|--|
| Boolean Value | <i>“True”</i> is returned when pipe was successfully closed. <i>“False”</i> is returned otherwise. |
|---------------|--|

6.3.45 PipeFlush

```
PipeFlush (vPort, vTimeout_ms);
```

6.3.45.1 Description

This function attempts to deliver pending data to target MCU. It also receives any new data waiting to be transmitted on the MCU side.

6.3.45.2 Arguments

| | |
|----------------------------|---|
| vPort [in] | The port number which identifies the pipe channel. Only lower 16bits of the value are used. |
| vTimeout_ms [in] | The maximum time in milliseconds to spend trying to deliver the pending data. When zero, a single attempt to deliver data is performed only. |

6.3.45.3 Return Value:

| | |
|---------------|--|
| Integer value | This function returns number of bytes which remains in the local transmit buffer after this flush attempt. |
|---------------|--|

6.3.46 PipeSetDefaultTxMode

```
PipeSetDefaultTxMode (vTxAllOrNothing);
```

6.3.46.1 Description

This function sets default transmission mode for subsequent calls to any PipeWrite function.

6.3.46.2 Arguments

| | |
|--------------------------------|---|
| vTxAllOrNothing [in] | Boolean value to be used as a default in next call to any of PipeWrite functions. |
|--------------------------------|---|

6.3.46.3 Return Value:

None.

6.3.47 PipeSetDefaultRxMode

```
PipeSetDefaultRxMode (vRxAllOrNothing, vRxTimeout_ms);
```

6.3.47.1 Description

This function sets default receiving mode for subsequent calls to any PipeRead function.

6.3.47.2 Arguments

| | |
|--------------------------------|--|
| vRxAllOrNothing [in] | Boolean value to be used as a default in next call to any of PipeRead functions. |
| vRxTimeout_ms [in] | Default timeout in milliseconds to used by subsequent PipeRead calls. |

6.3.47.3 Return Value:

None.

6.3.48 PipeSetDefaultStringMode

```
PipeSetDefaultStringMode (vUnicode);
```

6.3.48.1 Description

This function sets default string encoding used with PipeWriteString and PipeReadString functions.

6.3.48.2 Arguments

| | |
|-------------------------|---|
| vUnicode [in] | Boolean value to be used as a default in next call to PipeWriteString and PipeReadString functions. |
|-------------------------|---|

6.3.48.3 Return Value:

None.

6.3.49 PipeGetRxBytes

```
PipeGetRxBytes (vPort);
```

6.3.49.1 Description

This function returns number of bytes available in the local receive buffer to be read by any of PipeRead functions.

6.3.49.2 Arguments

| | |
|----------------------|---|
| vPort [in] | The port number which identifies the pipe channel. Only lower 16bits of the value are used. |
|----------------------|---|

6.3.49.3 Return Value:

| | |
|---------------|--|
| Integer value | This function returns number of bytes which are ready in the local receive buffer. |
|---------------|--|

6.3.50 PipeGetTxBytes

```
PipeGetTxBytes (vPort);
```

6.3.50.1 Description

This function returns number of bytes pending in the local transmit buffer.

6.3.50.2 Arguments

| | |
|----------------------|---|
| vPort [in] | The port number which identifies the pipe channel. Only lower 16bits of the value are used. |
|----------------------|---|

6.3.50.3 Return Value:

| | |
|---------------|---|
| Integer value | This function returns number of bytes which are pending in the local transmit buffer to be delivered to target MCU. |
|---------------|---|

6.3.51 PipeGetTxFree

```
PipeGetTxFree (vPort);
```

6.3.51.1 Description

This function returns free space in the local transmit buffer. Use this function before calls to any of PipeWrite functions to determine if the write would succeed.

6.3.51.2 Arguments

| | |
|----------------------|---|
| vPort [in] | The port number which identifies the pipe channel. Only lower 16bits of the value are used. |
|----------------------|---|

6.3.51.3 Return Value:

| | |
|---------------|--|
| Integer value | This function returns free space in the local transmit buffer. |
|---------------|--|

6.3.52 PipeGetRxBufferSize

```
PipeGetRxBufferSize (vPort);
```

6.3.52.1 Description

This function returns local receive buffer size. This is the size specified when initializing the pipe object with PipeOpen call.

6.3.52.2 Arguments

| | |
|----------------------|---|
| vPort [in] | The port number which identifies the pipe channel. Only lower 16bits of the value are used. |
|----------------------|---|

6.3.52.3 Return Value:

| | |
|---------------|----------------------------|
| Integer value | Local receive buffer size. |
|---------------|----------------------------|

6.3.53 PipeGetTxBufferSize

```
PipeGetTxBufferSize (vPort);
```

6.3.53.1 Description

This function returns local transmit buffer size. This is the size specified when initializing the pipe object with PipeOpen call.

6.3.53.2 Arguments

| | |
|----------------------|---|
| vPort [in] | The port number which identifies the pipe channel. Only lower 16bits of the value are used. |
|----------------------|---|

6.3.53.3 Return Value:

| | |
|---------------|-----------------------------|
| Integer value | Local transmit buffer size. |
|---------------|-----------------------------|

6.3.54 PipeWriteString

PipeWriteString (vPort, vString, vCharsToWrite, vAllOrNothing, vUnicode)

6.3.54.1 Description

This function writes characters from an input string into the selected pipe.

6.3.54.2 Arguments:

| | |
|--|--|
| vPort [in] | The port number which identifies the pipe channel. Only lower 16bits of the value are used. |
| vString [in] | The string to be written. |
| vCharsToWrite [in, optional] | The number of characters to be written to pipe. This is optional parameter, when omitted, the whole string is written (no termination character). |
| vAllOrNothing [in, optional] | When non-zero, the PipeWrite function will attempt to write all data at once to the local transmit buffer. When the data would not fit to the buffer, no data are transmitted. This is optional parameter, default value is determined by PipeSetDefaultTxMode function. |
| vUnicode [in, optional] | When non-zero, the individual characters will be written in Unicode encoding, two bytes per character. The Unicode characters are always written atomically, there is no risk of sending partial value. This is optional parameter, default value is determined by PipeSetDefaultStringMode function. |

6.3.54.3 Return Value:

| | |
|---------------|---|
| Integer value | This function returns number of characters written. |
|---------------|---|

6.3.55 PipeWriteXxxArray

To write array of 1,2, or 4-byte signed integers:

```
PipeWriteIntArray (vPort, vElemSize, arrData, vCount, vAllOrNothing)
```

To write array of 1,2, or 4-byte unsigned integers:

```
PipeWriteUIntArray (vPort, vElemSize, arrData, vCount, vAllOrNothing)
```

To write array of 4-byte floating point numbers:

```
PipeWriteFloatArray (vPort, arrData, vCount, vAllOrNothing)
```

To write array of 8-byte floating point numbers:

```
PipeWriteDoubleArray (vPort, arrData, vCount, vAllOrNothing)
```

6.3.55.1 Description

These functions translate a VB array of numbers passed by the caller into a block of bytes and write the data into the selected pipe. The individual array members are always written atomically to the transmit buffer, so there is no risk of having any of the value partially transmitted.

6.3.55.2 Arguments:

| | |
|--|---|
| vPort [in] | The port number which identifies the pipe channel. Only lower 16bits of the value are used. |
| vElemSize [in] | The size of an array element in bytes. The total number of bytes to be written to the target can be calculated as vCount * vElemSize. |
| arrData [in] | The array of the values to be written. |
| vCount [in, optional] | The number of elements to be written to pipe. This is optional parameter, when omitted, the whole array is written. |
| vAllOrNothing [in, optional] | When non-zero, the PipeWrite function will attempt to write all data at once to the local transmit buffer. When the data would not fit to the buffer, no data are transmitted. This is optional parameter, default value is determined by PipeSetDefaultTxMode function. |

6.3.55.3 Return Value:

| | |
|---------------|--|
| Integer value | This function returns number of array elements successfully written. |
|---------------|--|

6.3.56 PipeReadString

PipeReadString (vPort, vRxTimeout_ms, vCharsToRead, vAllOrNothing, vUnicode, vRetString)

6.3.56.1 Description

This function writes characters from an input string into the selected pipe.

6.3.56.2 Arguments:

| | |
|--|--|
| vPort [in] | The port number which identifies the pipe channel. Only lower 16bits of the value are used. |
| vRxTimeout_ms [in, optional] | Timeout in milliseconds to wait for requested number of characters to be received. This is optional parameter, default value is determined by PipeSetDefaultRxMode. |
| vCharsToRead [in, optional] | The number of characters to be read from pipe. This is optional parameter, when omitted, the maximum number of characters available until the timeout expires are read. |
| vAllOrNothing [in, optional] | When non-zero and vCharsToRead count is not zero, the function will attempt to read all required data at once. When the data are not available until the timeout expires, no data are received. This is optional parameter, default value is determined by PipeSetDefaultRxMode function. |
| vUnicode [in, optional] | When non-zero, the individual characters will be read in Unicode encoding, two bytes per character. The Unicode characters are always read atomically, there is no risk of receiving partial value. This is optional parameter, default value is determined by PipeSetDefaultStringMode function. |
| vRetString [out, optional] | This parameter variable receives the string. Value of this output parameter is mirrored in LastPipe_data member. Use it when the scripting language does not support [out] arguments. |

6.3.56.3 Return Value:

| | |
|---------------|--|
| Integer value | This function returns number of characters read. |
|---------------|--|

6.3.57 PipeReadXxxArray

To read 1,2, or 4-byte signed integers:

```
PipeReadIntArray (vPort, vElemSize, vRxTimeout_ms, vCount, vAllOrNothing, arrData)
PipeReadIntArray_v (vPort, vElemSize, vRxTimeout_ms, vCount, vAllOrNothing, arrData)
PipeReadIntArray_t (vPort, vElemSize, vRxTimeout_ms, vCount, vAllOrNothing, arrData)
```

To read 1,2, or 4-byte unsigned variable:

```
PipeReadUIntArray (vPort, vElemSize, vRxTimeout_ms, vCount, vAllOrNothing, arrData)
PipeReadUIntArray_v (vPort, vElemSize, vRxTimeout_ms, vCount, vAllOrNothing, arrData)
PipeReadUIntArray_t (vPort, vElemSize, vRxTimeout_ms, vCount, vAllOrNothing, arrData)
```

To access 4-byte floating point variable:

```
PipeReadFloatArray (vPort, vRxTimeout_ms, vCount, vAllOrNothing, arrData)
PipeReadFloatArray_t (vPort, vRxTimeout_ms, vCount, vAllOrNothing, arrData)
```

To access 8-byte floating point variable:

```
PipeReadDoubleArray (vPort, vRxTimeout_ms, vCount, vAllOrNothing, arrData)
PipeReadDoubleArray_t (vPort, vRxTimeout_ms, vCount, vAllOrNothing, arrData)
```

6.3.57.1 Description

These functions reads a requested count of integer or floating point numbers from a pipe and store them into an destination VB array. For each call, there is an optional function which can be used in different scripting environments:

- *PipeReadXxxArray* returns array elements as variants suitable for both scripting languages like VBScript and for compiled languages like Visual Basic. As the VBScript engine does not handle variants encapsulating 2 and 4-byte unsigned integer types, this method converts such a values to floating point format before returning.
- *PipeReadXxxArray_v* is the same as *PipeReadXxxArray* except it does not perform the VBScript translation for 2 and 4-byte unsigned integer types.
- *PipeReadXxxArray_t* returns data in the VB array of strictly typed values, which can be used in compiled languages like Visual Basic. Using typed arrays is significantly faster than using arrays of variants.

6.3.57.2 Arguments:

| | |
|--|--|
| vPort [in] | The port number which identifies the pipe channel. Only lower 16bits of the value are used. |
| vElemSize [in] | The size of an array element in bytes. The total number of bytes to be read from pipe can be calculated as vCount * vElemSize. |
| vRxTimeout_ms [in, optional] | Timeout in milliseconds to wait for requested number of characters to be received. This is optional parameter, default value is determined by PipeSetDefaultRxMode. |
| vCount [in, optional] | The number of elements to be written to pipe. This is optional parameter, when omitted, the maximum number of values available until the timeout expires are read. |
| vAllOrNothing [in, optional] | When true and vCount count is not zero, the function will attempt to read all required data at once. When the data are not available until the timeout expires, no data are received. This is optional parameter, default value is determined by PipeSetDefaultRxMode function. |

| | |
|-----------------------------------|--|
| arrData [out, optional] | This parameter variable receives the resulting VB array. Value of this output parameter is mirrored in LastPipe_data member. Use it when the scripting language does not support [out] arguments. |
|-----------------------------------|--|

6.3.57.3 Return Value:

| | |
|---------------|--|
| Integer value | This function returns number of array elements read. |
|---------------|--|

6.3.58 EnumVariables

```
EnumVariables (vIndex, bsVariableName);
```

6.3.58.1 Description

This function can be used to enumerate FreeMASTER variables in the current project. User script is expected to call this function with increasing vIndex value until an invalid (false) value is returned.

6.3.58.2 Arguments

| | |
|--|---|
| vIndex [in] | Index of variable to be retrieved. User is expected to call this function with vIndex increasing from 0 until the function returns false value. |
| bsVariableName [out, optional] | Variable name. Value of this output parameter is mirrored in LastEnum_name data member. Use it when the scripting language does not support [out] arguments. |

6.3.58.3 Return Value:

| | |
|---------------|--|
| Boolean value | False when vIndex does not map to any valid variable object and the enumeration should be stopped. |
|---------------|--|

6.3.59 EnumSymbols

```
EnumSymbols (vIndex, bsSymbolName);
```

6.3.59.1 Description

This function can be used to enumerate target application symbols as loaded in the current project. User script is expected to call this function with increasing vIndex value until an invalid (false) value is returned.

6.3.59.2 Arguments

| | |
|--|---|
| vIndex [in] | Index of symbol to be retrieved. User is expected to call this function with vIndex increasing from 0 until the function returns false value. |
| bsVariableName [out, optional] | Symbol name. Value of this output parameter is mirrored in LastEnum_name data member. Use it when the scripting language does not support [out] arguments. |

6.3.59.3 Return Value:

| | |
|---------------|--|
| Boolean value | False when vIndex does not map to any valid symbol object and the enumeration should be stopped. |
|---------------|--|

6.3.60 GetDetectedBoardInfo

GetDetectedBoardInfo ()

6.3.60.1 Description

This function can be used to retrieve information obtained during initial FreeMASTER protocol handshake between Host PC and target MCU board.

6.3.60.2 Arguments

There are no arguments, this function fills the output information into LastBoardInfo_xxx data members as follows:

| | |
|-----------------------------------|---|
| LastBoardInfo_protVer | Protocol version byte value. |
| LastBoardInfo_dataBusWidth | Data bus width, typically 1 on most of the MCU platforms. May be 2 on some DSC platforms. |
| LastBoardInfo_versionWord | FreeMASTER driver version word. |
| LastBoardInfo_cmdBuffSize | Size of the FreeMASTER serial communication buffer. |
| LastBoardInfo_recBuffSize | Size of the data buffer for Recorder use. |
| LastBoardInfo_recTimeBase | Base Recorder sampling rate encoded in units as described in Serial Driver documentation. |
| LastBoardInfo_description | Board or FreeMASTER driver description string. |

6.3.60.3 Return Value:

| | |
|---------------|---|
| Boolean value | True when board is detected and information retrieved. False otherwise. |
|---------------|---|

6.3.61 GetCommPortInfo

GetCommPortInfo ()

6.3.61.1 Description

This function can be used to retrieve information about current communication port settings.

6.3.61.2 Arguments

There are no arguments, this function fills the output information into LastCommPortInfo_xxx data members as follows:

| | |
|---------------------------------------|--|
| LastCommPortInfo_description | Friendly name of the communication port. |
| LastCommPortInfo_connectString | Full connection string used by FreeMASTER to open the port. |
| LastCommPortInfo_isPlugin | True when communication plug-in is used. |
| LastCommPortInfo_isRS232 | True when native serial port or USB-to-serial port is used. |
| LastCommPortInfo_speed | Serial baudrate used. Valid only if isRS232 member is true. |
| LastCommPortInfo_portNum | COM port number. Valid only if isRS232 member is true. |
| LastCommPortInfo_portHint | Hint word used to lookup the serial port. Valid only if isRS232 member is true. Hint may be any word contained in the serial port description and may be specified by user instead of direct COM port number. This FreeMASTER feature may be useful when using USB-to-Serial converters which map to random COM port numbers. |

6.3.61.3 Return Value:

| | |
|---------------|---|
| Boolean value | True when board is detected and information retrieved. False otherwise. |
|---------------|---|

6.4 FreeMASTER ActiveX Properties

Starting version 1.2.20, the FreeMASTER also supports scripting languages which do not implement by-reference passing of the function parameters (e.g. JScript). The “output” parameters in all ActiveX methods are declared as “optional” and do not need to be specified by the caller at the time the FreeMASTER method is called. After the call is made, all output values can be fetched using ActiveX properties as described in [Table 6-1](#).

For each property in the FreeMASTER ActiveX interface, there is also a “GetLast...” function available and implements the same functionality as when reading the property value itself. Such functions can be used in scripting environment where accessing ActiveX object properties is not possible or convenient.

Table 6-1. FreeMASTER ActiveX Object Properties

| Property Name | Description |
|--------------------------|--|
| LastResult | The return value of the last ActiveX function called |
| LastRetMsg | The error message returned by the last ActiveX function called (the value of bsRetMsg output parameter) |
| LastVariable_vValue | The value of the “vValue” output parameter returned by the last ReadVariable method called |
| LastVariable_tValue | The value of the “tValue” output parameter returned by the last ReadVariable method called |
| LastMemory_hexstr | The value of the “bsRet” output parameter returned by the last ReadMemoryHex method called |
| LastMemory_data | The array of values returned in “arrData” output parameter by the last call to one of the ReadMemory or ReadXxxArray methods. |
| LastRecorder_data | The array of values returned in “arrData” output parameter by the last call to GetCurrentRecorderData or GetCurrentRecorderSeries methods. |
| LastRecorder_serieNames | The array of values returned in “arrSerieNames” output parameter by the last call to GetCurrentRecorderData method. |
| LastRecorder_timeBaseSec | The value of the “timeBaseSec” output parameter returned by the last call to GetCurrentRecorderData method. |
| LastRecorder_state | The value of the “nState” output parameter returned by the last call to GetCurrentRecorderState method. |
| LastLocalFile_string | The “bsRetString” text buffer returned by the last call to LocalFileReadString method. |
| LastSymbolInfo_size | The “vSymSize” value returned by the last call to GetSymbolInfo method. |
| LastSymbolInfo_addr | The “vSymAddr” value returned by the last call to GetSymbolInfo method. |
| LastMemberInfo_size | The “vMembSize” value returned by the last call to GetStructMember method. |
| LastMemberInfo_offset | The “vMembOff” value returned by the last call to GetStructMember method. |
| LastAddressInfo_name | The “bsSymbolName” value returned by the last call to GetAddressInfo method. |
| LastAddressInfo_exact | The “blsExactMatch” value returned by the last call to GetAddressInfo method. |
| LastLinkName | The name returned by the last call to EnumHrefLinks or EnumProjectFiles methods. |

Table 6-1. FreeMASTER ActiveX Object Properties

| Property Name | Description |
|----------------------|--|
| LastPipe_data | The pipe data returned by the last call to any of PipeRead methods. |
| LastEnum_name | The enumerated name returned by the last call to EnumVariables or EnumSymbols methods. |
| LastBoardInfo_xxx | Multiple properties returned by the GetDetectedBoardInfo method. |
| LastCommPortInfo_xxx | Multiple properties returned by the GetCommPortInfo method. |

6.5 FreeMASTER ActiveX Object Events

6.5.1 OnRecorderDone

`OnRecorderDone ()`

6.5.1.1 Description

This event is called when the active recorder finishes downloading of a new data.

6.5.2 OnCommPortStateChanged

`OnCommPortStateChanged (vPortOpen)`

6.5.2.1 Description

This event is called when the communication port is open (`vPortOpen` is "True") or closed (`vPortOpen` is "False").

6.5.3 OnBoardDetected

`OnBoardDetected ()`

6.5.3.1 Description

This event is called when the FreeMASTER detects a valid target board and gets the first handshake information from it.

6.5.4 OnVariableChanged

`OnVariableChanged (bsVarName, vSubscriptionId)`

6.5.4.1 Description

This event is called when a subscribed variable value changes. See more details in *SubscribeVariable* method description.

6.6 Examples

Example applications shown in this section demonstrates how to initialize and use ActiveX object functions in different scripting environments. For all examples, have the target board connected and running the default demo application distributed within the FreeMASTER Serial Driver package.

6.6.1 VB Script Embedded in HTML Page

The VB Script is the default scripting language used in HTML pages. It natively supports by-reference [output] parameters returned from ActiveX methods. Using the FreeMASTER object with VB Script is very straightforward.

```
<BODY>
  <!-- create FreeMASTER ActiveX object "fmstr" -->
  <OBJECT id="fmstr" height="0" width="0" classid="clsid:48A185F1-FFDB-11D3-80E3-00C04F176153">
  </OBJECT>
  <script language="VBScript">
  Sub window_onload()
    Window.SetTimeout "PageTimer", 100
  End Sub

  Function PageTimer()
    'performs Read command from variable "var16" defined in project for FreeMASTER tool
    '!" - immediate performs the command
    bSucc = fmstr.ReadVariable("var16", vValue, tValue, bsRetMsg)
    If bSucc then
      varRefresh.InnerText = tValue
    else
      varRefresh.InnerText = bsRetMsg
    End If
    Window.SetTimeout "PageTimer", 100
  End Function

  Function ButtonRead()
    'writes 0x10 value into "var16inc" defined project for FreeMASTER tool
    bSucc = fmstr.WriteVariable("var16inc", 10, bsRetMsg)
  End Function
  </script>

  <h1>ActiveX example application</h1>
  periodic variable read: <span id = "varRefresh"> N/A </span><br />
  <input type="button" value="RunScript" onclick="ButtonRead()">
</BODY>
```

6.6.2 JScript Embedded in HTML Page

JScript is one of the most popular scripting languages for dynamic HTML pages. There are some special techniques needed to use it with FreeMASTER ActiveX control:

- JScript does not natively support by-reference [output] parameters. The parameters should be avoided and the output values should be retrieved from the LastResult and other Lastxxx property members. Refer to [Section 6.4, "FreeMASTER ActiveX Properties"](#) for more details.
- JScript uses different format of arrays, which is not compatible with "safearray" format used by ActiveX interface. A special conversion routines need to be used as demonstrated in the example code below.

```
<BODY onload="PageInit()">
  <!-- create FreeMASTER ActiveX object "fmstr" -->
  <OBJECT id="fmstr" height="0" width="0" classid="clsid:48A185F1-FFDB-11D3-80E3-00C04F176153">
  </OBJECT>
  <SCRIPT>
    function PageInit()
    {
      // create timebase 100ms, calls PageTimer every 100ms.
      setInterval(PageTimer, 100);
    }
  </SCRIPT>
```

```

function PageTimer()
{
    var result;
    var bSucc;
    // read the "var16" variable as defined in FreeMASTER project
    bSucc = fmstr.ReadVariable("!var16"); // "!" - immediate performs the command
    if(bSucc)
    {
        //store variable value from last call of ReadVariable()
        result = fmstr.LastVariable_vValue;
        document.getElementById("varRefresh").innerHTML = result;
    }
}

function ButtonRead()
{
    var result;
    var bSucc;
    // write 0x10 value into "var16inc" variable as defined in FreeMASTER project
    bSucc = fmstr.WriteVariable("var16inc", 0x10);

    // write 4 bytes to memory at given address. WriteMemory function expects SafeArray as input data
    bSucc = fmstr.WriteMemory(0x0030, 4, getSafeArray([11,22,33,44]));
    // reads 4 bytes from memory. var32inc represents an address of var32inc variable
    bSucc = fmstr.ReadMemory("var32inc", 4);
    if(bSucc)
    {
        //reads data of last call the ReadMemory()function. Data has to be converted to array.
        result = fmstr.LastMemory_data.toArray();
        alert(result); //show array
    }

    // write to memory at pointer. The value of "var8" represents the pointer, 0x02 is offset.
    // this is only demonstration !!Beware that wrong value of var8 may cause memory corruption!!
    bSucc = fmstr.WriteUIntArray("valueof(var8) + 0x02", 4, 1, getSafeArray([22,44,66,88]));
}

//function converts Array to SafeArray
function getSafeArray(jsArr) {
    var dict = new ActiveXObject("Scripting.Dictionary");
    for (var i = 0; i < jsArr.length; i++)
        dict.add(i, jsArr[i]);
    return dict.Items();
}
</SCRIPT>
periodic variable read: <span id = "varRefresh"> N/A </span><br />
<input type="button" value="RunScript" onclick="ButtonRead()">

</BODY>

```

6.6.3 VBA Script in Excel

The FreeMASTER object can also be used in Visual Basic for Applications (VBA) project in Excel. You need to register a reference to the FreeMASTER services in *Tools->References...* dialog.

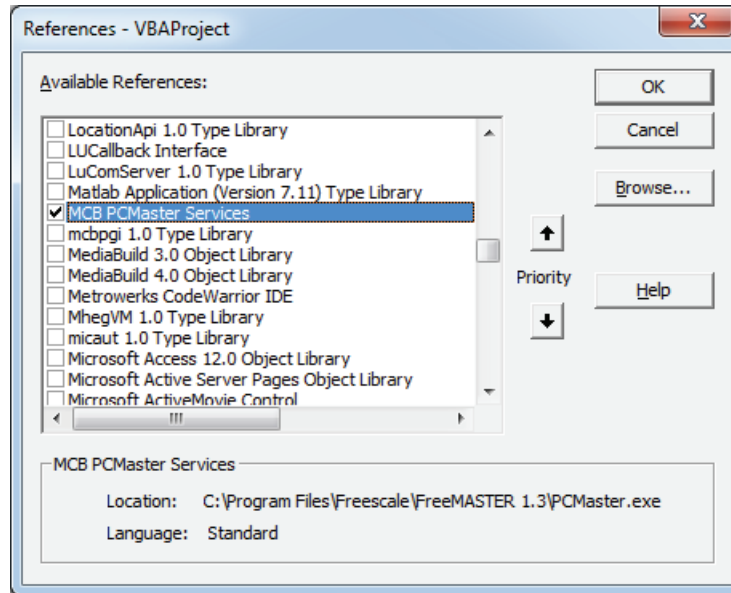


Figure 6-1. Registering the FreeMASTER Object in VBA Environment

```

'*****
'* Module1: *
'*****

Public ScheduleTime As Variant
Sub RunFmstrExample()
    'initialize FreeMASTER object
    Dim fmstr As McbPcm
    Set fmstr = New McbPcm

    Dim bSucc As Boolean
    Set sht = Worksheets("sheet1")
    ' write content of the sheet A1 cell into "var16inc" variable as defined in FreeMASTER project
    bSucc = fmstr.WriteVariable("var16inc", sht.Cells(1, 1).Value, bsRetMsg)

    ' read variable "var16" and store the value in the A2 cell
    ' "!" - immediate performs the command
    bSucc = fmstr.ReadVariable("!var16", tValue, bsRetMsg)
    If bSucc Then
        sht.Cells(1, 2).Value = tValue
    Else ' something failed
        MsgBox (bsRetMsg)
    End If

    Dim resultArray
    ' read 4 bytes from memory occupied by variable var32inc
    bSucc = fmstr.ReadMemory("var32inc", 4, resultArray, bsRetMsg)
    If bSucc Then
        sht.Cells(2, 1).Value = resultArray(0)
        sht.Cells(2, 2).Value = resultArray(1)
        sht.Cells(2, 3).Value = resultArray(2)
        sht.Cells(2, 4).Value = resultArray(3)
    Else
        MsgBox (bsRetMsg)
    End If

```

```
ScheduleTime = Now + TimeValue("00:00:05")
Application.OnTime ScheduleTime, "RunFmstrExample"
End Sub
```

6.6.4 Matlab m-script

Test this code at Matlab console

```
% create FreeMASTER ActiveX object
fmstr = actxserver ('MCB.PCM');

% write 0x10 value into "var16" variable defined in FreeMASTER project
bSucc = fmstr.WriteVariable('var16inc', 16);
var16 = 0;

% read the "var16" variable as defined in FreeMASTER project
% "!" - immediate performs the command
bSucc = fmstr.ReadVariable('var16');
if bSucc
    var16 = fmstr.LastVariable_vValue;
end
% show the value
var16

% configure matlab to accept safe array as single dimension
feature('COM_SafeArraySingleDim', 1) ;
% write 4 bytes to memory 0x20. WriteMemory function expects SafeArray as input data
bSucc = fmstr.WriteMemory(32, 4, {11;22;33;44})
% reads 4 bytes from memory at address of var32inc variable
bSucc = fmstr.ReadMemory('var32inc', 4);
if bSucc
    % reads data of last call the ReadMemory()function.
    readMemResult = fmstr.LastMemory_data
end

% write to memory at pointer. The value of "var8" represents the pointer, 0x02 is offset.
% this is only demonstration !!Beware that wrong value of var8 may cause memory corruption!!
array = [11;22;33;44];
bSucc = fmstr.WriteUIntArray('valueof(var8) + 0x02', 4, 1, {array});
```


Appendix A

References

- [1] *The FreeMASTER Tool home page at http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=FREEMASTER*
- [2] *FreeMASTER Serial Communication Driver User Manual. Installed with the FMASTERSCIDRV software from <https://www.freescale.com/webapp/sps/download/license.jsp?colCode=FMASTERSCIDRV>*
- [3] *The DSP56800E_Quick_Start Initialization and Development Tool home page at http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=DSP56800EQUICKSTART*

Appendix B Revision History

The following revision history table summarizes changes contained in this document.

| Revision | Date | Description |
|----------|---------|--|
| 0 | 2/2002 | Published as 56F800 SDK document SDK111/D |
| 1.0 | 6/2004 | Updated for FreeMASTER Application. Made as separate document. |
| 2.0 | 9/2007 | Updated for FreeMASTER version 1.3. New application screen shots used. New Freescale document template used. |
| 2.1 | 6/2011 | Updated comments for GetCurrentRecorderState function. |
| 2.2 | 10/2012 | Added description of new ActiveX methods, added examples of ActiveX object use. |
| 2.3 | 03/2014 | New ActiveX methods documented for version 1.4.1 |
| 2.4 | 06/2014 | Removed obsolete license text. See installation package for up-to-date license file. |
| 2.5 | 05/2016 | New ActiveX methods documented for version 2.0.2 |

How to Reach Us:

Home Page:

www.freescale.com

Web Support:

<http://www.freescale.com/support>

USA/Europe or Locations Not Listed:

Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2006. All rights reserved.

