NXP Semiconductors
User's Guide

Document Number: FMSTERUG
Rev. 3.0, 06/2019

FreeMASTER for Embedded Applications

Contents

Chapter 1 Introduction

1.1 Overview

This User Manual describes the FreeMASTER application developed by NXP's engineers to control an embedded application using a graphical environment running on a PC. The application was initially created for developers of real-time motor-control applications, but many users found it very useful for their specific 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 Windows® OS-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 the programming style, other software drivers requirements, and the way the drivers were configured.

In March 2006, Freescale released an SCI driver common for the majority of Freescale MCU platforms. Today, the driver supports MC56F800E, HC08, HCS08, HC12, S12, S12X, S12Z, ColdFire, Power Architecture, and Kinetis families of MCUs. 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

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

The **BDM Communication Plug-in** enables basic memory access operations to be performed by the FreeMASTER on 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 fully-featured communication with the 56F800E hybrid MCUs 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 the JTAG/EOnCE, which is very similar to the SCI communication.

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

For more details about the communication plug-in modules, see 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 NXP web pages. One is for the PC-side application (FMSTRSW) and the other one is for the new SCI driver for the embedded side (FMSTRSCIDRV). Both packages are available at the FreeMASTER home page (www.nxp.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, and others)
- 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)
- 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, and others)
- 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 connection to a target board over a network, including the Internet

Chapter 2 Questions and answers

While writing this user manual, the following questions were raised. Because the answers to these questions clarify the terms and topics described further in the manual, this section comes before the sections that are more detailed and perhaps more difficult to understand.

2.1 Why do I need FreeMASTER?

The primary goal of developing FreeMASTER was to deliver a tool for debugging and demonstrating of motor-control algorithms and applications. The result is a versatile tool that can be used for multi-purpose algorithms and applications. Some real-world uses include:

- Real-time debugging—FreeMASTER enables the users to debug applications in a true real-time fashion through its ability to watch variables. Moreover, it allows debugging at the algorithm level, which helps to shorten the development phase.
- Diagnostic tool—FreeMASTER's 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 to demonstrate the algorithm or application execution and variable outputs.
- Education—FreeMASTER may be used for educational purposes. Its application control features enable students to play with the application in the demonstration mode, learning how to control program execution.

2.2 What does FreeMASTER 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 user-friendly format:

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

2.3 Why is FreeMASTER 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 can be explored to observe how they affect output parameters. Each block has a description tab to explain algorithm details using a multimedia-capable and scriptable HTML format.

2.4 What can I do with FreeMASTER 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 toying with the project's defined blocks and parameters. The demo project enables 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 they can be used in your application. For a detailed explanation of the parameters, see FreeMASTER usage on page 9.

2.5 How is FreeMASTER connected to a target development board?

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

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

2.6 What are all of these dialog boxes for?

In FreeMASTER usage on page 9, there are pictures with dialog boxes. These dialog boxes are used as a questionnaire, where you enter the parameters that describe, 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
- To learn about your application during operator training phase

2.8 How do I set up remote control and 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 the 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 a user-friendly 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 the software on the target development board, and stores the changes of variables in real time. You can define the list of variables to record by the embedded-side timer periodic interrupt service routine. After the requested variable samples are stored within the Recorder buffer on the target board, they are downloaded from the board and displayed in the 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?

The FreeMASTER Oscilloscope is similar to a standard hardware oscilloscope. It shows the selected variables graphically 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 met by almost any Windows OS-based host PC available today. It was tested with the latest versions of Windows OS (7, 8, and 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

Hard drive space: 20 MB

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

3.2 Enabling FreeMASTER connection on the target application

To enable the FreeMASTER connection on the target board application, follow the instructions provided with the embedded-side driver (FMASTERSCIDRV). The recommended and fastest way to start using FreeMASTER is by running a 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.

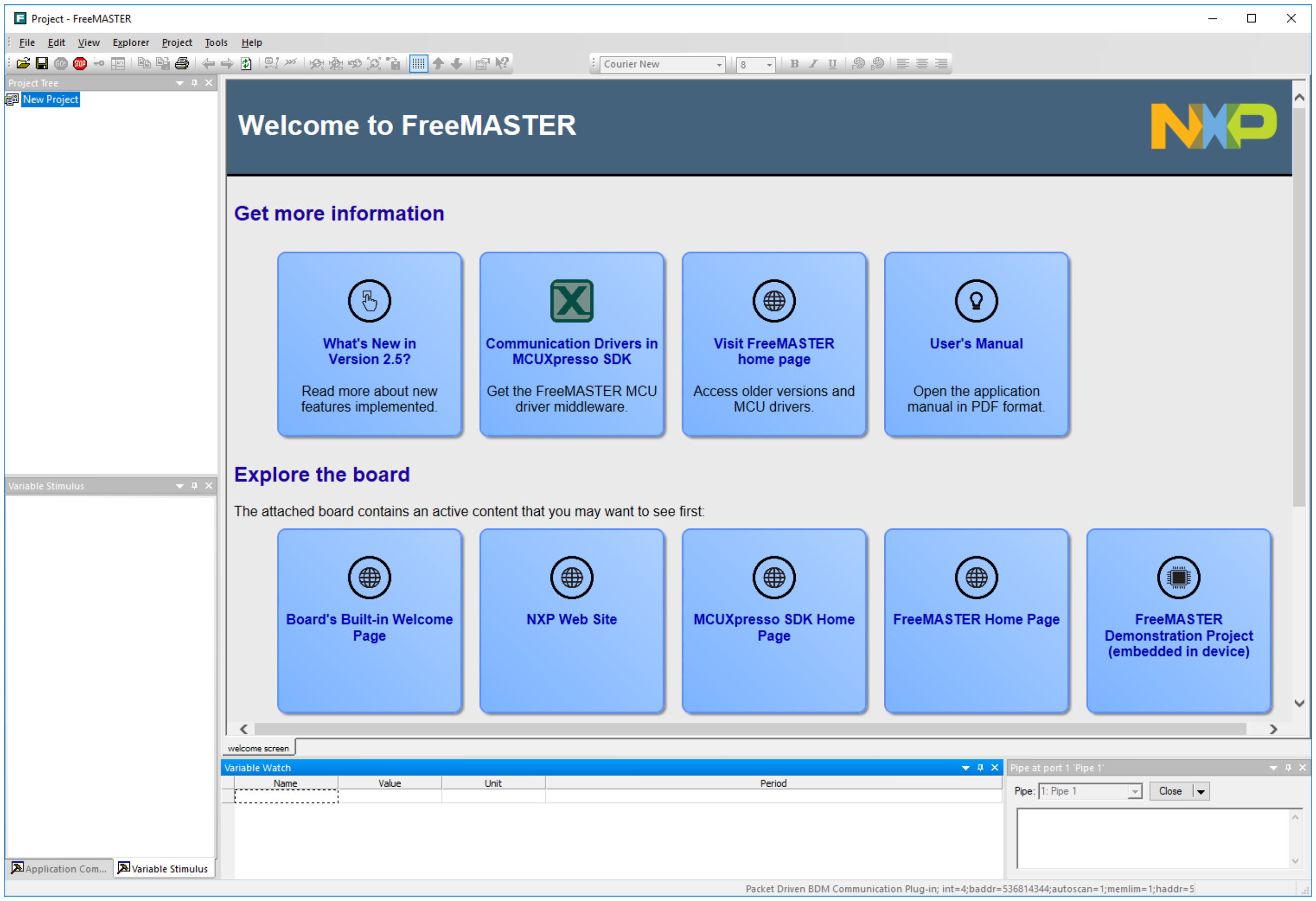
3.3 How to install FreeMASTER

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

Chapter 4
FreeMASTER usage

4.1 Application window description

When the application starts, 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 this figure:



Welcome to FreeMASTER

Get more information

What's New in Version 2.5? Read more about new features implemented.

Communication Drivers in MCUXpresso SDK. Get the FreeMASTER MCU driver middleware.

Visit FreeMASTER home page. Access older versions and MCU drivers.

User's Manual. Open the application manual in PDF format.

Explore the board

The attached board contains an active content that you may want to see first:

Board's Built-in Welcome Page

NXP Web Site

MCUXpresso SDK Home Page

FreeMASTER Home Page

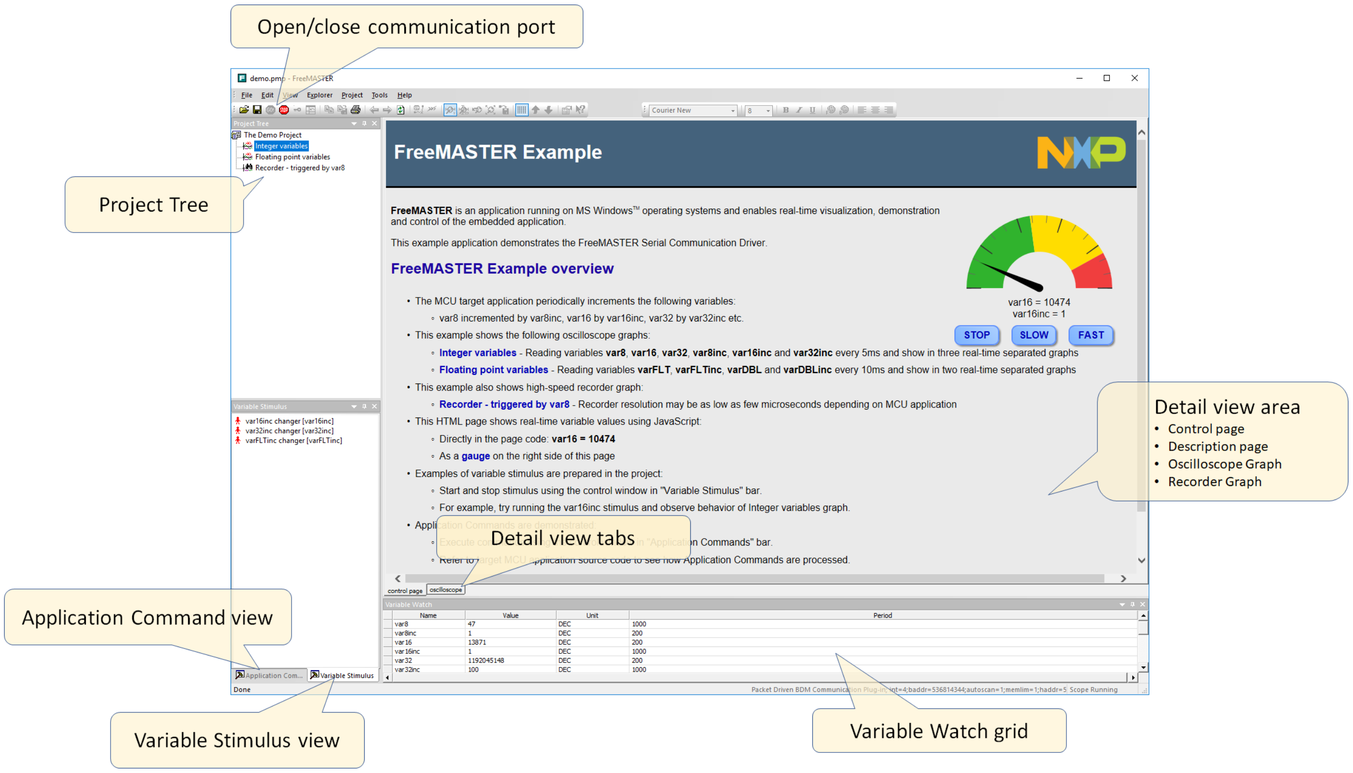
FreeMASTER Demonstration Project (embedded in device)

Packet Driven BDM Communication Plug-in; int=4;baddr=536814344;autoscan=1;memlim=1;haddr=5

Figure 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, the application usage is demonstrated using an example of a simple demo application, which is a part of the SCI Driver installation (FMASTERSCIDRV).



FreeMASTER Example

FreeMASTER is an application running on MS Windows™ operating systems and enables real-time visualization, demonstration and control of the embedded application.

This example application demonstrates the FreeMASTER Serial Communication Driver.

FreeMASTER Example overview

- The MCU target application periodically increments the following variables:
  - var8 incremented by var8inc, var16 by var16inc, var32 by var32inc etc.
- This example shows the following oscilloscope graphs:
  - **Integer variables** - Reading variables **var8**, **var16**, **var32**, **var8inc**, **var16inc** and **var32inc** every 5ms and show in three real-time separated graphs
  - **Floating point variables** - Reading variables **varFLT**, **varFLTinc**, **varDBL** and **varDBLinc** every 10ms and show in two real-time separated graphs
- This example also shows high-speed recorder graph:
  - **Recorder - triggered by var8** - Recorder resolution may be as low as few microseconds depending on MCU application
- This HTML page shows real-time variable values using JavaScript:
  - Directly in the page code: **var16 = 10474**
  - As a **gauge** on the right side of this page
- Examples of variable stimulus are prepared in the project:
  - Start and stop stimulus using the control window in "Variable Stimulus" bar.
  - For example, try running the var16inc stimulus and observe behavior of Integer variables graph.
- Application Commands are demonstrated:
  - Execute co in "Application Commands" bar.
  - Refer to target MCU application source code to see how Application Commands are processed.

var16 = 10474
var16inc = 1

Variable Watch

Name	Value	Unit	Period
var8	47	DEC	1000
var8inc	1	DEC	200
var16	13871	DEC	200
var16inc	1	DEC	1000
var32	1192045148	DEC	200
var32inc	100	DEC	1000

Packet Driven BDM Communication Plug-in; int=4;baddr=536814344;autoscan=1;memlim=1;haddr=5

Figure 2. Application window

As shown in the above figure, 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 on in View menu on page 35.

The*Project Tree* pane contains a logical tree structure of the application being monitored/controlled. You can add the project sub-blocks, Scope, and Recorder definitions to the project block in a logical structure to form a *Project Tree*. This pane provides the 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 with the item.

- *Control page* = An HTML page created to control 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 the application variables, as defined in the Scope properties.
- *Recorder* = A graph displaying the recorded application variables, as defined in the Recorder properties.
- The *control page* (when defined) is available for all *Project Tree* items to enable 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 the Scope or the 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 enables you to change them (if it 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* and *Detail View* HTML pages, real-time chart definitions, watch interface settings, variables, commands, stimulators, and more.

4.1.1 Project Tree

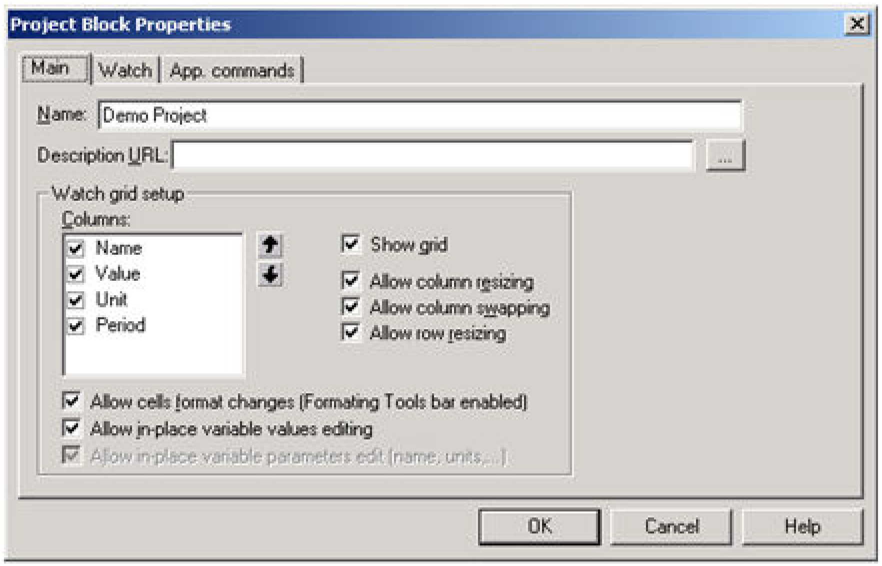
When a new project is created, the *Project Tree* window contains an empty structure with just one root project block called *"New Project"*. You can change the properties of this block or add sub-blocks, Scopes, or Recorders to the structure.

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

- Select an item in the *Project Tree* and right-click it to use the local menu.
- Select an item in the *Project Tree* and select the 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. The *sub-blocks* may be added when you 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 this figure:


Project Block Properties

Main | Watch | App. commands

Name: Demo Project

Description URL:

Watch grid setup

Columns:

- [✓] Name
- [✓] Value
- [✓] Unit
- [✓] Period

- [✓] Show grid
- [✓] Allow column resizing
- [✓] Allow column swapping
- [✓] Allow row resizing

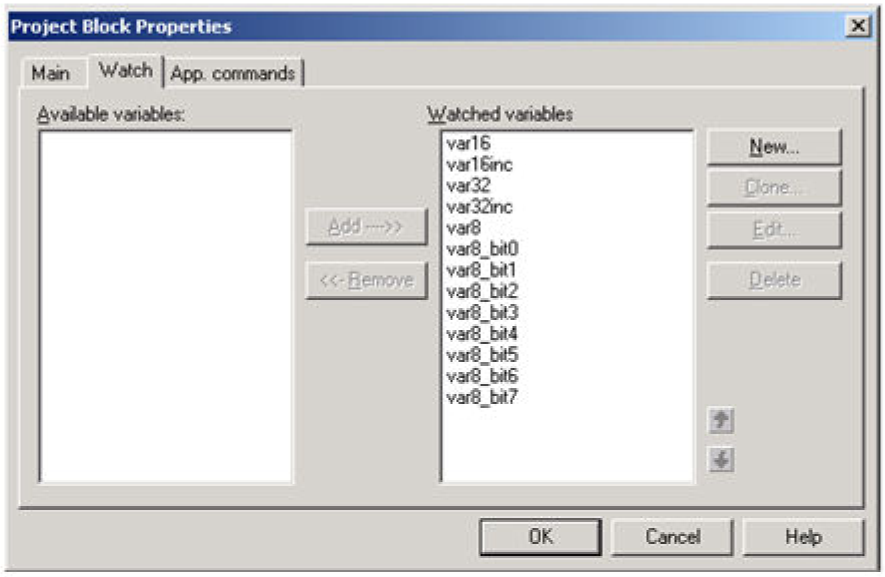
- [✓] Allow cells format changes (Formating Tools bar enabled)
- [✓] Allow in-place variable values editing
- [✓] Allow in-place variable parameters edit (name, units,...)

Figure 3. Project Block Properties window—Main page

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

- *Name* = Name of the project block that is displayed in the *Project Tree*.
- *Description URL* = Select a description URL or a path to *.htm* or *.html* files 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 the MS Front Page Express or Netscape Composer. With the demo application used as our example, the description page is left empty, causing the "Algorithm Block Description" tab to be hidden. See Detail View on page 21 for more details.
- *Watch-grid setup* = You can select the 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 the grid cells with Toolbar (see Watch Bar on page 38), and edit the in-place variable values by checking the next option boxes.

The *Watch* page shown in the following figure selects which FreeMASTER project variables are to be watched in the context of this project block.



var16
var16inc
var32
var32inc
var8
var8_bit0
var8_bit1
var8_bit2
var8_bit3
var8_bit4
var8_bit5
var8_bit6
var8_bit7

Figure 4. Project Block Properties window—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 Variables on page 24)
- *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 the so-called “application commands” (see Commands on page 28). 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 the following figure, sets which commands are available in the *Fast Access* pane in the context of this project block and also enables the management of application commands.

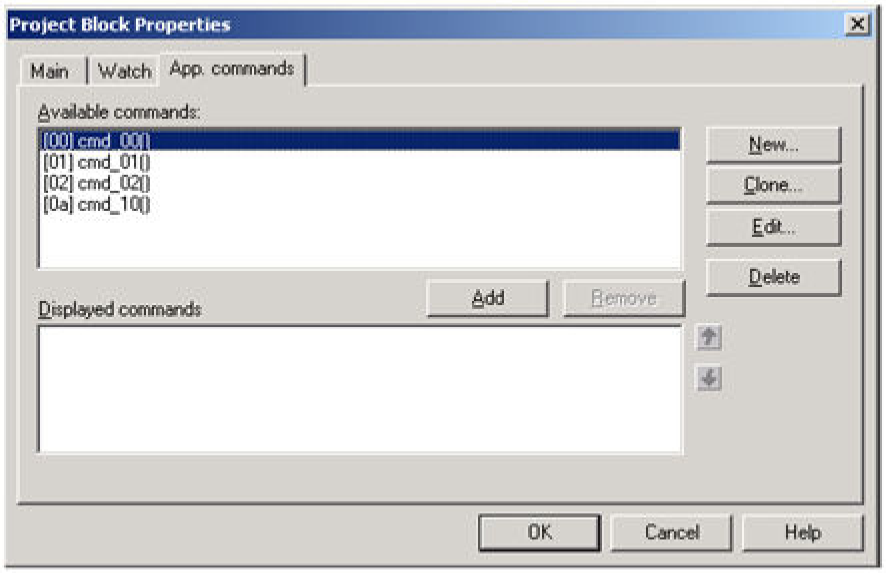


Figure 5. Project Block Properties window—App. commands page

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 Commands on page 28)
- *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 the 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 the following figure, 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 is displayed in the *Project Tree*.
- *Description URL* = Specify the URL of the document or a 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.

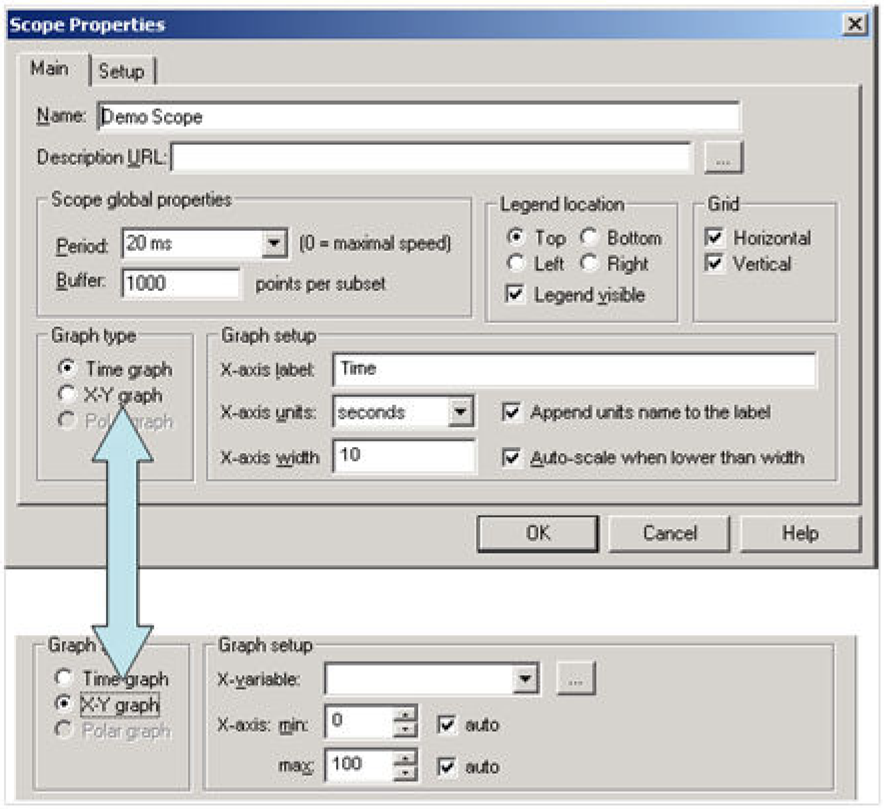
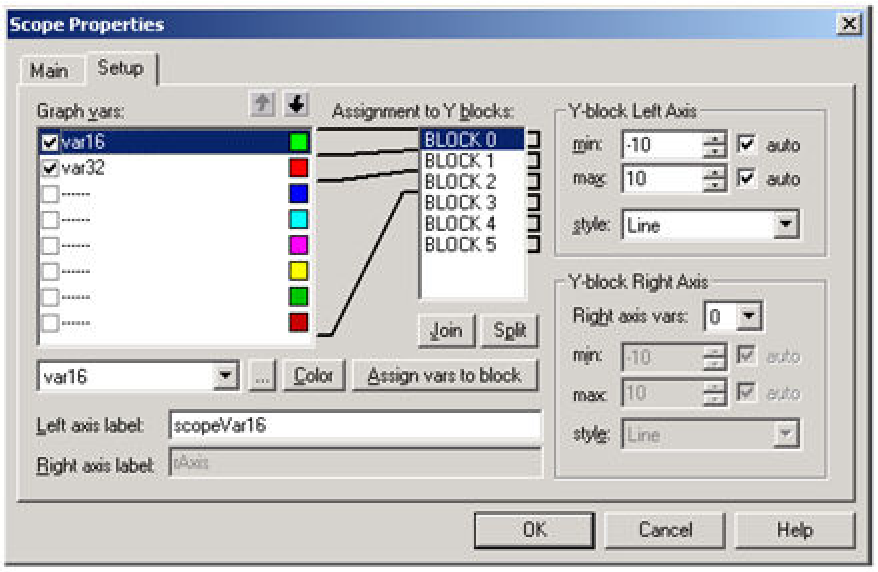


Figure 6. Scope Properties window—Main page

- *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)* is displayed in the chart.
- *X-Y graph* = Inter-variable dependencies (*value* versus *value*) are 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 the 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 are 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 (shown in the following figure) is used to assign variables to be displayed in the oscilloscope chart. Up to eight *Chart vars* (seven in the *X-Y* mode) may be selected for display in the chart and assigned to a maximum of five Y-blocks. Select one of the 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.



Y-block Left Axis: min: -10 [✓ auto]; max: 10 [✓ auto]; style: Line

Y-block Right Axis: Right axis vars: 0; min: -10 [✓ auto]; max: 10 [✓ auto]; style: Line

var16

Left axis label: scopeVar16

Right axis label: rAxis

Figure 7. Scope Properties window—Setup page

The Y block is a graph element represented by one left Y-axis and, optionally, one right Y-axis. 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 the above figure.
- In the *Y-block Left axis* frame, set the axis range by specifying the *min* and *max* axis value, or check the *auto* box to enable automatic minimum and/or maximum tracking. Select the *Style* of drawing the data subsets from the drop-down list box.
- Type the *Left axis label*, which is assigned to a selected Y-block.

The resulting oscilloscope chart is displayed in the following figure.

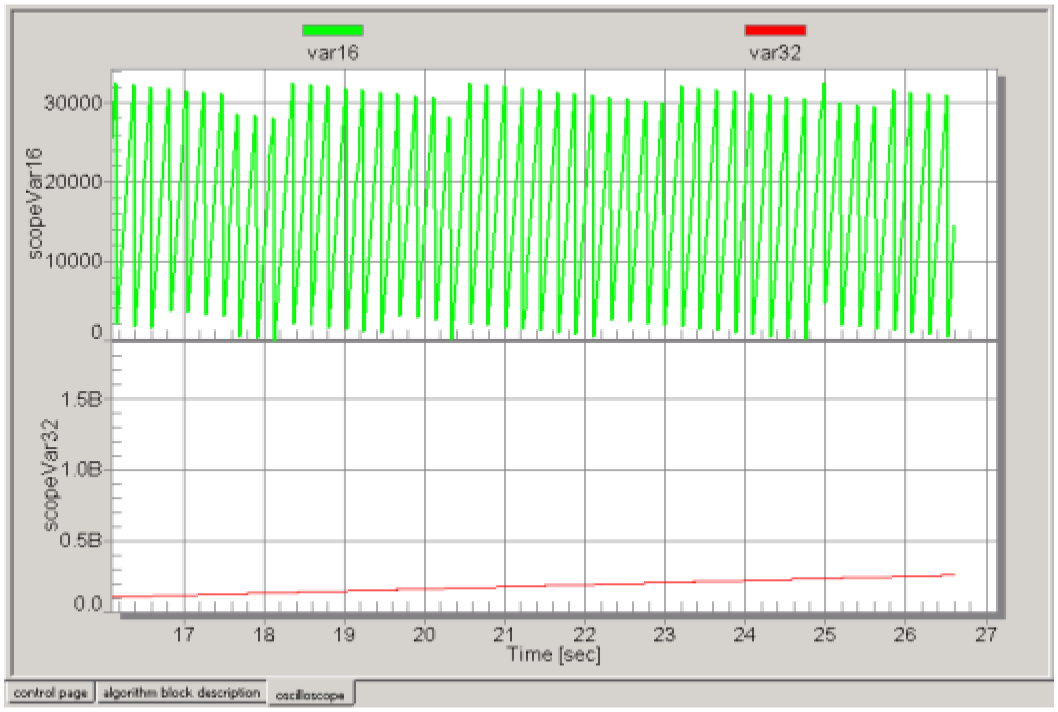
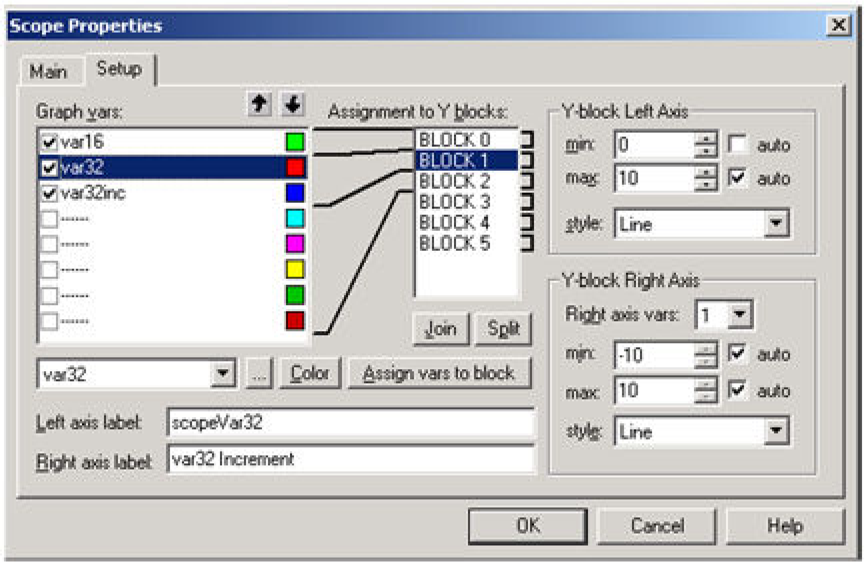


Figure 8. Basic oscilloscope chart

- When more than one variable is 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 uses 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 the left axis, specify the *min*, *max*,, *style*, and *axislabel* for the right axis in the selected Y-block.

In the following figure, one variable is added to the second Y-block and the *Right axis vars* is set to 1. This means that the added variable (var32inc) is assigned to the right axis labeled *var32 Increment*.



Graph vars: ✓ var16 🟩, ✓ var32 🟥, ✓ var32inc 🟦

Selected variable: var32

Left axis label: scopeVar32

Right axis label: var32 Increment

Y-block Left Axis: min: 0 [ ] auto; max: 10 [✓] auto; style: Line

Y-block Right Axis: Right axis vars: 1; min: -10 [✓] auto; max: 10 [✓] auto; style: Line

Figure 9. Third variable added

The *var32inc* variable value is changed manually from 100 to 1000 to achieve the waveform shown in the following figure.

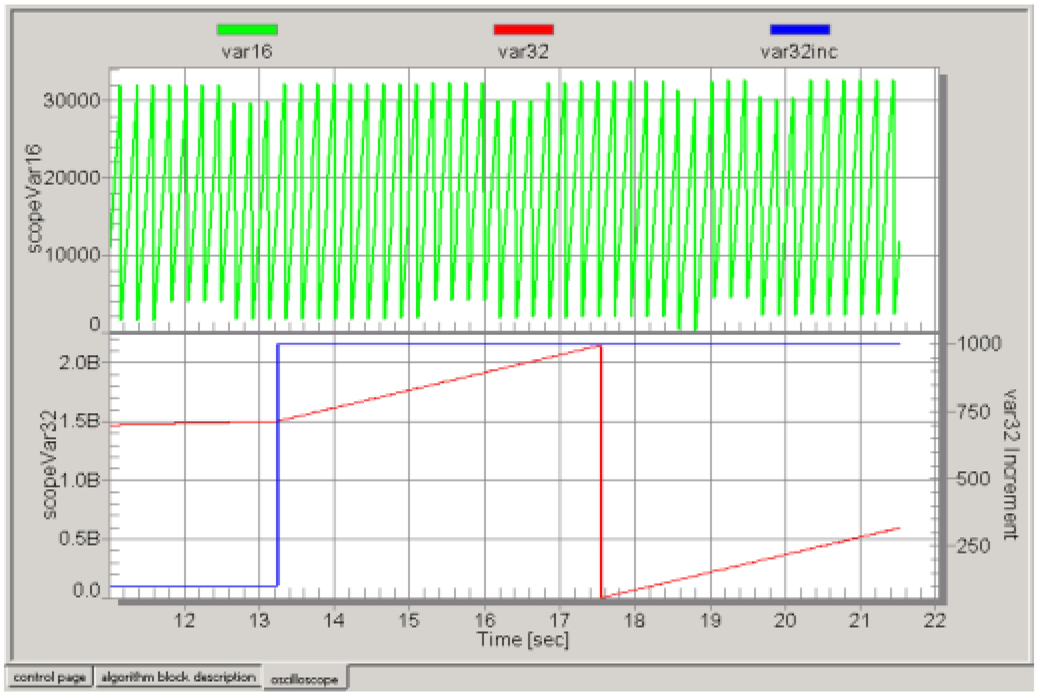
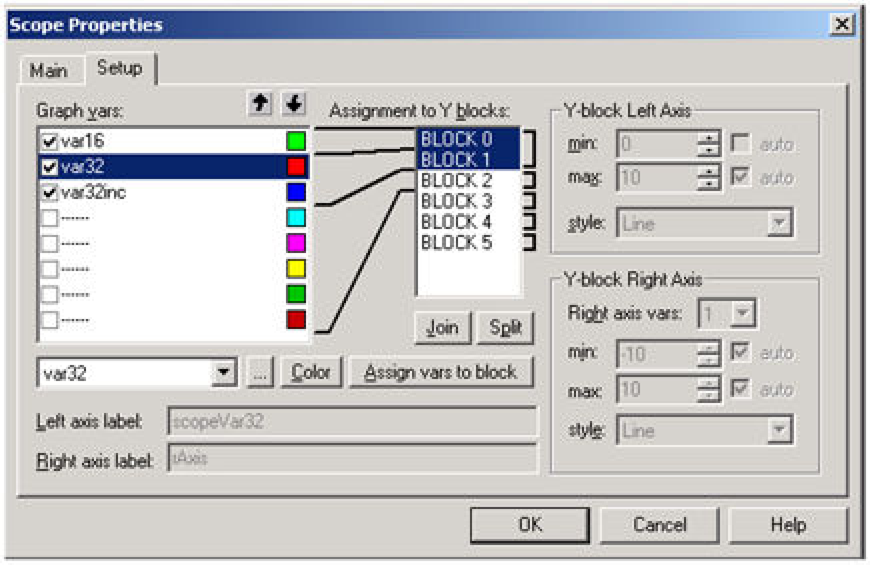


Figure 10. Third variable added—the result

The following figure shows the two Y-blocks joined using the “Join” button. Both Y-blocks are plotted “overlapped”, causing multiple X-axes to be drawn (two in our case).


Scope Properties

Main | Setup

Graph vars:

- ✓ var16 🟩
- ✓ var32 🟥
- ✓ var32inc 🟦

Assignment to Y blocks: BLOCK 0, BLOCK 1, BLOCK 2, BLOCK 3, BLOCK 4, BLOCK 5

Join | Split

var32 | Color | Assign vars to block

Left axis label: scopeVar32

Right axis label: rAxis

Y-block Left Axis — min: 0, auto ☐; max: 10, auto ✓; style: Line

Y-block Right Axis — Right axis vars: 1; min: -10, auto ✓; max: 10, auto ✓; style: Line

OK | Cancel | Help

Figure 11. Joining two Y-blocks

The resulting chart is shown in the following figure.

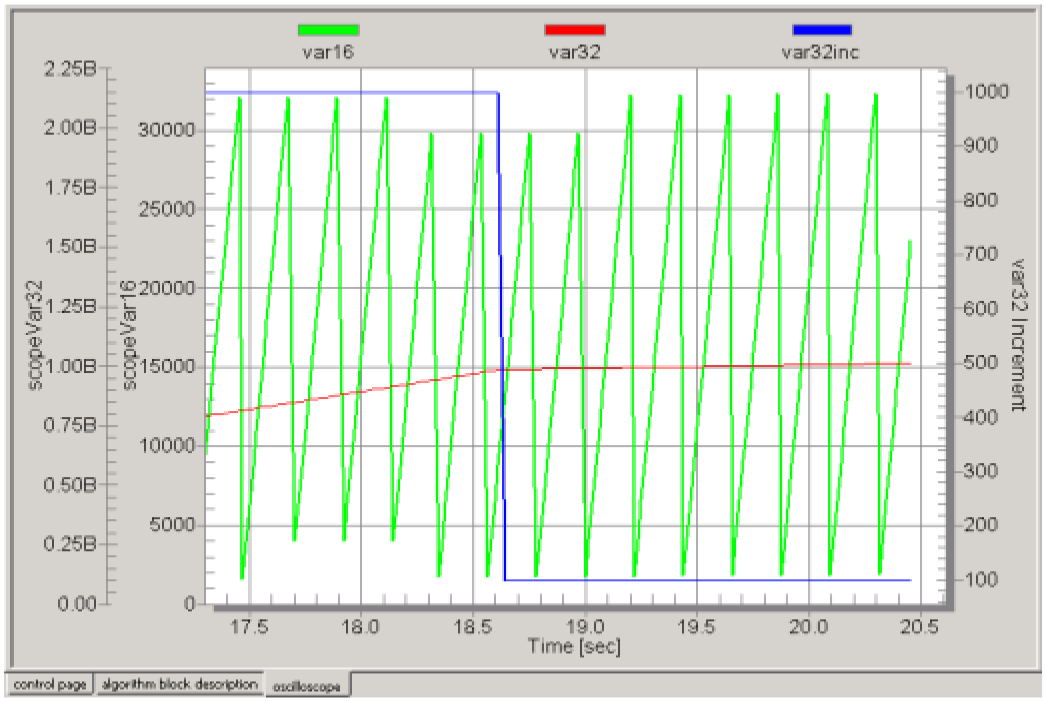
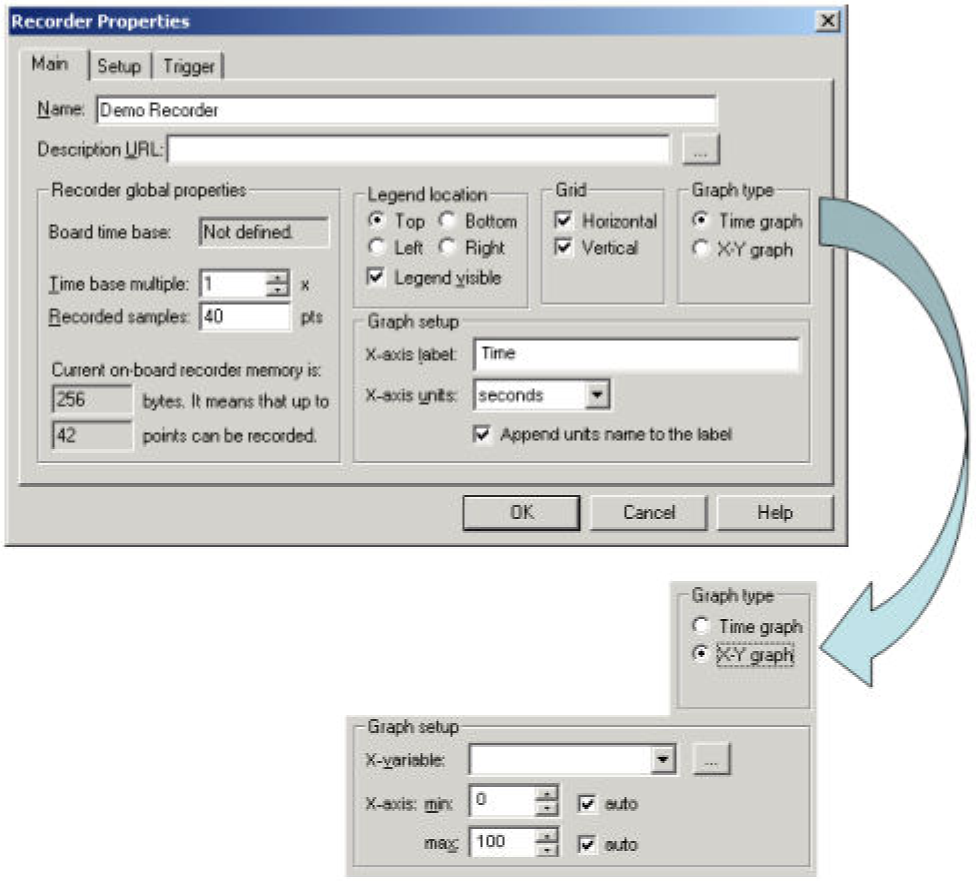


Figure 12. Joining two Y-blocks—the result

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 the use of a much shorter sampling period and enables sampling and plotting of very fast actions.



Recorder Properties

Main | Setup | Trigger

Name: Demo Recorder

Description URL:

Recorder global properties

Board time base: Not defined.

Time base multiple: 1 x

Recorded samples: 40 pts

Current on-board recorder memory is:

256 bytes. It means that up to

42 points can be recorded.

Legend location

[✓ Top] Bottom Left Right

[✓ Legend visible]

Grid

[✓ Horizontal]

[✓ Vertical]

Graph type

[✓ Time graph] X-Y graph

Graph setup

X-axis label: Time

X-axis units: seconds

[✓ Append units name to the label]

OK | Cancel | Help

Graph type

Time graph [✓ X-Y graph]

Graph setup

X-variable:

X-axis: min: 0 [✓ auto]

max: 100 [✓ auto]

Figure 13. Recorder Properties window—Main page

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

- *Name* = The name of the Recorder item that is 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 the 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 the 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 the Recorder operation.

— Time graph = A variable (*values* versus *time*) is displayed in the chart.

— X-Y graph = inter-variable dependencies (*value* versus *value*) are displayed in the chart.

- Graph setup (for the Time graph):

  — *X-axis label* = Specify the name displayed for the X-axis.

  — *X-axis units* = Selects the axis units.

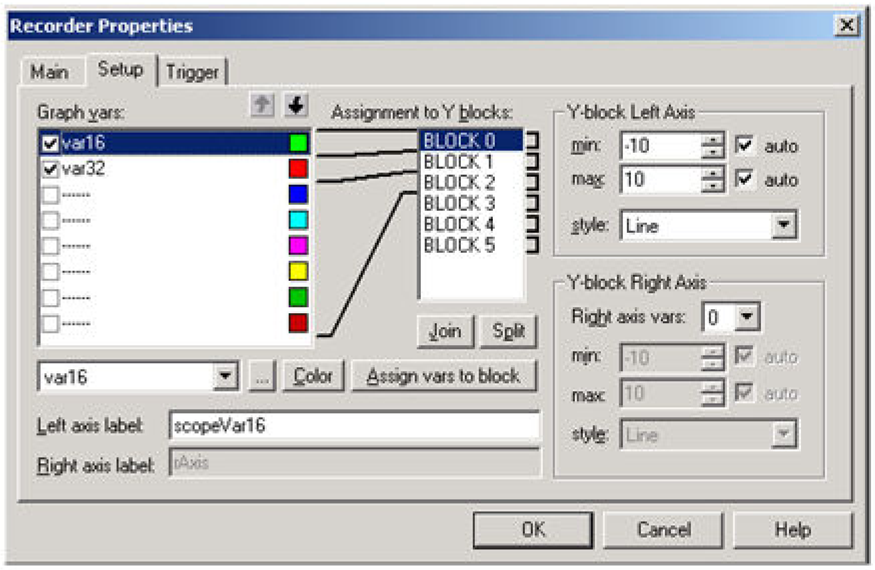
- Graph setup (for the X-Y graph):

  — *X-variable* = Selects the variable whose values are used for the X-axis values.

  — *X-axis min* = Sets the lower limit value of the X-axis (check the auto box to enable the auto-scaling of the X-axis).

  — *X-axis max* = Sets the upper limit value of the X-axis (check the auto box to enable the auto-scaling of the X-axis).

The *Setup* page of the *Recorder properties* dialog, shown in the following figure, 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 Scope on page 13.



Graph vars: [✓] var16, [✓] var32

Y-block Left Axis: min: -10 [✓ auto]; max: 10 [✓ auto]; style: Line

Y-block Right Axis: Right axis vars: 0; min: -10 [✓ auto]; max: 10 [✓ auto]; style: Line

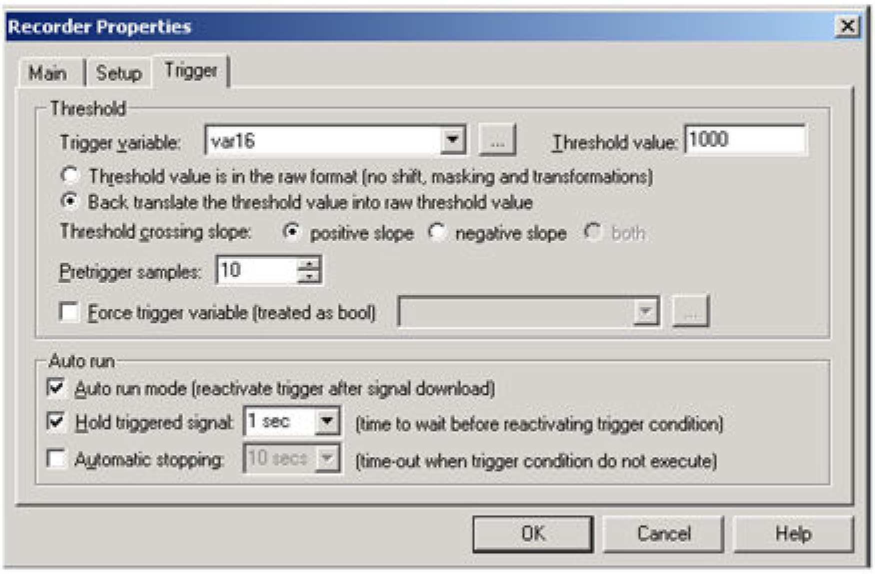
var16

Left axis label: scopeVar16

Right axis label: rAxis

Figure 14. Recorder Properties window—Setup page

The *Trigger* page, shown in the following figure, 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).



Threshold

Trigger variable: var16 Threshold value: 1000

( ) Threshold value is in the raw format (no shift, masking and transformations)

(✓) Back translate the threshold value into raw threshold value

Threshold crossing slope: [✓ positive slope] negative slope both

Pretrigger samples: 10

☐ Force trigger variable (treated as bool)

Auto run

☑ Auto run mode (reactivate trigger after signal download)

☑ Hold triggered signal: 1 sec (time to wait before reactivating trigger condition)

☐ Automatic stopping: 10 secs (time-out when trigger condition do not execute)

Figure 15. Recorder Properties window—Trigger page

- *Threshold* = Specify the 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. When crossed, it causes the trigger event. Specify a threshold value and select whether the value is in the raw format (as in the board application) or whether it must be translated back into the raw format (for example, 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 the 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 before the trigger event.
- *Auto run* = Specifies the conditions for reactivating the trigger for repeated recording.
  - — *Auto run mode* = Select it 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 the actual buffer data is downloaded.

4.1.2 Detail View

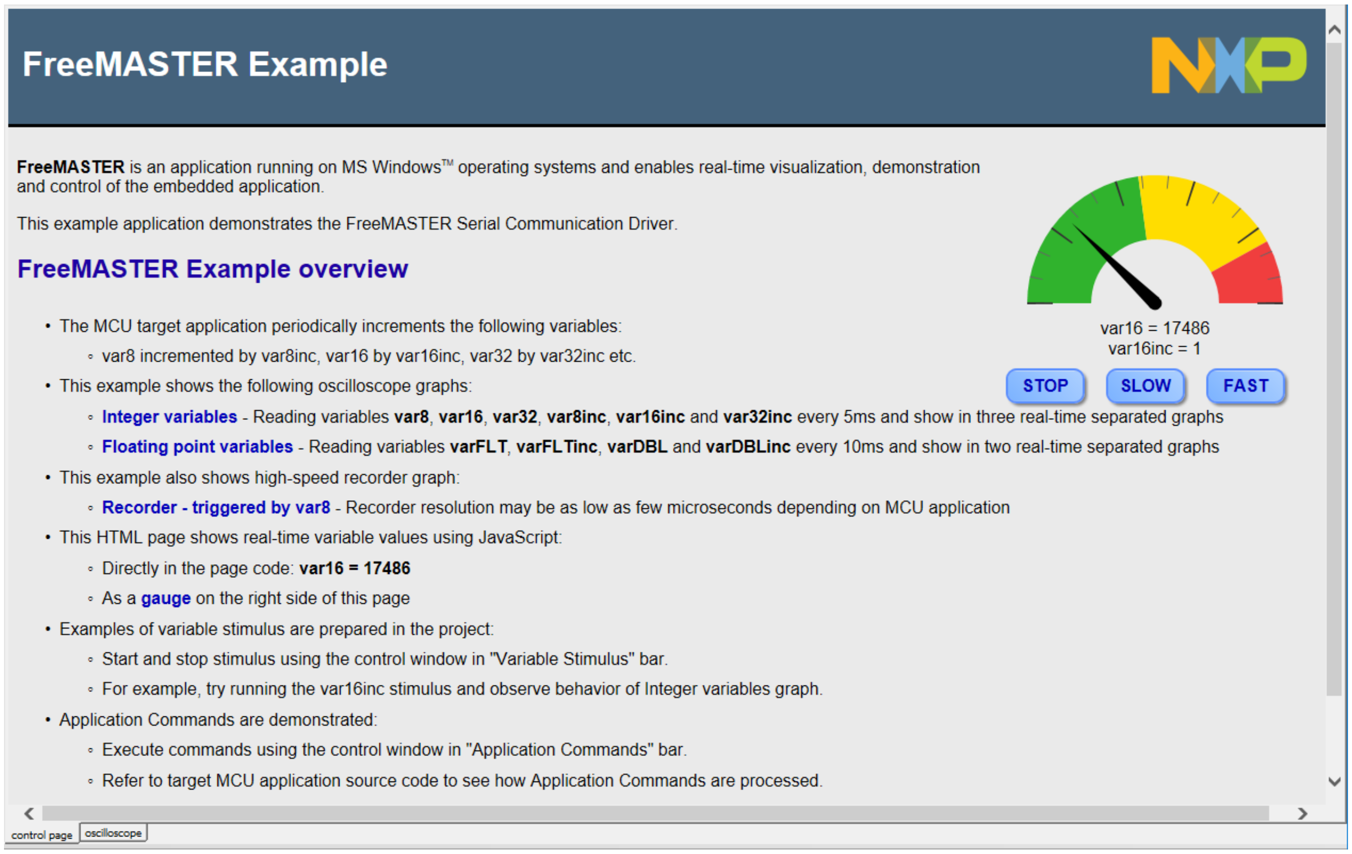
The *Detail View* is a multi-page pane. The availability of various pages in the *Detail View* depends on the type of item selected in the *Project Tree*.

When the control page is defined in the project *Options* (HTML pages on page 45), it is available for all *Project Tree* items to enable you to control the board at any time. The content of the *algorithm block description* page changes with the *Project Tree* item selected. When the Scope or Recorder items are 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 provide a 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 the project *Options* dialog, described in HTML pages on page 45.

The control page for the demo application, used as an example, is shown in the below figure. Despite its name, there are no “control” features utilized in this simple application. Still, it demonstrates a JScript scripting technique to display variables' values directly in the HTML-coded page. For more details about the HTML and scripting, see FreeMASTER ActiveX object on page 47.


FreeMASTER Example

FreeMASTER is an application running on MS Windows™ operating systems and enables real-time visualization, demonstration and control of the embedded application.

This example application demonstrates the FreeMASTER Serial Communication Driver.

FreeMASTER Example overview

- The MCU target application periodically increments the following variables:
  - var8 incremented by var8inc, var16 by var16inc, var32 by var32inc etc.
- This example shows the following oscilloscope graphs:
  - **Integer variables** - Reading variables **var8**, **var16**, **var32**, **var8inc**, **var16inc** and **var32inc** every 5ms and show in three real-time separated graphs
  - **Floating point variables** - Reading variables **varFLT**, **varFLTinc**, **varDBL** and **varDBLinc** every 10ms and show in two real-time separated graphs
- This example also shows high-speed recorder graph:
  - **Recorder - triggered by var8** - Recorder resolution may be as low as few microseconds depending on MCU application
- This HTML page shows real-time variable values using JavaScript:
  - Directly in the page code: **var16 = 17486**
  - As a **gauge** on the right side of this page
- Examples of variable stimulus are prepared in the project:
  - Start and stop stimulus using the control window in "Variable Stimulus" bar.
  - For example, try running the var16inc stimulus and observe behavior of Integer variables graph.
- Application Commands are demonstrated:
  - Execute commands using the control window in "Application Commands" bar.
  - Refer to target MCU application source code to see how Application Commands are processed.

var16 = 17486
var16inc = 1

Figure 16. Demo application Control Page

A few more sophisticated Control Page screen shots are shown below. The applications shown here use some third-party instrumentation components, inserted into the HTML code as embedded ActiveX objects. More screen shots are available on the FreeMASTER home page on www.nxp.com (see Where to find the latest version on page 4 for more details).

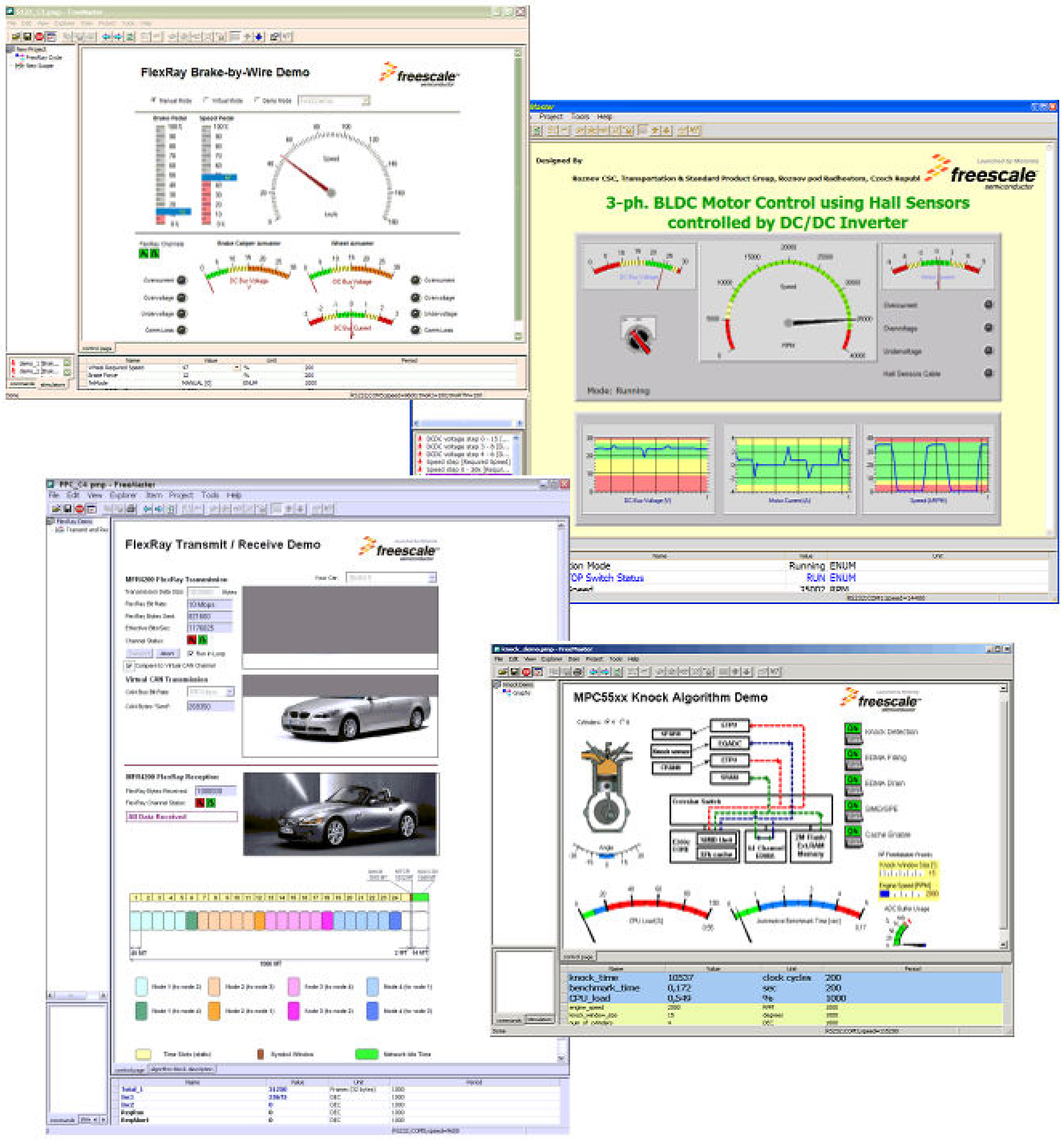

FlexRay Brake-by-Wire Demo
freescale
3-ph. BLDC Motor Control using Hall Sensors controlled by DC/DC Inverter
FlexRay Transmit / Receive Demo
MPC55xx Knock Algorithm Demo

Figure 17. HTML control page examples

4.1.2.2 Algorithm Block Description

The *Algorithm Block Description* page in the *Detail View* pane is used for placing an HTML page that describes the selected block functionality. The URL or local path to the source file is specified in the block item properties dialog, described in Project block and sub-block on page 11. This page is displayed when it is defined and when you select 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 and other controls, but it is not a common practice.

4.1.2.3 Current item help

The *Current item help* tab is designated to contain an HTML page describing the selected *Scope* or *Recorder.* This page should contain such information as definitions or use instructions. It 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 shown in Scope on page 13. Similarly, the *Recorder page* contains the chart created from the recorded data, as described in Recorder on page 18.

4.1.3 Watch-Grid

The *Watch-Grid* pane at 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.

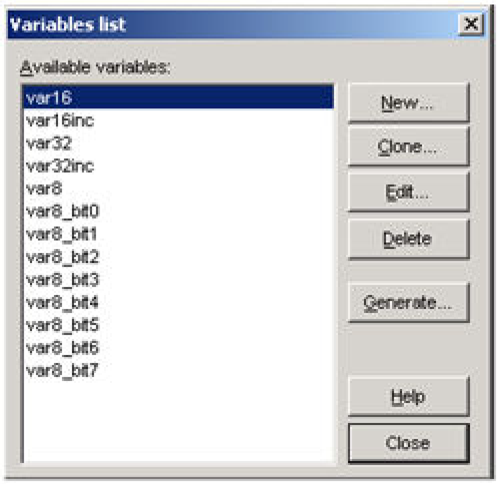
During the definition of a variable, the variable name, units, and number format are specified. Moreover, the *Watch bar* can be used to change the graphical look of the variable, including font type and size, foreground and background color, and alignment. See Watch Bar on page 38 for details.

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. For details about variables, see Variables on page 24.

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 the following figure, can be opened by selecting the *Variables* item from the *Project* menu. It can also be opened from other project-development points, where it can be used to manage variables (for example, *Scope Setup*).



var16
var16inc
var32
var32inc
var8
var8_bit0
var8_bit1
var8_bit2
var8_bit3
var8_bit4
var8_bit5
var8_bit6
var8_bit7

Figure 18. Variables list dialog box

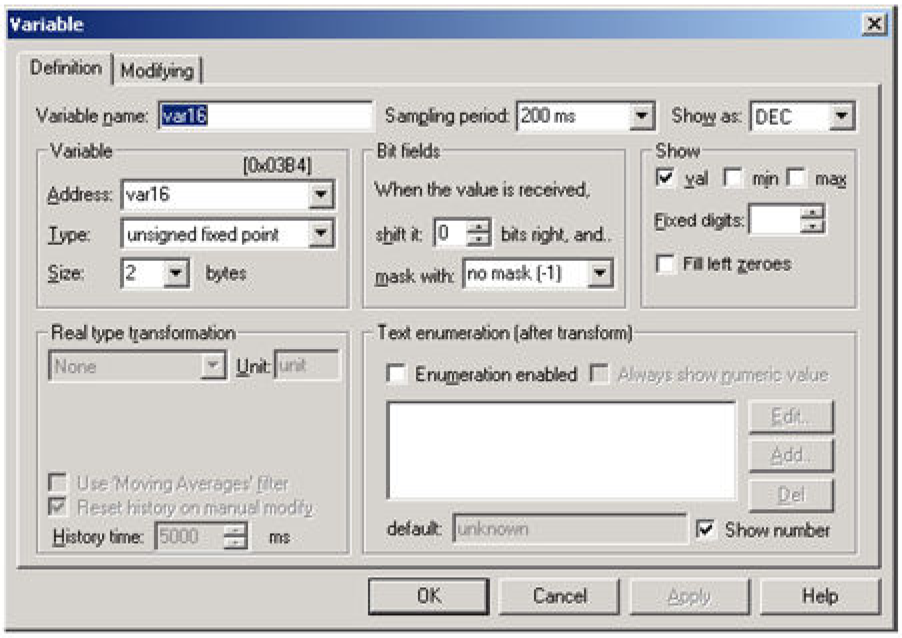
To define a new variable, use the *New* button. This opens the *Variable* dialog box, where you can set the variable properties. When you want to create a copy of an existing variable, select the original variable and press the *Clone* button.

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

The *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 Generating variables on page 28. The loading of symbol files is described in Symbol files on page 40.

Variable Settings

The *Variable* definition dialog has two tabs: *Definition,* shown in Figure 19. on page 25, and *Modifying*, shown in Figure 20. on page 27.



Variable name: var16
Sampling period: 200 ms
Show as: DEC
[0x03B4]
Address: var16
Type: unsigned fixed point
Size: 2 bytes
When the value is received,
shift it: 0 bits right, and..
mask with: no mask (-1)
[✓ val] min max
Fixed digits:
Fill left zeroes
None Unit: unit
Use 'Moving Averages' filter
[✓ Reset history on manual modify]
History time: 5000 ms
Enumeration enabled
Always show numeric value
default: unknown
[✓ Show number]

Figure 19. Variable dialog box—Definition tab

In the *Definition* tab, specify the 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 a standard ELF/ Dwarf1 format) or from the text-based MAP file generated by the linker. See Symbol files on page 40 for more information about loading the symbol tables from the 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 the target application.
- *Bit fields* = The parameters for extracting a single bit or bit groups from a given variable.

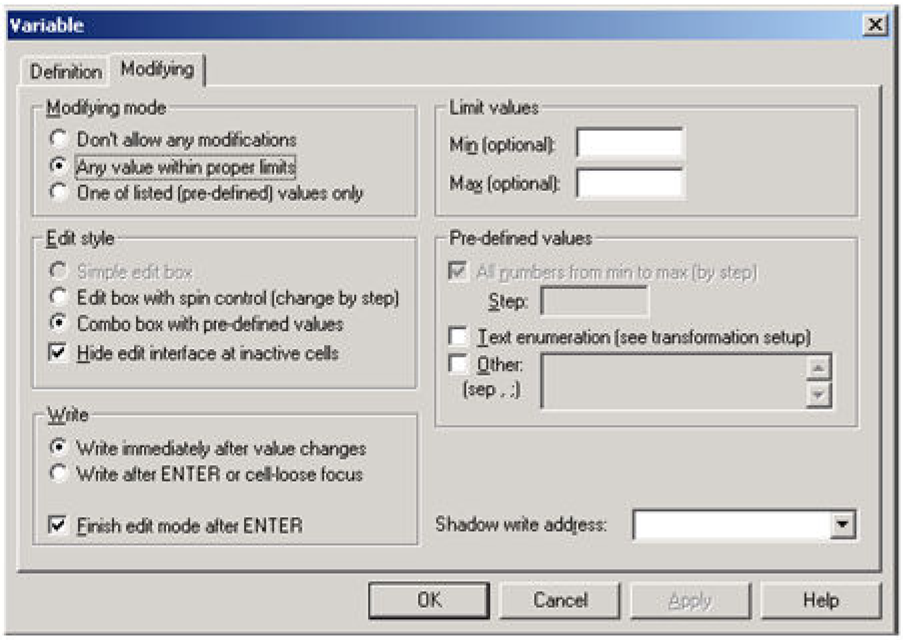
  - *Shift* = Specify the number of bits by which the received value is right-shifted before it is masked.
  - *Mask* = Select or specify the mask value which is *AND*-ed with the shifted value.
  - Using the Shift and Mask fields, you can extract any bit field from the received variable. For example, 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 these 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 a 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 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 may 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 the *Edit* , *Add*, and *Del* buttons to manage the look-up table with the 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 the following figure, contains the settings and restrictions for variable value modifications.



Variable

Definition | Modifying

Modifying mode
- ( ) Don't allow any modifications
- [✓ Any value within proper limits]
- ( ) One of listed (pre-defined) values only

Limit values
- Min (optional):
- Max (optional):

Edit style
- ( ) Simple edit box
- ( ) Edit box with spin control (change by step)
- [✓ Combo box with pre-defined values]
- [✓ Hide edit interface at inactive cells]

Pre-defined values
- [✓ All numbers from min to max (by step)]
- Step:
- [ ] Text enumeration (see transformation setup)
- [ ] Other: (sep , ;)

Write
- [✓ Write immediately after value changes]
- ( ) Write after ENTER or cell-loose focus
- [✓ Finish edit mode after ENTER]

Shadow write address:

Figure 20. Variable 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 you enter into the watch window is then validated with the specified limits.
    - — *One of listed values only* = When you specified a list of values, only those values are 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*" are treated as predefined values.

  Text enumeration = Treat all values from the text enumeration look-up table as predefined.

  Other = Specify any other predefined values (separated by a comma or a semicolon).
- *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 is displayed in the form of an edit box with two spin arrows to increment and decrement the value.
    - — *Combo box with pre-defined values* = The variable value edit interface is displayed in the form of a drop-down list box. The predefined values are available in the list.
    - — *Hide edit interface at inactive cells* = The variable edit interface is hidden when the appropriate cell in the watch grid looses the 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 is sent to the embedded application each time you press the spin arrow button or select a new value in the drop-down list box.
    - — *Write after ENTER or kill focus only* **=** The modified variable value is not sent to the embedded application until you press the "Enter" key.

4.2.1 Generating variables

The *Generate* button in the variable list dialog (Figure 18. on page 24) opens the *Generate variables* dialog box shown in the following figure.

In this dialog, you can automatically generate the variable objects for the symbols loaded from an embedded application executable file (ELF) or a linker MAP file (see Symbol files on page 40 for more information about symbol tables).

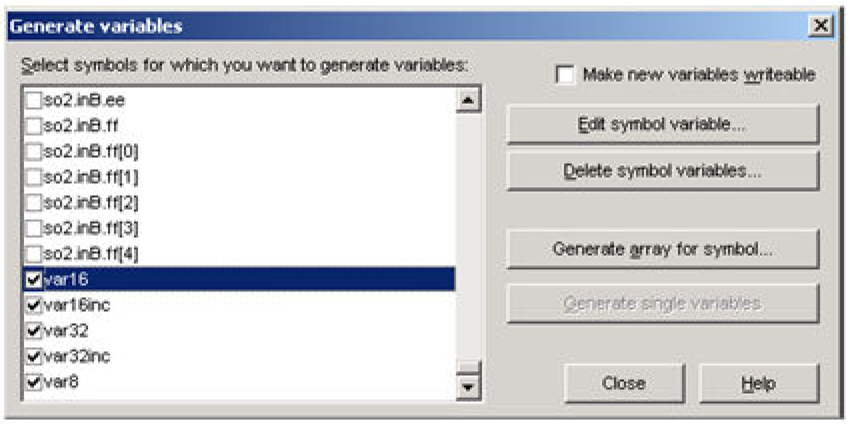


Figure 21. Generating variables

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 this button 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 click 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 the following figure. Use the *New* , *Clone* , *Edit*, and *Delete* buttons to manage the list. It is very similar to the variables' management:

- *New* button = Creates a new application command.
- *Edit* button = Edits the 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.

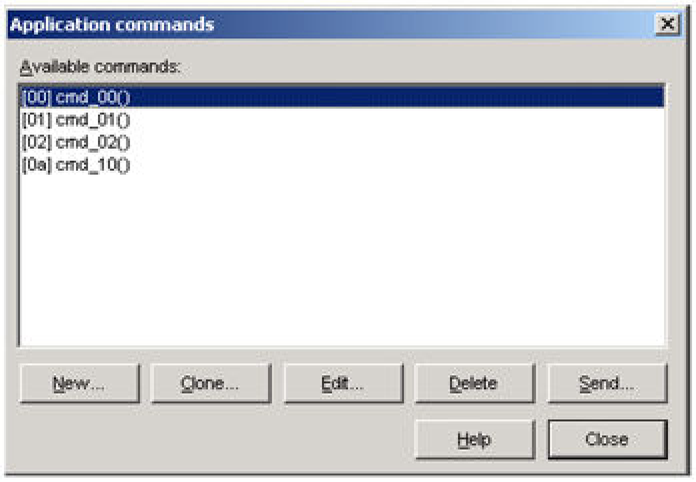
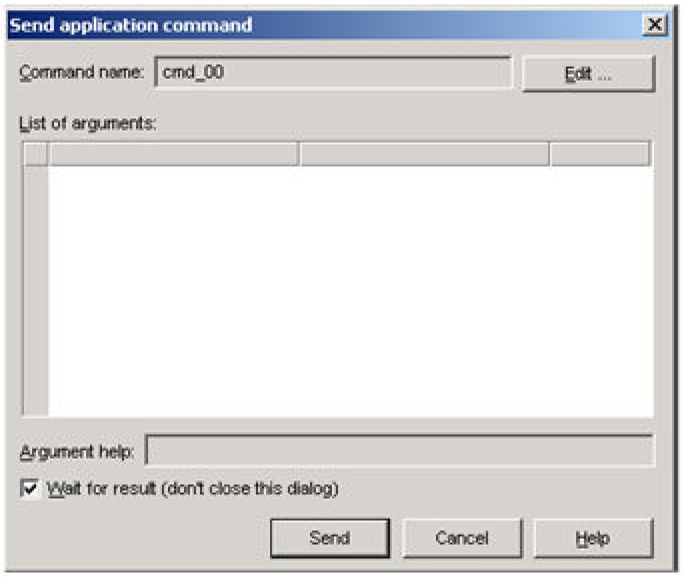


Figure 22. Project application commands

In the *Sendapplication 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 23. on page 29. For each argument, you can define the help message, which is displayed in this dialog when typing the argument value, as shown in Figure 24. on page 30.



Command name: cmd_00

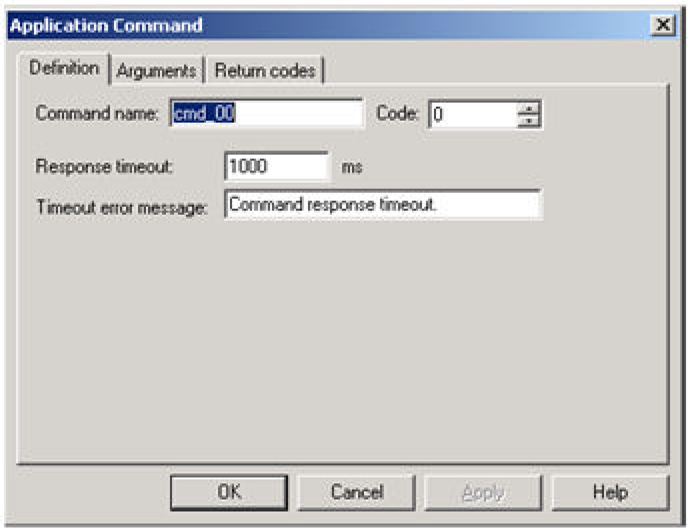
Argument help:

☑ Wait for result (don't close this dialog)

Figure 23. Sending application command

If you want 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 opens. The first of three pages of the dialog are shown in the following figure.

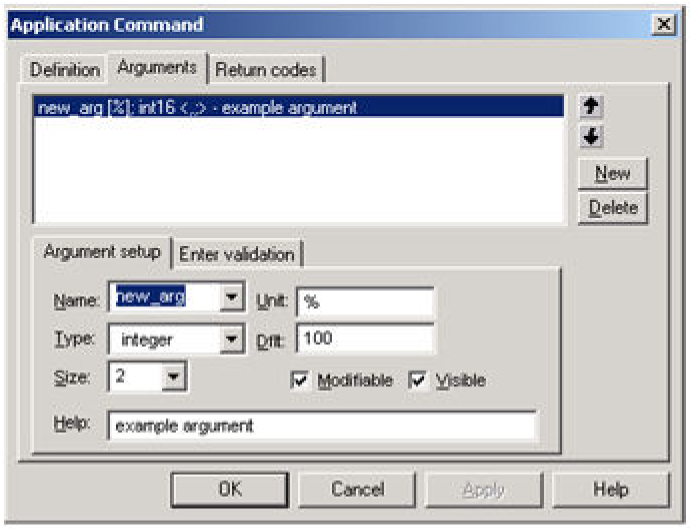


Command name: cmd_00
Code: 0
Response timeout: 1000 ms
Timeout error message: Command response timeout.

Figure 24. Application Command window—Definition tab

In 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 the alert window.



new_arg [%]: int16 <,,> - example argument

Name: new_arg
Unit: %
Type: integer
Dflt: 100
Size: 2
Modifiable: ✓
Visible: ✓
Help: example argument

Figure 25. Application Command window—Arguments tab (page 1)

The *Arguments* tab, shown in the above figure, is used for the definition of command arguments. The commands that do not have any arguments have an empty argument list. The 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, define the selected argument parameters:

- *Name* = Specify the argument name as it shall appear in the list and in the *Send application command* dialog when you are prompted for argument values. You can also select the existing argument name from the drop-down list box.

- *Type* = Specify the argument numeric value (*integer* or *floating point*).
- *Size* = Specify the argument value size (in bytes).
- *Unit* = Specify any text to be displayed as argument units. This text is not sent to the target application.
- *Dflt* = Enter the default value of the argument. This value is set in the argument list of the *Send application command* dialog. If it is empty, type the value every time you send the command.
- *Modifiable* = Unless this box is checked, you are not allowed to change the default argument value in the argument list of the *Send application command* dialog.
- *Visible* = If this box is not checked, the argument is not displayed in the argument list and its default value is always sent to the target application.
- *Help* = Write any text information to be shown in the *Send application command* dialog when you are prompted for an argument value.

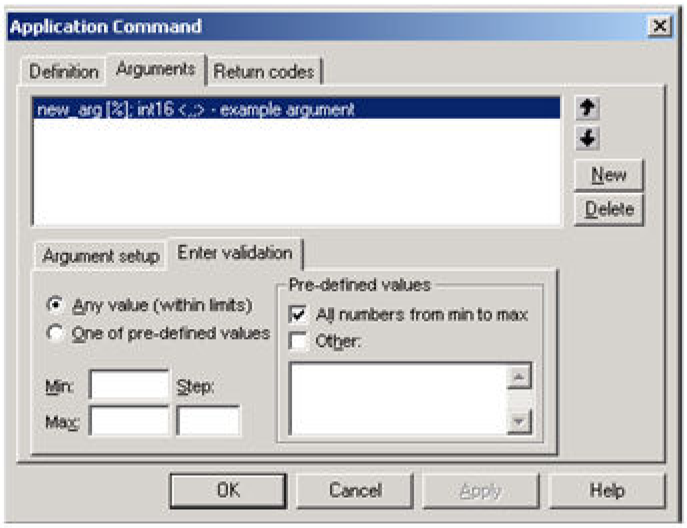
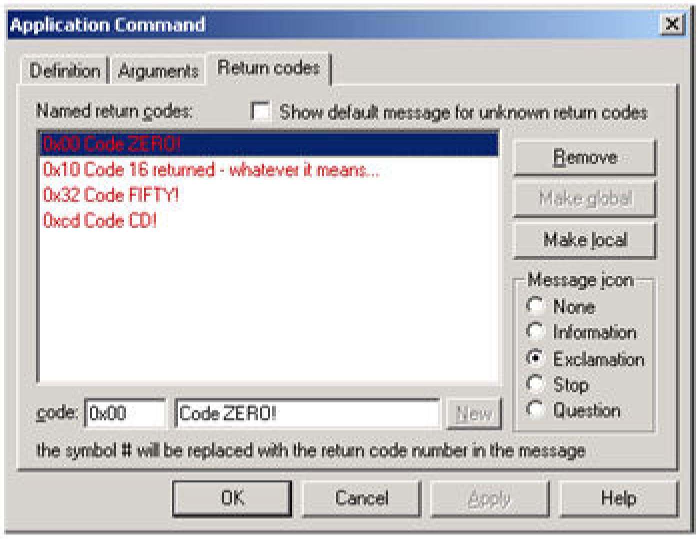


Figure 26. Application Command window—Arguments tab (page 2)

In the *Enter validation* sub-page shown in the above figure, 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 the *Min* and *Max* limits, if they 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 the *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 (separated by a comma) which are valid as the argument value.

The *Return codes* page, shown in the following figure, is used to specify the command return code messages. To create a return code, enter the return code value in a hexadecimal (0x00) or decimal form in the *code* field at the lower left-hand side of the page, enter the return code message in the next field, and click the *New* button. The return code item appears in the list. Repeat to create all desired return codes. The *Message icon* may be assigned to each return code message from the panel at the lower right-hand side of the page. It then appears in the message dialog, together with the text of the message.



0x00 Code ZERO!
0x10 Code 16 returned - whatever it means...
0x32 Code FIFTY!
0xcd Code CD!

code: 0x00 Code ZERO!

the symbol # will be replaced with the return code number in the message

Figure 27. Application Command window—Return codes tab

Return codes can be local or global. The local return codes apply to a single command, while the global commands' return codes are valid for all commands of the project. To switch between the 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 the 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 the 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 28. on page 33. After specifying the name of the original project file, 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, 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 imported one.
- *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 case, the name is 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 is imported and inserted as a standard block item.

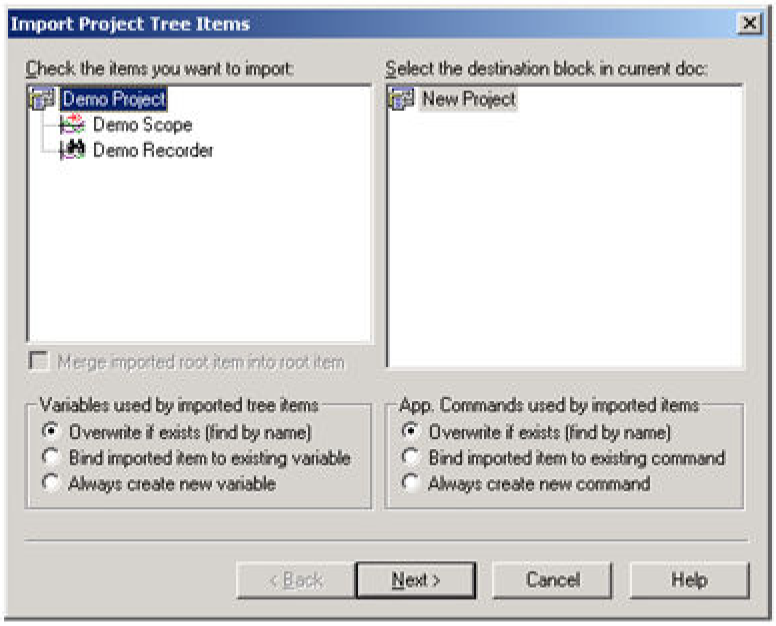

Import Project Tree Items
Check the items you want to import:
Demo Project
Demo Scope
Demo Recorder
Select the destination block in current doc:
New Project
Merge imported root item into root item
Variables used by imported tree items
Overwrite if exists (find by name)
Bind imported item to existing variable
Always create new variable
App. Commands used by imported items
Overwrite if exists (find by name)
Bind imported item to existing command
Always create new command
< Back
Next >
Cancel
Help

Figure 28. Import Project Tree Items window

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 29. on page 34. The three check-box lists contain the objects found in the source project file:

- *Variables* = Put a check mark next to each variable you want to import. You don't need to import variables that are referenced from the tree items selected in the previous dialog from Figure 28. on page 33; such variables are always unconditionally imported.
- *App. commands* = Put a check mark next to each application command definitions you want to import. As with variable objects, you do not have to check commands that are referenced from the block tree items selected in the 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 next to each variable stimulator you want to import.

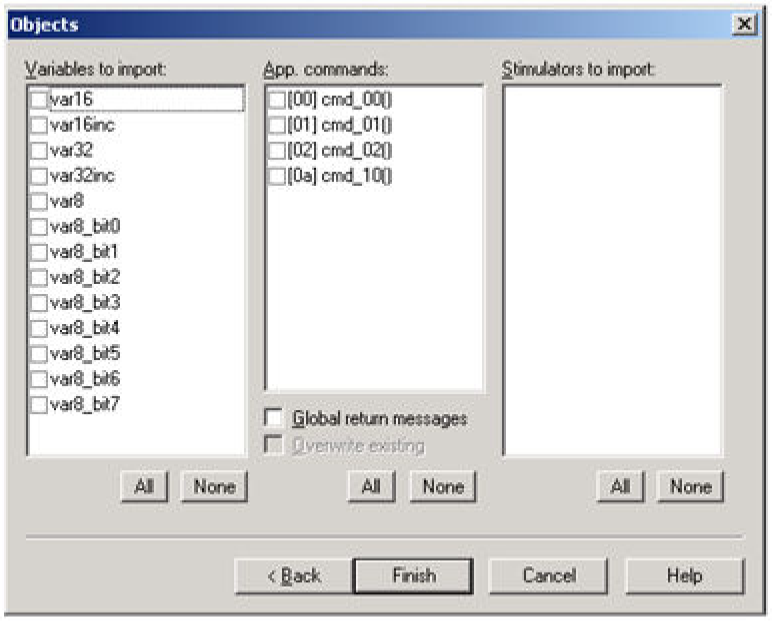


Figure 29. Import project objects

4.5 Menu description

4.5.1 File menu

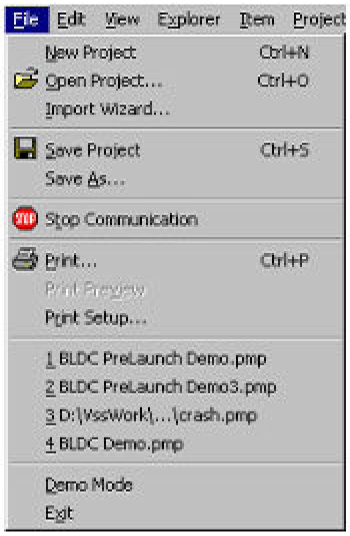


Figure 30. File menu

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.

SaveProject saves the open project into the current file, while *Save As* enables you 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, you can choose to reload the new version of the file.

Print prints the content of a 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.

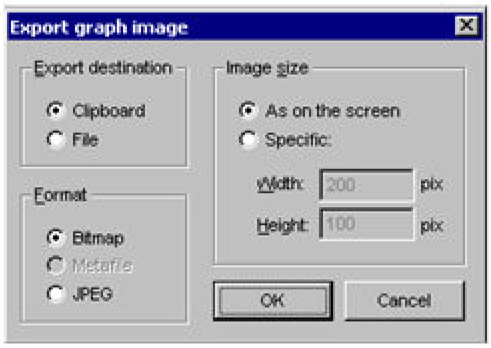
Demo Mode is a switch to enter or leave the application demonstration mode. While in the *Demo Mode*, you cannot modify any important project settings.

Exit exits the application.

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 setup dialog is shown in the following figure.


Export graph image

Export destination
- [✓ Clipboard]
- File

Format
- [✓ Bitmap]
- Metafile
- JPEG

Image size
- [✓ As on the screen]
- Specific:

Width: 200 pix
Height: 100 pix

OK | Cancel

Figure 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.***

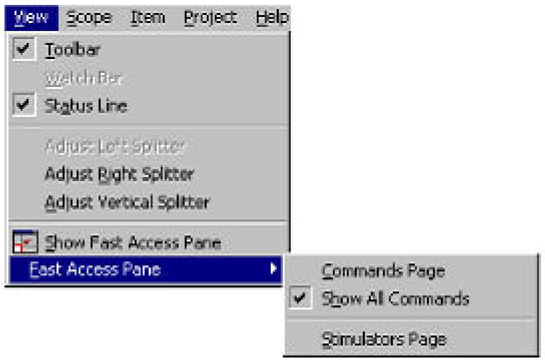


Figure 32. View menu

Another feature of the *View* menu is the ability to adjust the size of individual panes without using a mouse. Use *Adjust Left Splitter* to change the height of the *Fast Access* pane in the *Tree* pane. *Adjust Right Splitter* can be used to change 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-hand part of the application window. It contains two pages: the *Commands Page*, which lists the 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 5. on page 13 and Project block and sub-block on page 11 for more information.

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.

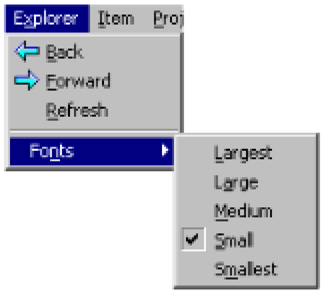


Figure 33. Explorer menu

The *Back*, *Forward*, and *Refresh* items represent the 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.

4.5.5 Scope menu

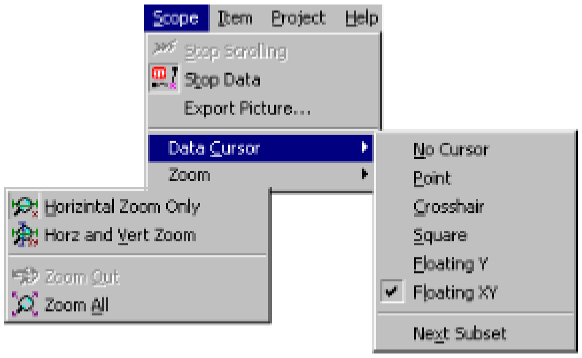


Figure 34. Scope menu

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* sub-menu consists of zooming modes and commands. The *Horizontal Zoom Only* mode allows "x-axis only" zooming. *Horz and Vert Zoom* allows free rectangle zooming.

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 right-click 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

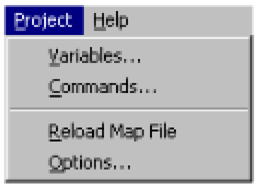


Figure 35. Project menu

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 the board application variables from the *.map* file.

Options opens the multi-page dialog box for all project option settings.

4.6 Toolbars

4.6.1 Toolbar

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

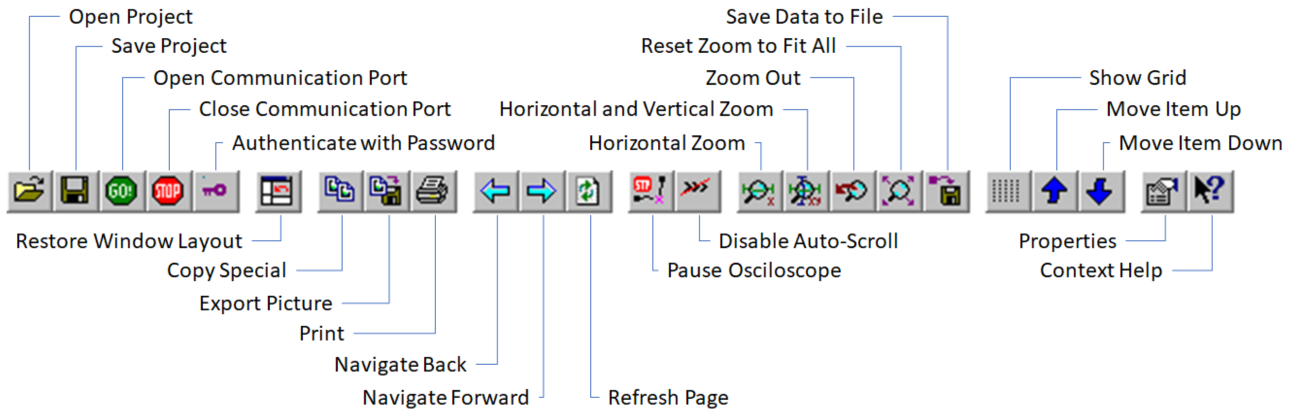


Figure 36. Toolbar

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 the variables.

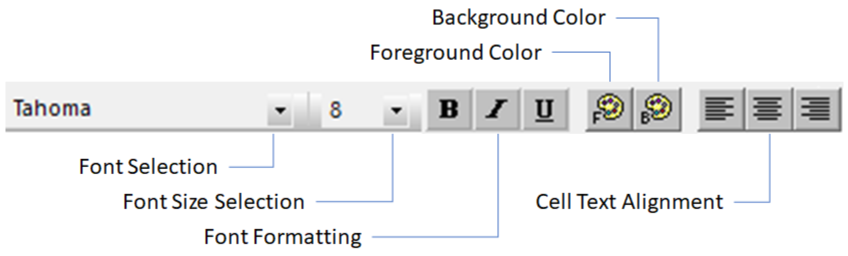


Figure 37. Watch Bar

Chapter 5 Project options

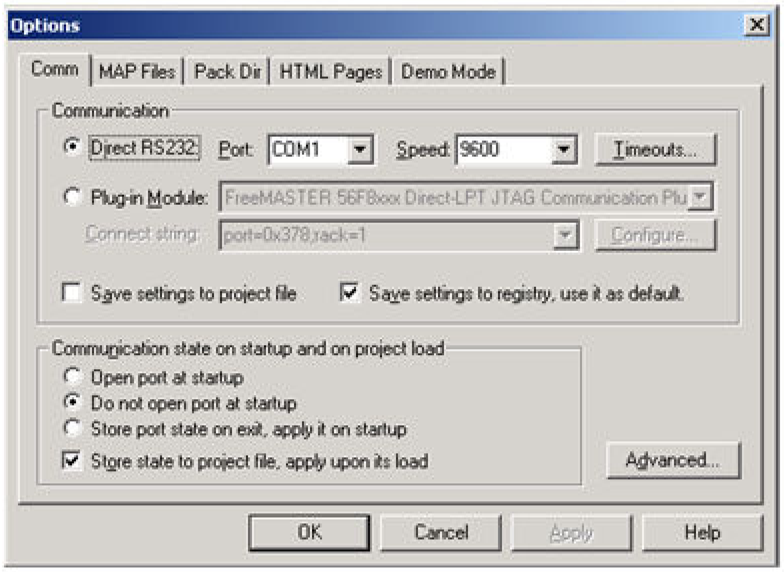
To set application and project options, use the *Options* dialog, which can be activated by selecting *Options* from the *Project* menu. The dialog consists of several pages, each dedicated to a different group of settings.

5.1 Communication

To set up the parameters related to the communication between the FreeMASTER application and the target board, select the first tab, as shown in Figure 38. on page 39.

- Communication
  - — Direct RS232 connection = The standard serial interface (3-wire RS232 cable) is 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 is 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 the 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 a string form, internally called a “connect string”. You can directly specify this string, or press the *Build* button to open the module-specific configuration dialog.
  - — Save settings to project file = If checked, the communication parameters are stored within the project file during the next Save Project operation. When the project is loaded the next time, 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 are used as default when the application is started the next time.



Communication

[✓ Direct RS232] Port: COM1 Speed: 9600

Plug-in Module: FreeMASTER 56F8xxx Direct-LPT JTAG Communication Plu

Connect string: port=0x378;rack=1

☐ Save settings to project file ☑ Save settings to registry, use it as default

Communication state on startup and on project load

Open port at startup

[✓ Do not open port at startup]

Store port state on exit, apply it on startup

☑ Store state to project file, apply upon its load

Figure 38. Communication options

NOTE

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

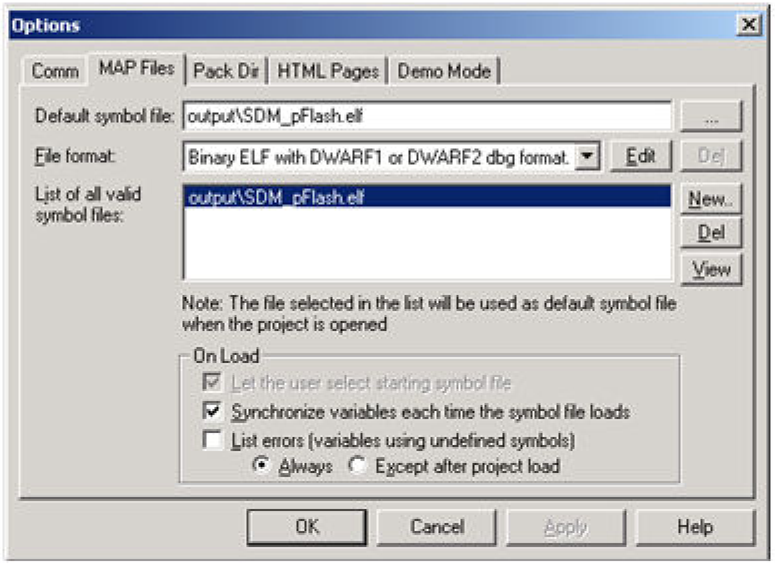
- Communication state = By selecting one of three options, you can set whether the communication port is opened when the application is started or not.
  - — Open port at startup.
  - — Do not open port at startup.
  - — Store port status on exit, apply it on start-up.
  - — Store status to a project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

5.2 Symbol files

When defining the project variables, it is often useful to specify the physical address of the target memory location as the symbol name instead of the 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 the code running from the RAM memory and the other one for the code running from the flash.

The following figure shows the *MAP Files* tab in the project *Options* dialog.



Default symbol file: output\SDM_pFlash.elf

File format: Binary ELF with DWARF1 or DWARF2 dbg format.

List of all valid symbol files: output\SDM_pFlash.elf

Note: The file selected in the list will be used as default symbol file when the project is opened

[✓] Let the user select starting symbol file
[✓] Synchronize variables each time the symbol file loads
[ ] List errors (variables using undefined symbols)
(●) Always ( ) Except after project load

Figure 39. Symbol files options

- *Default symbol file* = Specify the name of the symbol file to be loaded by default. 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 the HiWare Smart Linker v5.0.9.
  — *Define new Regular Expression-based parser* = Choose this to define a new text file parser based on the regular expression pattern matching; see Regular expression-based MAP file parser on page 41.
- *List of valid symbol files* = The list of symbol files defined in the project. Use the *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 Packing resource files into project file on page 43 for more information.

- *On Load* panel options control the actions performed with the symbol files after the project is loaded:
  — *Let the user select the symbol file* = When checked, you are 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. You can 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 the ELF output format and the MAP file cannot be parsed by the built-in HiWare 5.0.9 parser, describe the internal MAP file structure using the regular expression pattern.

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

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

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

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

Firstly, describe the line format by the regular expression pattern. The pattern

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

can be read as follows:

- 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+

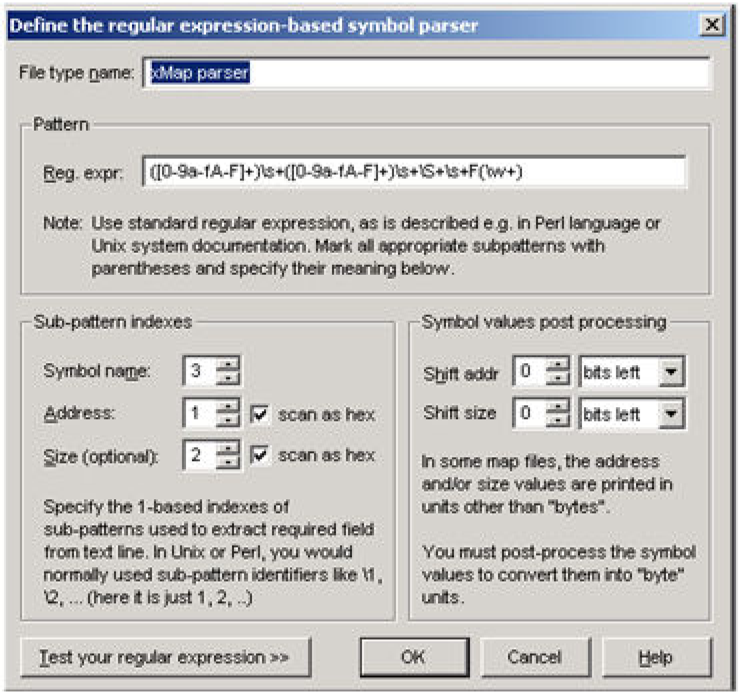
To identify the sub-patterns which describe the symbol parameters, put them in the round parentheses:

```
([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 to the symbol address (index 1).
- The next sub-pattern corresponds to the symbol size (index 2).
- The next sub-pattern corresponds to the symbol name (index 3).

Supply all the parameters prepared above in the regular expression dialog, as shown in the following figure.


Define the regular expression-based symbol parser

File type name: xMap parser

Pattern

Reg. expr: ([0-9a-fA-F]+)\s+([0-9a-fA-F]+)\s+\S+\s+F(\w+)

Note: Use standard regular expression, as is described e.g. in Perl language or Unix system documentation. Mark all appropriate subpatterns with parentheses and specify their meaning below.

Sub-pattern indexes

Symbol name: 3

Address: 1 [✓] scan as hex

Size (optional): 2 [✓] scan as hex

Specify the 1-based indexes of sub-patterns used to extract required field from text line. In Unix or Perl, you would normally used sub-pattern identifiers like \1, \2, ... (here it is just 1, 2, ..)

Symbol values post processing

Shift addr 0 bits left

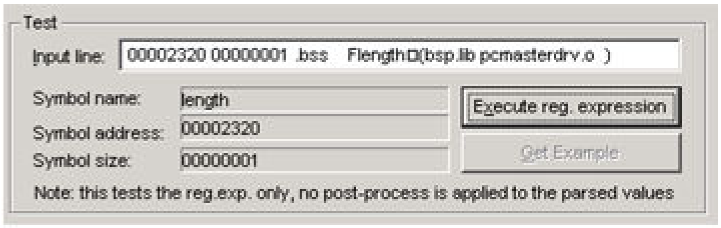
Shift size 0 bits left

In some map files, the address and/or size values are printed in units other than "bytes".

You must post-process the symbol values to convert them into "byte" units.

Figure 40. Regular expression-based xMap file parser

By pressing the *Test your regular* expression button, the dialog vertically expands and the test panel is displayed, as shown in the following figure. Cut and paste a single line from the *xMap* file to *Input line* field and press the *Execute reg. expression* button:


Test

Input line: 00002320 00000001 .bss Flength□(bsp.lib pcmasterdrv.o)

Symbol name: length

Symbol address: 00002320

Symbol size: 00000001

Note: this tests the reg.exp. only, no post-process is applied to the parsed values

Figure 41. Testing your regular expression

The *Symbol name*, *Symbol address*, and *Symbol size* output fields shall be displayed, as shown in the above figure. If you have errors, make sure that you have Internet Explorer 5.5 or higher installed on your machine for the regular expression functionality.

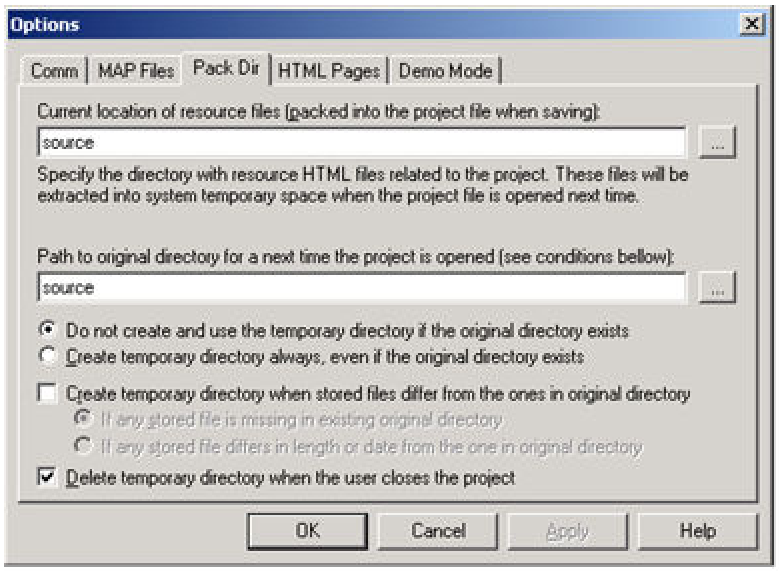
To finish our example, set a one-bit *shifting* of the *size* value in the *Symbol post-processing* parameters. This is done because the symbol size is printed in word units (16-bit) in the *xMap* file, but we need this value in bytes.

NOTE

All created regular-expression parsers are automatically 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 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 40. on page 42, you can choose to pack the resource files into the project file when it is saved.


Options

Comm | MAP Files | Pack Dir | HTML Pages | Demo Mode

Current location of resource files (packed into the project file when saving): source

Specify the directory with resource HTML files related to the project. These files will be extracted into system temporary space when the project file is opened next time.

Path to original directory for a next time the project is opened (see conditions bellow): source

(●) Do not create and use the temporary directory if the original directory exists
( ) Create temporary directory always, even if the original directory exists

[ ] Create temporary directory when stored files differ from the ones in original directory
(●) If any stored file is missing in existing original directory
( ) If any stored file differs in length or date from the one in original directory

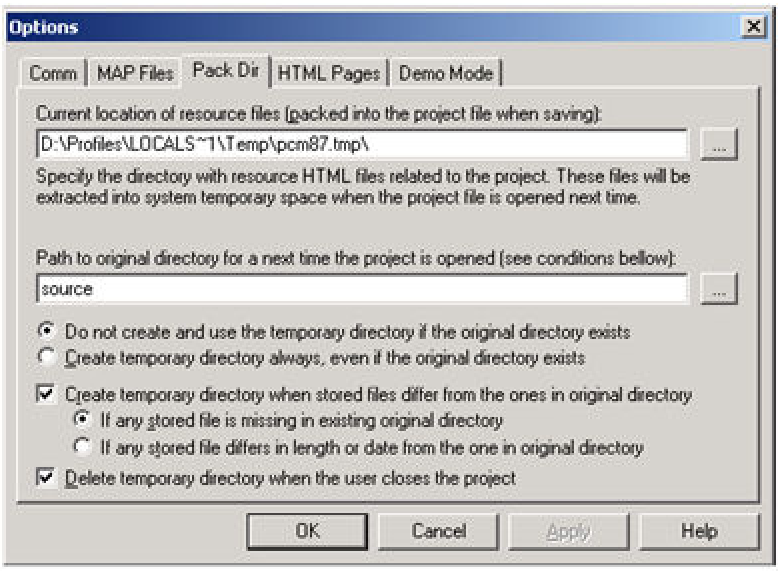
[✓] Delete temporary directory when the user closes the project

Figure 42. Pack directory options

To pack the 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 by the application the next time, 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 the default one for the rest of resource files. It is very important to specify the FreeMASTER paths to the HTML files or to the symbol file relative to the directory specified in the *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.



Current location of resource files (packed into the project file when saving):
D:\Profiles\LOCALS~1\Temp\pcm87.tmp\
Specify the directory with resource HTML files related to the project. These files will be extracted into system temporary space when the project file is opened next time.

Path to original directory for a next time the project is opened (see conditions bellow):
source

[✓ Do not create and use the temporary directory if the original directory exists]
Create temporary directory always, even if the original directory exists

[✓ Create temporary directory when stored files differ from the ones in original directory]
[✓ If any stored file is missing in existing original directory]
If any stored file differs in length or date from the one in original directory
[✓ Delete temporary directory when the user closes the project]

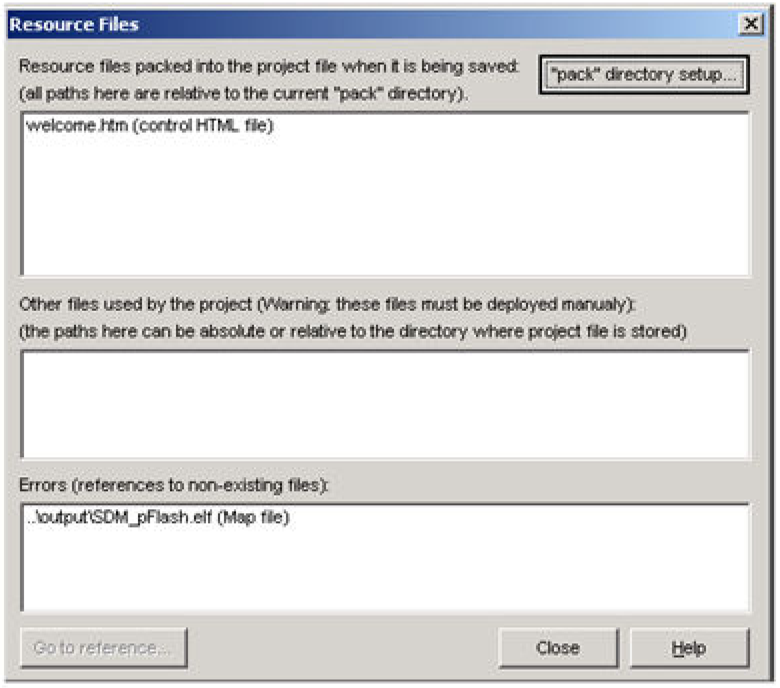
Figure 43. Pack directory options (example 2)

Using the other Pack Directory options shown in the above figure, you can set the conditions under which the temporary directory is created.

5.3.1 Resource files manager

Before deploying a complex project that 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 *Project / Resource Files* menu. The typical look of the window is shown in the following figure.



Resource files packed into the project file when it is being saved:
(all paths here are relative to the current "pack" directory).
welcome.htm (control HTML file)

Other files used by the project (Warning: these files must be deployed manualy):
(the paths here can be absolute or relative to the directory where project file is stored)

Errors (references to non-existing files):
..\output\SDM_pFlash.elf (Map file)

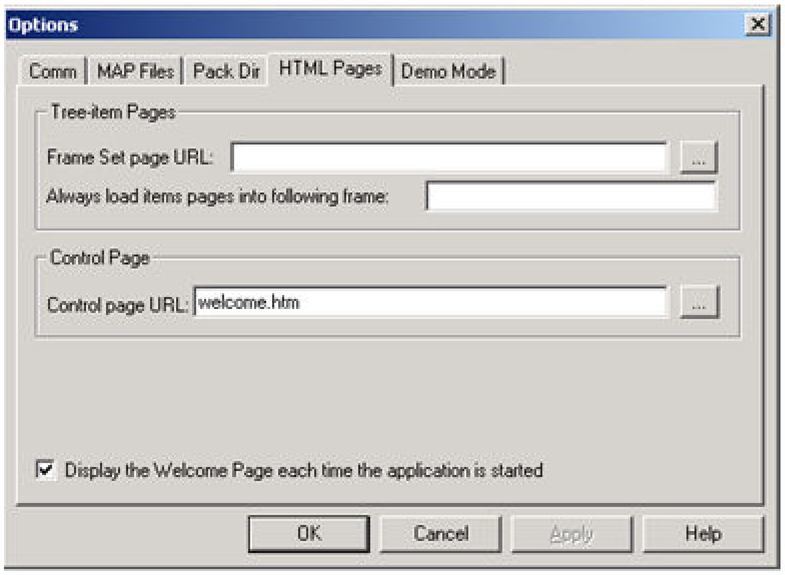
Figure 44. Resource files manager

The dialog displays all external files directly referenced by the project. Note that there can also be other files (for example, 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 are properly stored in the project file when it is saved. On other hosts, the files are extracted to a temporary space if needed.
- The second list in the dialog contains the files that are successfully used by the project, but located outside the pack directory. Their file names are specified either by the absolute path or by a path relative to the directory where the project is stored. However, the files are not automatically stored in the project file and you must deploy them manually.
- The third list contains references to non-existing 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 the 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 paths or URLs of the pages are specified in the property dialog for each tree item. On the *HTML Pages* options tab, shown in the following figure, you can specify the paths to common HTML files, displayed in the application main window.



Frame Set page URL:
Always load items pages into following frame:
Control page URL: welcome.htm
[✓] Display the Welcome Page each time the application is started

Figure 45. HTML pages options

Specify the *Frame Set page URL* if you want to divide the pages displayed for the project tree items into 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 may contain frames, controls, and scripts (for example, 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 displayed in the separate tab in the detail view of the application main window.

5.5 Demo mode

An important part of the FreeMASTER capabilities is the demonstration and description of the target board application. It is essential that the demonstration project, when prepared, is not accidentally modified. To prevent modification, the project's author can prevent the project from changes by switching it into the *Demo Mode*. See the following figure for details of the *Demo Mode* tab.

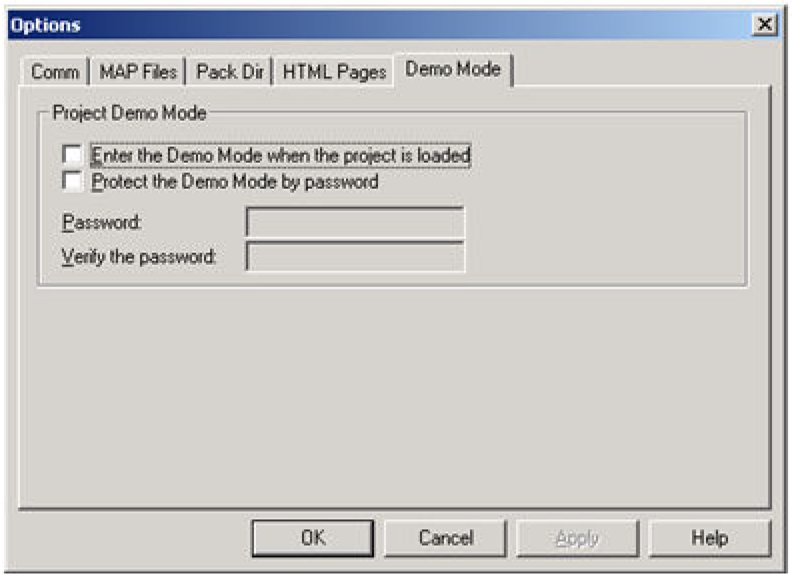
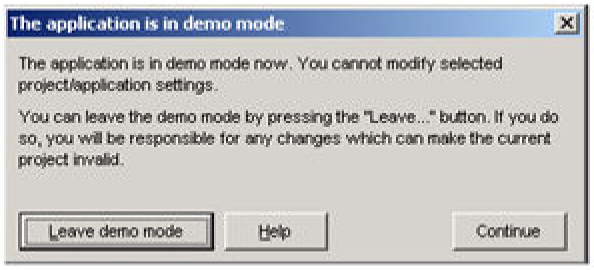


Figure 46. Demo mode options

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. The exit from the demo mode can be protected by a password.

In the demo mode, you cannot change the *Project Tree* item properties, add or remove the tree items, nor change any project options, except for those in the communication page.

When you want to leave the demo mode, the warning message shown in the following figure appears and you are prompted for the password if the demo mode is password-protected.



The application is in demo mode now. You cannot modify selected project/application settings.

You can leave the demo mode by pressing the "Leave..." button. If you do so, you will be responsible for any changes which can make the current project invalid.

Figure 47. Exit demo mode confirmation dialog

Chapter 6
HTML and scripting

All HTML pages used in the 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 the scripts to embed and access third-party ActiveX controls. The FreeMASTER application itself implements a non-visual ActiveX component to let the script-based code access and control the target board application.

The following text is recommended primarily for the 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, you can use special hyperlinks to let users navigate in the project tree or to invoke selected application commands (see Commands on page 28).

The application command invocation hyperlinks include:

- *HREF=pcmaster:cmd:cmdname(arguments)* sends the *cmdname* application command. The 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 the *cmdname* application command and waits until the target application processes it.
- *HREF=pcmaster:cmddlg:cmdname(arguments)* displays the “Send Application Command” dialog for the *cmdname* application command. Any specified argument values are filled into the appropriate fields in the dialog.
- *HREF=pcmaster:selitem:itemname:tabname* selects a project tree item named *itemname* and displays the detail view and watch view contents exactly as if you had selected the item manually. It then selects the specified tab in the detail view.

The 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 "; the version-independent name is " MCB.PCM ".

The 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 these 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 the connection status.
- *GetHtmlDocument* retrieves the HTML DOM object from the FreeMASTER window.
- *SendCommand* sends a FreeMASTER-defined command.
- *SendCommandDlg* opens a command dialog and send a FreeMASTER-defined command.
- *ReadVariable* reads a value from a FreeMASTER-defined variable.
- *WriteVariable* writes a value to a FreeMASTER-defined variable.
- *ReadMemory*, *ReadMemoryHex* read 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 a single value to a specified location.
- *GetCurrentRecorderData* retrieves the 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* enable to open and close a local file for temporary text-based storage.
- *LocalFileWriteString* writes text-based data to a file.
- *LocalFileReadString* reads text-based data from a 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* selects 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 the TSA Active Content items. Refer to the FreeMASTER Serial Driver documentation for more information about the TSA and Active Content.
- *PipeOpen, PipeClose, PipeWrite* and many other *Pipe* functions are used for lossless communication with the target board.
- *EnumVariables, EnumSymbols* enumerate the variables and symbols loaded in the current project.
- *GetDetectedBoardInfo* to retrieve information about the connected board.
- *GetCommPortInfo* to retrieve information about the current communication port settings.

Callback events implemented:

- *OnRecorderDone* called when the currently-selected recorder finishes downloading the new recorder data.
- *OnCommPortStateChanged* called when the communication port status changes.

- *OnBoardDetected* called when a valid connection to a target board is established.
- *OnVariableChanged* called when a subscribed variable changes.

6.3 FreeMASTER ActiveX object methods

6.3.1 GetAppVersion

`GetAppVersion (bsRetMsg)`

Description

This function gets the FreeMASTER application version.

Arguments:

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

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)`

Description

This function opens the specified project file.

Arguments:

bsProject [in]	String containing the project file path.

Return Value:

Boolean Value	*True* if open was successful.

6.3.3 StartStopComm

`StartStopComm (bStart)`

Description

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

Arguments:

bStart [in]	Set as *"True"* to start communication. Set as "*False"* to close communication

Return Value:

Boolean Value	*"True"* is returned if the command was successfully performed. "*False"* is returned if the command failed to open or close the communication interface.

6.3.4 IsCommPortOpen

```
IsCommPortOpen ()
```

Description

This function returns the state of the communication port.

Arguments:

None.

Return Value:

Boolean Value	*"True"* if the communication port is currently open. "*False"* if the port is closed.

6.3.5 IsBoardDetected

```
IsBoardDetected ()
```

Description

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

Arguments:

None.

Return Value:

Boolean Value	*"True"* if the communication port is currently open and the board is online. *"False"* if the port is closed or if the board could not be enumerated.

6.3.6 GetHtmlDocument

```
GetHtmlDocument (nWinId, bActivate)
```

Description

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

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 to activate the selected HTML view.

Return Value:

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

6.3.7 SendCommand

```
SendCommand (bsCmd, bsRetMsg)
```

Description

This function sends the FreeMASTER Application Command.

Arguments:

bsCmd [in]	String value with the command name and arguments that 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. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean value	*"True"* is returned if the command has been sent without errors. "*False"* is returned if the command could not be found in FreeMASTER project.

6.3.8 SendCommandDlg

```
SendCommandDlg (bsCmd, bsRetMsg)
```

Description

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

Arguments:

bsCmd [in]	String value with the command name and arguments that 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 the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean value	*"True"* is returned if the command was sent without errors. "*False"* is returned if the command could not be found in the FreeMASTER project.

6.3.9 ReadVariable

```
ReadVariable (bsVar, vValue, tValue, bsRetMsg)
```

Description

This function reads a 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 does not retrieve the live value from the target. To disable this caching mechanism, use an exclamation mark '!' in front of the variable name in the bsVar parameter.

Arguments:

bsVar [in]	String value with the name of the variable to be read. The variable must be defined in the FreeMASTER project that is currently opened.

Table continues on the next page...

Table continued from the previous page...

vValue [out, optional]	Returns a numeric representation of the variable *bsVar.* The value of this output parameter is mirrored in the LastVariable_vValue data member. Use it when the scripting language does not support the [out] arguments.
tValue [out, optional]	Returns a string value which represents the variable format and units necessary for displaying the variable value. The value of this output parameter is mirrored in the LastVariable_tValue data member. Use it when the scripting language does not support the [out] arguments.
bsRetMsg [out, optional]	When an error occurs, this value contains an error message; otherwise, it is empty. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean value	*"True"* is returned if the variable was read without errors. "*False"* is returned if the specified variable was not found in the FreeMASTER project or if a communication error occurred.

6.3.10 WriteVariable

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

Description

This function writes a value to a FreeMASTER variable.

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]	The value to write to the variable.
bsRetMsg [out, optional]	When an error occurs, this value contains an error message. Otherwise, it is empty. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean value	*"True"* is returned if the variable was written without errors. *"False"* is returned if the specified variable was not found in the FreeMASTER project, if the value passed was invalid, or if a communication error occurred.

6.3.11 ReadMemory

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

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

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

Description

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

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

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 the offset value.
vSize [in]	The size of the memory block to be read.
arrData [out, optional]	The return array of values. The value of this output parameter is mirrored in the LastMemory_data data member. Use it when the scripting language does not support the [out] arguments.
bsData [out, optional]	The return data in the string format. The value of this output parameter is mirrored in the LastMemory_hexstr data member. Use it when the scripting language does not support the [out] arguments.
bsRetMsg [out, optinal]	When an error occurs, this value contains an error message; otherwise, it is empty. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean value	*"True"* is returned if the memory block was read without errors. "*False"* is returned if the specified address was invalid or if a communication error occurred.

6.3.12 WriteMemory

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

Description

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

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 the offset value.
vSize [in]	The size of the memory block to be written.
arrData [in]	The safe array of bytes to be written. It must contain at least the vSize elements.
bsRetMsg [out, optinal]	When an error occurs, this value contains an error message. Otherwise, it is empty. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean value	*"True"* is returned if the memory block was read without errors. "*False"* is returned if the specified address was invalid or if a communication error occurred.

6.3.13 ReadXxxArray

1. ReadXxxArray

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

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

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

To access an 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 an array of 4-byte floating-point numbers:

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

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

To access an array of 8-byte floating-point numbers:

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

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

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 that can be used in different scripting environments:

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

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 the 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. The value of this output parameter is mirrored in the LastMemory_data data member. Use it when the scripting language does not support the [out] arguments.
bsRetMsg [out, optinal]	When an error occurs, this value contains an error message. Otherwise, it is empty. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean value	*"True"* is returned if the memory block was read without errors. "*False"* is returned if the specified address was invalid or if a communication error occurred.

6.3.14 WriteXxxArray

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

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

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

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

To access an array of 4-byte floating-point numbers:

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

To access an array of 8-byte floating-point numbers:

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

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.

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 the 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, optinal]	When an error occurs, this value contains an error message. Otherwise, it is empty. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean value	*“True”* is returned if the memory block was written without errors. “*False”* is returned if the specified address was invalid or if a communication error occurred.

6.3.15 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)
```

Description

These functions read a memory cell from the target application and return it to the caller in the form of an 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 that can be used in different scripting environments:

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

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 the offset value.
vSize [in]	The size of the variable to read.
vValue [out, optional]	The return value. The value of this output parameter is mirrored in the LastVariable_vValue data member. Use it when the scripting language does not support the [out] arguments.
bsRetMsg [out, optinal]	When an error occurs, this value contains an error message. Otherwise, it is empty. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean value	*"True"* is returned if the memory was read without errors. "*False"* is returned if the specified address was invalid or if a communication error occurred.

6.3.16 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)
```

Description

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

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 the offset value.
vSize [in]	The size of the target variable.
vValue [in]	The value to be written.
bsRetMsg [out, optinal]	When an error occurs, this value contains an error message. Otherwise, it is empty. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean value	*"True"* is returned if the memory block was written without errors. "*False"* is returned if the specified address was invalid or if a communication error occurred.

6.3.17 GetCurrentRecorderData

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

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

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

Arguments:

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

Return Value:

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

6.3.18 GetCurrentRecorderSeries

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

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

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 safe array of variants, which can be used in scripting languages, such as VBScript, and in compiled languages, such as Visual Basic.

- *GetCurrentRecorderSeries_t* returns data in the safe array of type “double floating point”, which can be used in compiled languages, such as Visual Basic. Using typed arrays is faster than using arrays of variants.

Arguments:

bsSerieName [in]	String with the name of the series whose data is to be retrieved.
arrData [out, optional]	Return array of the values. The value of this output parameter is mirrored in the LastRecorder_data data member. Use it when the scripting language does not support the [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 value. The value of this output parameter is mirrored in the LastRecorder_timeBaseSec data member. Use it when the scripting language does not support the [out] arguments.
bsRetMsg [out, optional]	When an error occurs, this value contains an error message. Otherwise, it is empty. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

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

6.3.19 StartCurrentRecorder

```
StartCurrentRecorder (bsRetMsg)
```

Description

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

Argument:

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

Return value:

Boolean value	"*True*" is returned if the recorder was started successfully. "*False*" is returned if there is no recorder currently active.

6.3.20 StopCurrentRecorder

```
StopCurrentRecorder (bsRetMsg)
```

Description

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

Argument:

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

Return Value:

Boolean value	*"True"* is returned if the recorder was successfully stopped. "*False"* is returned if there is no recorder currently active.

6.3.21 GetCurrentRecorderState

```
GetCurrentRecorderState (nState, bsRetMsg)
```

Description

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

Arguments:

nState
 	
[out, optional]
	Returns numeric value identifying the current recorder state. The valid state codes are:
	
	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;
	
	The value of this output parameter is mirrored in the LastRecorder_state data member. Use it when the scripting language does not support the [out] arguments.

bsRetMsg [out, optional]	When an error occurs, this value contains an error message. Otherwise, it contains a text description of the current recorder state. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean value	*"True"* is returned if the recorder state was read successfully. "*False"* is returned if there is no recorder currently active.

6.3.22 RunStimulators

`RunStimulators (bsStimNames)`

Description

This function starts one or more FreeMASTER variable stimulators.

Arguments:

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

Return Value:

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

6.3.23 StopStimulators

```
StopStimulators (bsStimNames)
```

Description

This function halts one or more FreeMASTER variable stimulators.

Arguments:

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

Return Value:

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

6.3.24 LocalFileOpen

```
LocalFileOpen (bsFileName, bsOpenMode)
```

Description

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

Arguments:

bsFileName [in]	Name of the 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 the current project file. FMSTR_PACKDIR_PATH - location of the current resource module folder.

Table continues on the next page...

Table continued from the previous page...

bsOpenMode [in, optional]	Specifies the 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.

Return Value:

Handle Value	Numeric handle to file or a false boolean value in case the file could not be opened.

6.3.25 LocalFileClose

`LocalFileClose (nHandle)`

Description

This function closes the file identified by the nHandle parameter.

Arguments:

nHandle [in]	Handle to the file to be closed.

Return Value:

Boolean Value	*“True”* is returned if the file was successfully closed. *“False”* is returned in case the file handle is not valid.

6.3.26 LocalFileWriteString

`LocalFileWriteString (nHandle, bsData, nCharsToWrite, bUnicode)`

Description

This function writes data to a file.

Arguments:

nHandle [in]	Handle to open file.
bsData [in]	Text to be written to the file.

Table continues on the next page...

Table continued from the previous page...

nCharsToWrite [in, optional]	The number of characters to be actually written. If this parameter is omitted or set 0, the whole text passed in the vData parameter is written.
bUnicode [in, optional]	*"True"* - write data in unicode format (not supported). *"False"* - write data in ASCII format.

Return Value:

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

6.3.27 LocalFileReadString

`LocalFileReadString (nHandle, nCharsToRead, bUnicode, bsRetString)`

Description

This function reads data from a file.

Arguments:

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

Return Value:

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

6.3.28 GetSymbolInfo

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

Description

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

Arguments:

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

Return Value:

Boolean Value	*"True"* is returned if the symbol exists and the information was retrieved. "*False"* is returned when the symbol was not found.

6.3.29 GetStructMember

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

Description

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

Arguments:

bsTypeName [in]	Name of the structure type.
bsMember [in]	Name of the structure member.

Table continues on the next page...

Table continued from the previous page...

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

Return Value:

Boolean Value	*“True”* is returned if the type exists and information was retrieved. “*False”* is returned when the structure type was not found.

6.3.30 GetAddressInfo

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

Description

This function returns information about the memory location of 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.

Arguments:

vAddr [in]	Memory address.
vSize [in]	Size of the memory area to look up.

Table continues on the next page...

Table continued from the previous page...

bIsExactMatch [out, optional]	Returns "*True*" if the returned symbol matches the exact memory address and area size. The value of this output parameter is mirrored in the LastAddressInfo_exact data member. Use it when the scripting language does not support the [out] arguments.
bsSymbolName [out, optional]	Returns the best-matching name of the memory area. The value of this output parameter is mirrored in the LastAddressInfo_name data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean Value	*"True"* is returned if the memory area was mapped to any global or static symbol. "*False"* is returned when the memory area was not mapped to any symbol.

6.3.31 DefineSymbol

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

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 the symbols for dynamically allocated variables or other non-global objects for which you know the address and type. When the symbol is defined, a FreeMASTER variable can be defined in the GUI (or again by the *DefineVariable* script method).

Arguments:

bsSymbol [in]	Symbol name. It can also contain special characters, such as ., ->, or [] when defining complex symbols mapping structure members.
vAddr [in]	Symbol address.

Table continues on the next page...

Table continued from the previous page...

bsTypeName [in]	Name of the type behind the symbol. Typically, this is the name of an 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, FreeMASTER defines also symbols for each structure member. The separator between the symbol name and the 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 the symbols with valid C language syntax.
vSize [in]	Symbol size. You may also use special expressions, such as "sizeof(TYPE)" or mathematical expressions.
bsRetMsg [out, optional]	When an error occurs, this value contains an error message. Otherwise, it is empty. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

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 the application executable file.

6.3.32 DeleteAllScriptSymbols

```
DefineSymbol();
```

Description

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

Arguments:

None.

Return Value:

Boolean Value	This method returns the "*True*" value.

6.3.33 SubscribeVariable

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

Description

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

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 the variable changes too fast. The *OnVariableChanged* 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. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Numeric Value	A value other than zero is the Subscription ID which can be used to unsubscribe the variable. This ID also identifies the variable in the *OnVariableChanged* event callback. *Zero* when the variable could not be subscribed.

6.3.34 UnSubscribeVariable

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

Description

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

Arguments:

vNameorId [in]	Variable name used to subscribe the variable or subscription ID which was returned from the *SubscribeVariable* call.
bsRetMsg [out, optional]	When an error occurs, this value contains an error message. Otherwise, it is empty. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean Value	*"True"* is returned if the variable was unsubscribed. "*False"* if an error occurred.

6.3.35 SelectItem

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

Description

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

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: - *control* ... Control Page Tab - *blkinfo* ... Parent Block description page - *iteminfo* ... Item description page - *scope* ... Scope graph tab - *recorder* ... Recorder graph tab

Return Value:

Boolean Value	*"True"* is returned when the item was found and selected. *"False"* otherwise.

6.3.36 DefineVariable

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

Description

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

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 a list of valid JSON parameter names, save the 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.

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. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean Value	*"True"* is returned when the object was created or modified. *"False"* if an error occurred.

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));
}
```

6.3.37 DefineScope

`DefineScope (bsName, bsDefString, bsRetMsg);`

Description

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

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 a list of valid JSON parameter names, save the 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.

Arguments:

bsName [in]	Name of the FreeMASTER oscilloscope item to be created or modified in the project tree.

Table continues on the next page...

Table continued from the previous page...

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. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean Value	*"True"* is returned when the object was created or modified. *"False"* if an error occurred.

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.strinigy(def));
}
```

6.3.38 DefineRecorder

`DefineRecorder (bsName, bsDefString, bsRetMsg);`

Description

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

6.3.39 IsBoardWithActiveContent

```
IsBoardWithActiveContent ();
```

Description

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

Arguments

None.

Return Value:

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

JScript Example

See the JScript example of using this function in the source code of the 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.

6.3.40 EnumHrefLinks

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

Description

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

Arguments

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

Return Value:

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

JScript Example

See the JScript example of using this function in the source code of the 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 EnumProjectFiles

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

Description

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

Arguments

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

Return Value:

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

JScript Example

See the JScript example of using this function in the source code of the 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 PipeOpen

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

Description

This function initializes the pipe object used to communicate with the target MCU using a lossless 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 for the communication to run correctly. See the FreeMASTER Serial Driver documentation for more details.

Arguments

vPort [in]	The port number which identifies the pipe channel. Only the lower 16 bits of the value are used.
vTxBufferSize [in, optional]	Local transmit buffer size. This buffer is used to accumulate the data being transmitted to the target MCU by calling any PipeWrite function until the MCU is ready to accept it. Optional, defaults to 0x1000.

Table continues on the next page...

Table continued from the previous page...

vRxBufferSize [in, optional]	Local receive buffer size. This buffer is used to accumulate the data being received from the target MCU until the script reads them by calling any PipeRead function. Optional, defaults to 0x1000.
bsRetMsg [out, optional]	When an error occurs, this value contains an error message. Otherwise, it is empty. The value of this output parameter is mirrored in the LastRetMsg data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Boolean Value	*"True"* is returned when the pipe was successfully initialized. *"False"* is returned otherwise.

6.3.43 PipeClose

```
PipeOpen (vPort);
```

Description

This function de-initializes the pipe object.

Arguments

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

Return Value:

Boolean Value	*"True"* is returned when the pipe was successfully closed. *"False"* is returned otherwise.

6.3.44 PipeFlush

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

Description

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

Arguments

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

Return Value:

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

6.3.45 PipeSetDefaultTxMode

```
PipeSetDefaultTxMode (vTxAllOrNothing);
```

Description

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

Arguments

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

Return Value:

None.

6.3.46 PipeSetDefaultRxMode

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

Description

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

Arguments

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

Return Value:

None.

6.3.47 PipeSetDefaultStringMode

```
PipeSetDefaultStringMode (vUnicode);
```

Description

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

Arguments

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

Return Value:

None.

6.3.48 PipeGetRxBytes

```
PipeGetRxBytes (vPort);
```

Description

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

Arguments

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

Return Value:

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

6.3.49 PipeGetTxBytes

```
PipeGetTxBytes (vPort);
```

Description

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

Arguments

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

Return Value:

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

6.3.50 PipeGetTxFree

```
PipeGetTxFree (vPort);
```

Description

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

Arguments

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

Return Value:

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

6.3.51 PipeGetRxBufferSize

```
PipeGetRxBufferSize (vPort);
```

Description

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

Arguments

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

Return Value:

Integer value	Local receive buffer size.

6.3.52 PipeGetTxBufferSize

```
PipeGetTxBufferSize (vPort);
```

Description

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

Arguments

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

Return Value:

Integer value	Local transmit buffer size.

6.3.53 PipeWriteString

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

Description

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

Arguments:

vPort [in]	The port number which identifies the pipe channel. Only the lower 16 bits of the value are used.
vString [in]	The string to be written.
vCharsToWrite [in, optional]	The number of characters to be written to a pipe. This is an optional parameter. When omitted, the whole string is written (no termination character).
vAllOrNothing [in, optional]	When non-zero, the PipeWrite function attempts to write all data at once to the local transmit buffer. When the data do not fit into the buffer, no data are transmitted. This is an optional parameter, the default value is determined by the 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 a partial value. This is an optional parameter; the default value is determined by the PipeSetDefaultStringMode function.

Return Value:

Integer value	This function returns the number of characters written.

6.3.54 PipeWriteXxxArray

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

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

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

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

To write an array of 4-byte floating-point numbers:

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

To write an array of 8-byte floating-point numbers:

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

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 values transmitted partially.

Arguments:

vPort [in]	The port number which identifies the pipe channel. Only the lower 16 bits 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 values to be written.
vCount [in, optional]	The number of elements to be written to the pipe. This is an optional parameter. When omitted, the whole array is written.
vAllOrNothing [in, optional]	When non-zero, the PipeWrite function attempts to write all data to the local transmit buffer at once. When the data would not fit to the buffer, no data are transmitted. This is an optional parameter, the default value is determined by the PipeSetDefaultTxMode function.

Return Value:

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

6.3.55 PipeReadString

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

Description

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

Arguments:

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

Table continues on the next page...

Table continued from the previous page...

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

Return Value:

Integer value	This function returns the number of characters read.

6.3.56 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 a 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 a 4-byte floating-point variable:

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

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

To access an 8-byte floating-point variable:

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

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

Description

These functions read a requested count of integer or floating-point numbers from a pipe and store them into the 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 scripting languages, such as VBScript, and for compiled languages, such as Visual Basic. Because the VBScript engine does not handle variants encapsulating 2 and 4-byte unsigned integer types, this method converts such values to a floating-point format before returning.
- *PipeReadXxxArray_v* is the same as *PipeReadXxxArray*, except that it does not perform the VBScript translation for 2 and 4-byte unsigned integer types.
- *PipeReadXxxArray_t* returns the data in the VB array of strictly typed values, which can be used in compiled languages, such as Visual Basic. Using typed arrays is significantly faster than using arrays of variants.

Arguments:

vPort [in]	The port number which identifies the pipe channel. Only the lower 16 bits of the value are used.
vElemSize [in]	The size of an array element in bytes. The total number of bytes to be read from a 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 an optional parameter and the default value is determined by PipeSetDefaultRxMode.
vCount [in, optional]	The number of elements to be written to a pipe. This is an 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 attempts to read all required data at once. When the data are not available until the timeout expires, no data are received. This is an optional parameter and the default value is determined by the PipeSetDefaultRxMode function.
arrData [out, optional]	This parameter variable receives the resulting VB array. The value of this output parameter is mirrored in the LastPipe_data member. Use it when the scripting language does not support the [out] arguments.

Return Value:

Integer value	This function returns the number of array elements read.

6.3.57 EnumVariables

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

Description

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

Arguments

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

Return Value:

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

6.3.58 EnumSymbols

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

Description

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

Arguments

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

Return Value:

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

6.3.59 GetDetectedBoardInfo

```
GetDetectedBoardInfo ()
```

Description

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

Arguments

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

LastBoardInfo_protVer	Protocol version byte value.
LastBoardInfo_dataBusWidth	Data bus width, typically 1 on most 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 the Serial Driver documentation.
LastBoardInfo_description	Board or FreeMASTER driver description string.

Return Value:

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

6.3.60 GetCommPortInfo

```
GetCommPortInfo ()
```

Description

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

Arguments

There are no arguments, this function fills the output information into the 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 a communication plug-in is used.
LastCommPortInfo_isRS232	True when the native serial port or the USB-to-serial port is used.

Table continues on the next page...

Table continued from the previous page...

LastCommPortInfo_speed	Serial baudrate used. Valid only if the isRS232 member is true.
LastCommPortInfo_portNum	COM port number. Valid only if the isRS232 member is true.
LastCommPortInfo_portHint	Hint word used to lookup the serial port. Valid only if the isRS232 member is true. The hint may be any word contained in the serial port description and may be specified by the user instead of the direct COM port number. This FreeMASTER feature may be useful when using USB-to-Serial converters which map to random COM port numbers.

Return Value:

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

6.4 FreeMASTER ActiveX properties

Starting with version 1.2.20, FreeMASTER also supports scripting languages that do not implement by-reference passing of the function parameters (for example, JScript). The “output” parameters in all ActiveX methods are declared as “optional” and do not need to be specified by the caller when the FreeMASTER method is called. After the call is made, all output values can be fetched using ActiveX properties, as described in the table below.

For each property in the FreeMASTER ActiveX interface, there is also a “GetLast...” function available and it 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 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 the “arrData” output parameter by the last call to one of the ReadMemory or ReadXxxArray methods.

Table continues on the next page...

Table 1. FreeMASTER ActiveX object properties (continued)

Property Name	Description
LastRecorder_data	The array of values returned in the “arrData” output parameter by the last call to the GetCurrentRecorderData or GetCurrentRecorderSeries methods.
LastRecorder_serieNames	The array of values returned in the “arrSerieNames” output parameter by the last call to the GetCurrentRecorderData method.
LastRecorder_timeBaseSec	The value of the “timeBaseSec” output parameter returned by the last call to the GetCurrentRecorderData method.
LastRecorder_state	The value of the “nState” output parameter returned by the last call to the GetCurrentRecorderState method.
LastLocalFile_string	The “bsRetString” text buffer returned by the last call to the LocalFileReadString method.
LastSymbolInfo_size	The “vSymSize” value returned by the last call to the GetSymbolInfo method.
LastSymbolInfo_addr	The “vSymAddr” value returned by the last call to the GetSymbolInfo method.
LastMemberInfo_size	The “vMembSize” value returned by the last call to the GetStructMember method.
LastMemberInfo_offset	The “vMembOff” value returned by the last call to the GetStructMember method.
LastAddressInfo_name	The “bsSymbolName” value returned by the last call to the GetAddressInfo method.
LastAddressInfo_exact	The “bIsExactMatch” value returned by the last call to the GetAddressInfo method.
LastLinkName	The name returned by the last call to the EnumHrefLinks or EnumProjectFiles methods.
LastPipe_data	The pipe data returned by the last call to any of the PipeRead methods.
LastEnum_name	The enumerated name returned by the last call to the 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 ()
```

Description

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

6.5.2 OnCommPortStateChanged

`OnCommPortStateChanged (vPortOpen)`

Description

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

6.5.3 OnBoardDetected

`OnBoardDetected ()`

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)`

Description

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

6.6 Examples

The example applications shown in this section demonstrate how to initialize and use the 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 the ActiveX methods. Using a 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", 100End Sub Function
      PageTimer()
        'perfroms 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", 100End 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 the FreeMASTER ActiveX control:

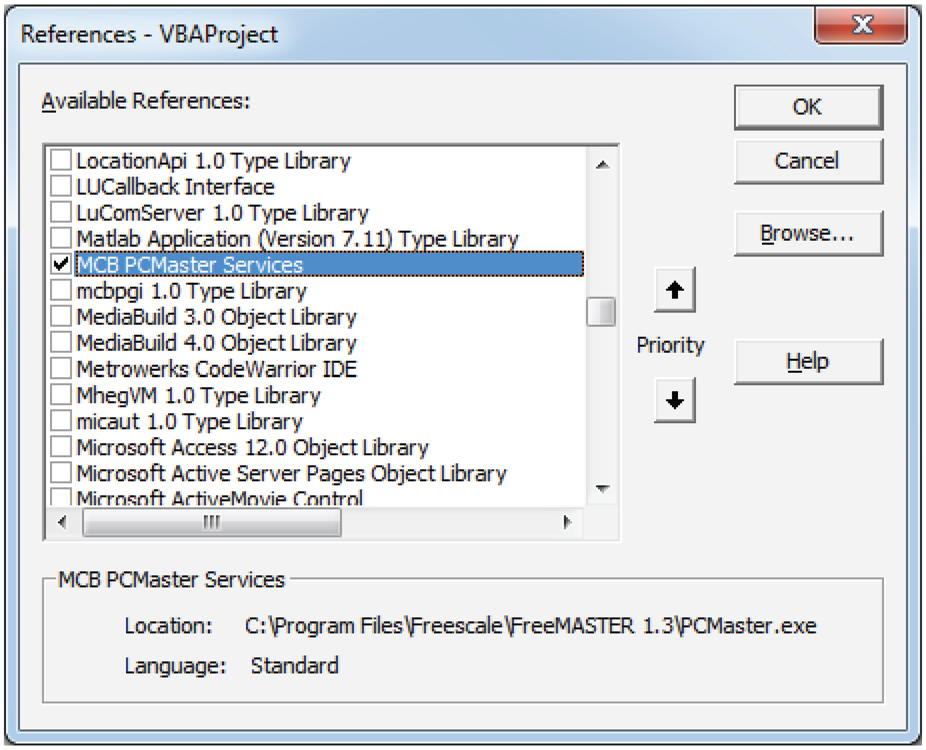
- JScript does not natively support by-reference [output] parameters. The parameters should be avoided and the output values should be retrieved from the LastResult and other Last*xxx* property members. See FreeMASTER ActiveX properties on page 87 for more details.
- JScript uses a different format of arrays which is not compatible with the “safearray” format used by the ActiveX interface. Special conversion routines must be used, as demonstrated in the following example code.

```
<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);
    }
    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 coverts 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 the Visual Basic for Applications (VBA) project in Excel. Register a reference to the FreeMASTER services in the *Tools->References...* dialog.


References - VBAProject
Available References:
OK
Cancel
Browse...
Help
Priority
LocationApi 1.0 Type Library
LUCallback Interface
LuComServer 1.0 Type Library
Matlab Application (Version 7.11) Type Library
MCB PCMaster Services
mcbpgi 1.0 Type Library
MediaBuild 3.0 Object Library
MediaBuild 4.0 Object Library
Metrowerks CodeWarrior IDE
MhegVM 1.0 Type Library
micaut 1.0 Type Library
Microsoft Access 12.0 Object Library
Microsoft Active Server Pages Object Library
Microsoft ActiveMovie Control

MCB PCMaster Services

Location: C:\Program Files\Freescale\FreeMASTER 1.3\PCMaster.exe

Language: Standard

Figure 48. Registering 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 in the 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

- The FreeMASTER Tool home page at www.nxp.com/FREEMASTER
- FreeMASTER Serial Communication Driver User Manual. Installed with the FMASTERSCIDRV software from www.nxp.com/webapp/sps/download/license.jsp?colCode=FMASTERSCIDRV
- The DSP56800E_Quick_Start Initialization and Development Tool home page at www.nxp.com/pages/dsc-quick-start-initialization-and-development-tool:DSP56800EQUICKSTARTT

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
3.0	06/2019	Reformatted the document to the current NXP look and feel.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 06/2019

Document identifier: FMSTERUG