# FMSTERUG

## FreeMASTER for Embedded Applications

**Rev. 4.8 — 18 September 2024**

**User guide**

# 1   Introduction

## 1.1  Overview

This user guide 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 version 3 application is fully backward compatible with previous 2.x and 1.x versions and even with the oldest "PC Master" 1.0 version.

## 1.2  Supported platforms

The FreeMASTER desktop application can be installed on any Windows® OS-based system supported today.

The FreeMASTER installation package also contains the FreeMASTER "Lite" service which may run on other operating systems (like Linux OS) and may act as a communication interface between various control pages and target microcontrollers.

For the target microcontroller side, communication drivers supporting the latest version V4 of the FreeMASTER communication protocol are available through the NXP MCUXpresso SDK suite, other SDKs, and also as a standalone downloadable package. An older driver supporting previous versions of the protocol is available as a standalone package for legacy Freescale microcontroller platforms like HCS08, S12, S12X, S12Z, ColdFire, and Power Architecture.

### 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 FreeMASTER on the HCS08, S12Z, ColdFire, and Arm® Cortex®-M 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 BDM and JTAG debugging probes like P&E Multilink, SEGGER jLink, and Arm CMSIS-DAP.

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 JTAG or 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 **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, FlexCAN, or other kinds of peripheral modules.

The **FreeMASTER TCP/UDP Communication Plug-in** enables the FreeMASTER to connect over a network protocol directly to the targets which support Ethernet or WiFi connection and a TCP/IP stack. The first prototype of this plug-in is part of FreeMASTER 3.1.2. The sample MCU applications named "fmstr_net", utilizing the lwIP stack and wired Ethernet connection, are available for the i.MX-RT1xxx and Kinetis K64F platforms as a part of the MCUXpresso SDK version 2.10. This support is going to be extended in future versions to cover more platforms and a WiFi communication.

The network communication enables multiple clients (running instances of the FreeMASTER tool) to connect to a single server (MCU board) which may cause some access conflicts. The FreeMASTER driver released in

the MCUXpresso SDK 2.10 supports multiple sessions and "feature locking", which enable all clients to access target resources in a properly synchronized way.

Version 3.1.3 further extends the network plug-in by a communication over SEGGER RTT interface. This interface uses a JTAG port and is supported by SEGGER J-Link debugger probes. An advantage of RTT communication is that FreeMASTER may be used along with an active debugger session.

For more details about the communication plug-in modules, see the "readme" documents installed together with the latest FreeMASTER application.

## 1.3  Where to find the latest version

The latest installation package of the FreeMASTER package can be found at www.nxp.com/freemaster in the "Downloads" section. The page also contains older versions of the tool.

The target microcontroller drivers are at the same page for a standalone download. However, it is recommended to use the driver from the MCUXpresso SDK at www.nxp.com/mcuxpresso/sdk or other kinds of SDKs provided by NXP. Visit mcuxpresso.nxp.com and use the SDK Builder to generate an SDK package with the FreeMASTER middleware component and example applications.

## 1.4  FreeMASTER features

- Graphical environment and easy-to-understand navigation
- Simple serial (UART) native connection and other options possible on selected platforms (BDM, JTAG, CAN, TCP, UDP, SEGGER RTT, and others)
- Real-time access to C variables in a running target microcontroller application
- Visualization of real-time data in the Oscilloscope window
- Acquisition of fast data changes using the on-target Recorder
- Built-in support for standard variable types (integer, floating point, bit fields) and extended types (fractional)
- Value interpretation using custom-defined text messages
- Several built-in transformations for real-type variables
- Automatic variable address and size extraction from compiler output files
- Support for Target-Side Addressing (TSA) information about variable objects and types to be retrieved from the target application
- Demo mode with password protection support
- HTML-based control or description pages rendered by Internet Explorer, Chromium, or Microsoft Edge engines.
- ActiveX interface to enable VBScript or JavaScript control over embedded applications from Internet Explorer and Microsoft Edge pages or a third-party application that supports the ActiveX technology.
- JSON-RPC interface to enable control from Chromium or Microsoft Edge views, a standalone Chrome or Edge browser, node.js scripts, Python scripts, or a third-party application that supports the JSON-RPC protocol.
- New in version 3.2: the Microsoft Edge WebView2 component used inside FreeMASTER main application window enables a partial compatibility with COM+ and ActiveX objects including the FreeMASTER ActiveX object itself.

## 1.5  FreeMASTER online community

FreeMASTER community forum is available online at https://community.nxp.com/community/freemaster. Use it to post questions, share feedback, or reach out to the development or support teams.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**3 / 142**

## 2   Questions and answers

While writing this user guide, the following questions were raised. Because the answers to these questions clarify the terms and topics described further in the document, 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 of the real-world uses are:

- Real-time debugging—FreeMASTER enables 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:

- Variable Watch—the values of selected variables are presented in a grid in a defined text-based format. Values can be modified directly in the grid.
- 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 sample rate of the Oscilloscope. While the Oscilloscope 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 Section 4.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**4 / 142**

## 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 RS-232 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.

The FreeMASTER version 3.1.2 and MCUXpresso SDK version 2.10 also introduce a new communication option of a direct TCP or UDP communication with a target board.

## 2.6 What are all of these dialog boxes for?

In Section 4, 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 the debug process
- To provide service maintenance capabilities
- To learn about your application during the 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 the 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 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.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

**Rev. 4.8 — 18 September 2024**

Document feedback

5 / 142

## 2.11 What is the Oscilloscope?

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

## 2.12 What is the Control Page?

The Control Page is a page encoded in the HTML tagging language and equipped with JavaScript code. The page is displayed inside the FreeMASTER window using a built-in Internet Explorer, Microsoft Edge WebView2, or Chromium browser components. The control page behaves just like a common webpage loaded from a local computer file.

FreeMASTER users may create any control page and code a script which may communicate with the "parent" FreeMASTER to gain access to the target MCU variables. Such a page and script may show variable values using graphical objects in the page (like gauges or sliders) and it may contain push-buttons and other controls to control the embedded application.

## 2.13 Has the Control Page changed in FreeMASTER 3.0?

FreeMASTER 3.0 still supports the old Internet Explorer rendering engine and ActiveX interface which enables the JavaScript code to access the FreeMASTER features. This should assure full backward compatibility with older projects.

However, version 3.0 brings brand new support of the built-in Chromium browser component. The Control Page may now be also displayed in a standalone Chrome browser running on the same or even remote computer and communicate with the FreeMASTER tool using the modern JSON-RPC protocol. Because Chromium does not support the old ActiveX technology, the existing pages and scripts must be rewritten to use the JSON-RPC communication. On the other hand, JSON-RPC brings support for asynchronous JavaScript programming using a Promise interface and a lot of modern coding techniques used in today's progressive web applications.

A standalone Chrome browser may be used as a perfect JavaScript debugger environment when developing the new-style FreeMASTER Control Pages.

New in version 3.2: the new version enables the third option of how the Control Page or a custom dashboard can be displayed. A new Microsoft Edge WebView2 browser component can now be used in addition to Internet Explorer and Chromium options. The Microsoft Edge fully supports the new JSON-RPC technology as well as it adds partial support of COM+ and ActiveX object scripting. This makes Microsoft Edge a great option when migrating older applications based on Internet Explorer into a new modern browser environment. See the Internet Explorer migration guide in Section 6.10.

## 2.14 What is FreeMASTER Lite?

FreeMASTER Lite is a new software service whose first version appears in the FreeMASTER 3.0 package.

The service may be started on a computer which is physically attached to the target microcontroller board and provides a JSON-RPC interface for remote clients to access the board. The JSON-RPC interface implemented by the service is almost identical to that provided by the standard FreeMASTER tool, as discussed in the previous question. Unlike the full FreeMASTER application though, the Lite service does not have a user interface. It is configured by a local configuration file and runs silently on the user computer.

The control pages and other clients (like script applications written in *node.js* or Python) may connect to the service from local or remote computers and access the target microcontroller application. The major difference from the standard FreeMASTER application is that the Lite service also acts as a standard web server and may provide control pages and their resources to remote clients like tablets or mobile phones.

## 2.15 Can I access connect to multiple target boards at once?

One running FreeMASTER instance may only be connected to a single target board. However, you can run multiple instances and connect each of them to a different board.

When using TCP or UDP network communication, the target board may support multiple sessions and let multiple FreeMASTER client instances to connect. All connected instances will be able to read and write the target memory, but there is a "feature locking" mechanism which will protect the use of high-level features, such as the Recorder or Pipe Communication, so that only the first client will be granted access. The next instance will only be able to use the same feature after it is unlocked and no longer used by the original one.

More typically, you will need to have data from multiple targets to be collected within a single application running on a host PC. With FreeMASTER, it is possible to create a Control Page which uses multiple JSON-RPC or ActiveX connections to collect data from multiple running FreeMASTERs and show all of them on a single page. Each FreeMASTER instance must use a different JSON-RPC server port to make this scenario possible.

Version 3.1.3 supports automatic JSON-RPC port allocation, so each running instance uses a different server port. A script running in an HTML application inside FreeMASTER can detect the connection parameters. See Section 6.3 and the example code in section Section 6.8.2.

# 3 Installation

## 3.1 System requirements

**Operating system**: Microsoft Windows 7 or later. Microsoft Windows 10 or later is recommended.

**Required software:** The Chromium Embedded Framework and Microsoft Edge WebView2 components are installed along with the FreeMASTER software.

**Hard drive space:** 1.5 GB of free space is required.

**Other hardware requirements:** Mouse and the serial RS-232 or USB ports for communication with the target board and network access for remote operation. You can optionally use the USB-to-CAN, CMSIS-DAP, Segger J-Link, PE Micro Multilink, or other communication or debugging interfaces.

## 3.2 Enabling connection to the target application

To enable the FreeMASTER connection to the target board application, use the driver acquired from MCUXpresso SDK available at www.nxp.com/mcuxpresso/sdk or downloaded from the www.nxp.com/freemaster homepage.

The recommended and fastest way to start using FreeMASTER is by trying one of the sample applications.

When using a microcontroller driver supporting the FreeMASTER communication protocol, you may see references to the new version (v4) of the protocol as well as some older versions. The v4 protocol version has been defined in 2019 and is supported by FreeMASTER version 2.5 and later.

FreeMASTER desktop application version 2.5 and all future versions will support the v4 protocol and will remain backward compatible with older protocol versions.

## 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 FreeMASTER homepage, run it, and proceed according to the instructions on the screen.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**7 / 142**

The FreeMASTER desktop application is available only for Windows OS. A Linux OS installer is also available, but includes only FreeMASTER Lite service.

FreeMASTER Lite installation requires an activation code that can be acquired free of charge from www.nxp.com webpage. Direct link to the FreeMASTER product license page is available on the component selection window during the installation process. The license key can be also accessed in the FreeMASTER Lite Product Information section of the main *Software Licensing and Support* page. In both cases, you should be logged into your NXP account.

## 3.4 Running FreeMASTER

After the installation is complete, run the FreeMASTER using an icon in the Windows OS Start menu. This runs FreeMASTER in the default mode with all features enabled and both JSON-RPC and ActiveX servers activated and listening for incoming connections (see more details in Section 6). The FreeMASTER main executable (*pcmaster.exe*) may also be started from a command line or manually created Windows OS shortcut. This enables you to specify one or more command-line options and optionally also a project filename to open by default. Use the "/help" option to display all possible command-line options.

# 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:
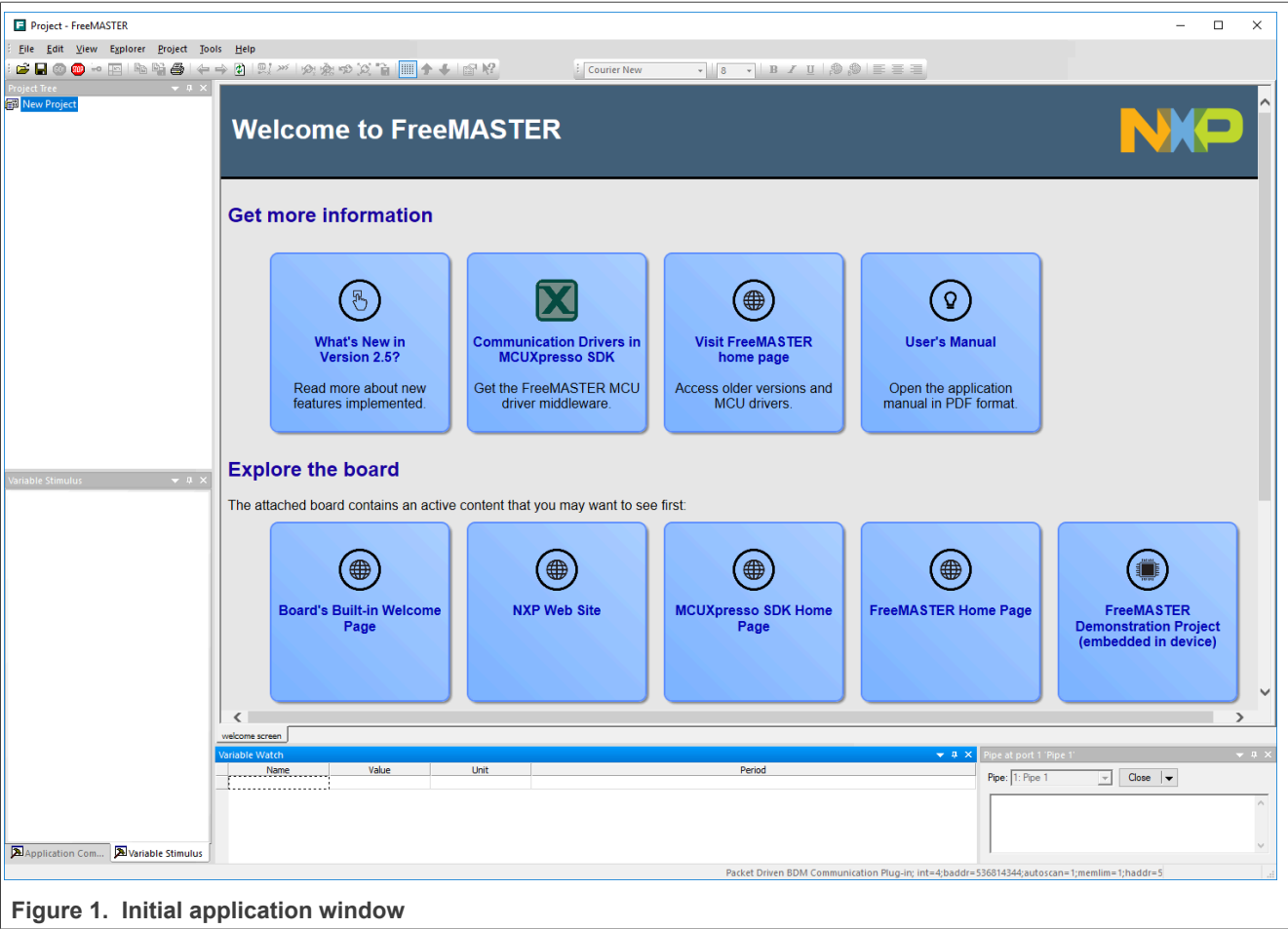
**Figure 1. Initial application window**

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 MCUXpresso SDK example application.

FMSTERUG

**User guide**

All information provided in this document is subject to legal disclaimers.

**Rev. 4.8 — 18 September 2024**

© 2024 NXP B.V. All rights reserved.
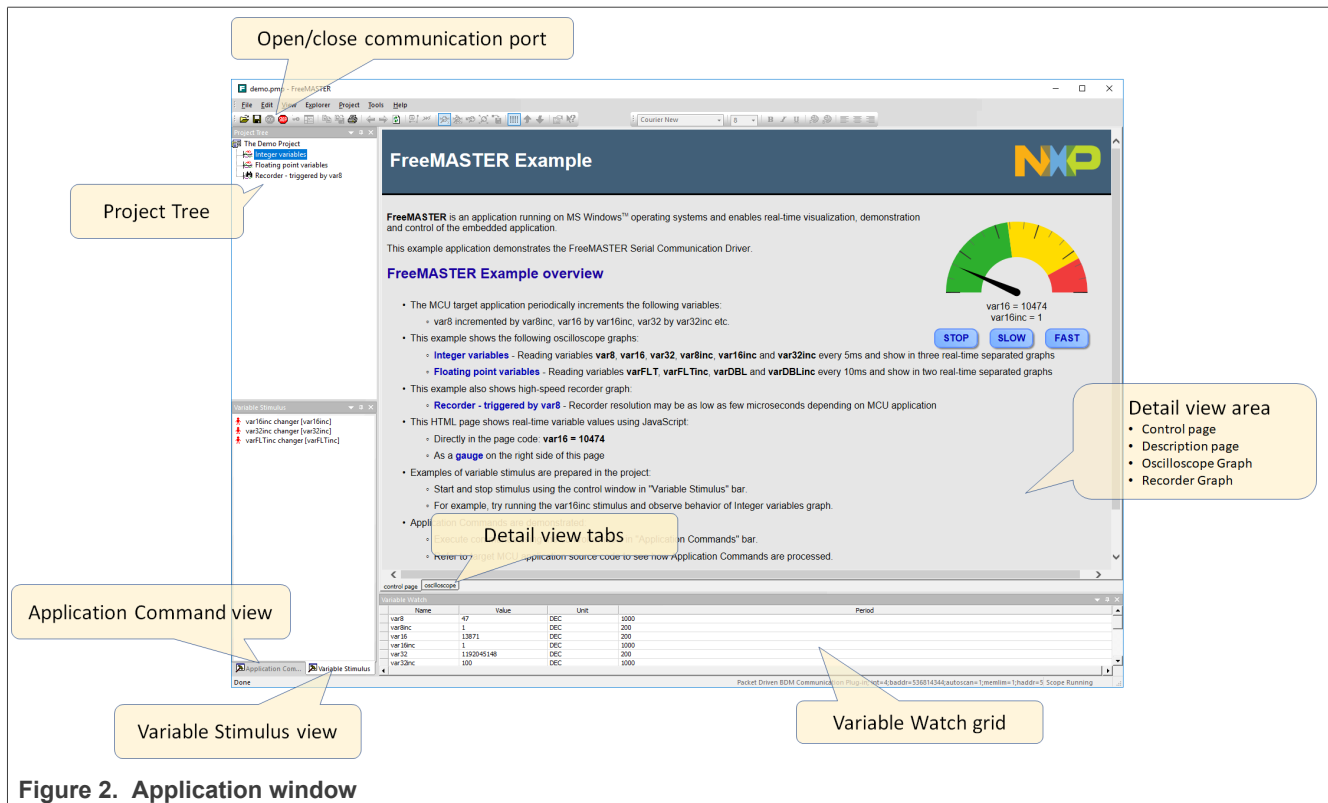
Document feedback

**9 / 142**

Figure 2. Application window

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

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

- Control page = an HTML page created to control the target system. When Control Page is defined, it is instantly available regardless of what item is selected in the Project Tree items to enable the user to control the board at any time.
- Algorithm block description = an HTML page or another document whose URL is defined in the selected Project Tree item's properties. The content of this view changes when the Project Tree item selection changes.
- Current item help = another HTML document whose URL is defined in the Oscilloscope or Recorder properties.
- Oscilloscope = a real-time graph displaying the application variables, as defined in the Oscilloscope properties.
- Recorder = a graph displaying the recorded application variables, as defined in the Recorder properties.
- Pipe = A text-based or graph-based view of FreeMASTER Pipe data.

The Variable Watch pane contains the list of variables assigned to the watch. The pane displays the immediate variable values and enables the user to change them if it is enabled in the variable definition.

All the information related to an application is stored in a single project file with a *.pmp* extension or a newer XML-formatted *.pmpx* extension. Such project file includes the communication settings and options, the Project Tree, HTML pages, real-time chart definitions, Variable Watch interface settings, variables, commands, stimulators, and all other objects used.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**10 / 142**

The project file may be open anytime to restore all objects and settings in the FreeMASTER application. The new *.pmpx* format also enables the user to save the window layout and the layout of all view panes used in the project. To enable it, select the Store Layout in Project File item in the View menu.
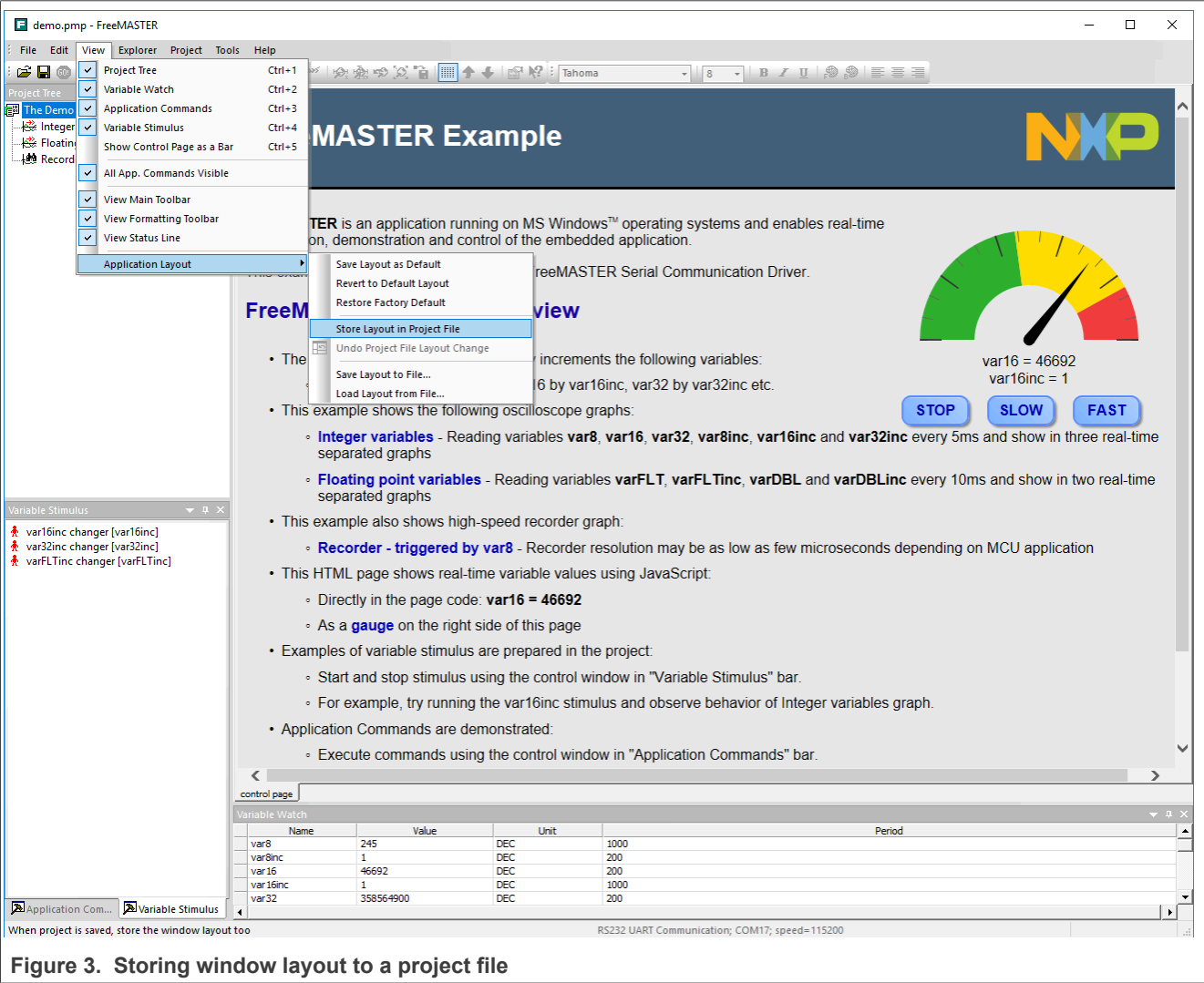


Figure 3.  Storing window layout to a project file

## 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". The user can change the properties of this block or add sub-blocks, Oscilloscopes, Recorders, or Pipes to the structure.

The properties can be changed and new Project Tree items can be added in the context menu which appears after right-clicking the tree item to be modified.

### 4.1.1.1  Project block and sub-block

A project block typically covers an integral component of the application or algorithm demonstrated with FreeMASTER. 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 Figure 4.

FMSTERUG

**User guide**

**Rev. 4.8 — 18 September 2024**
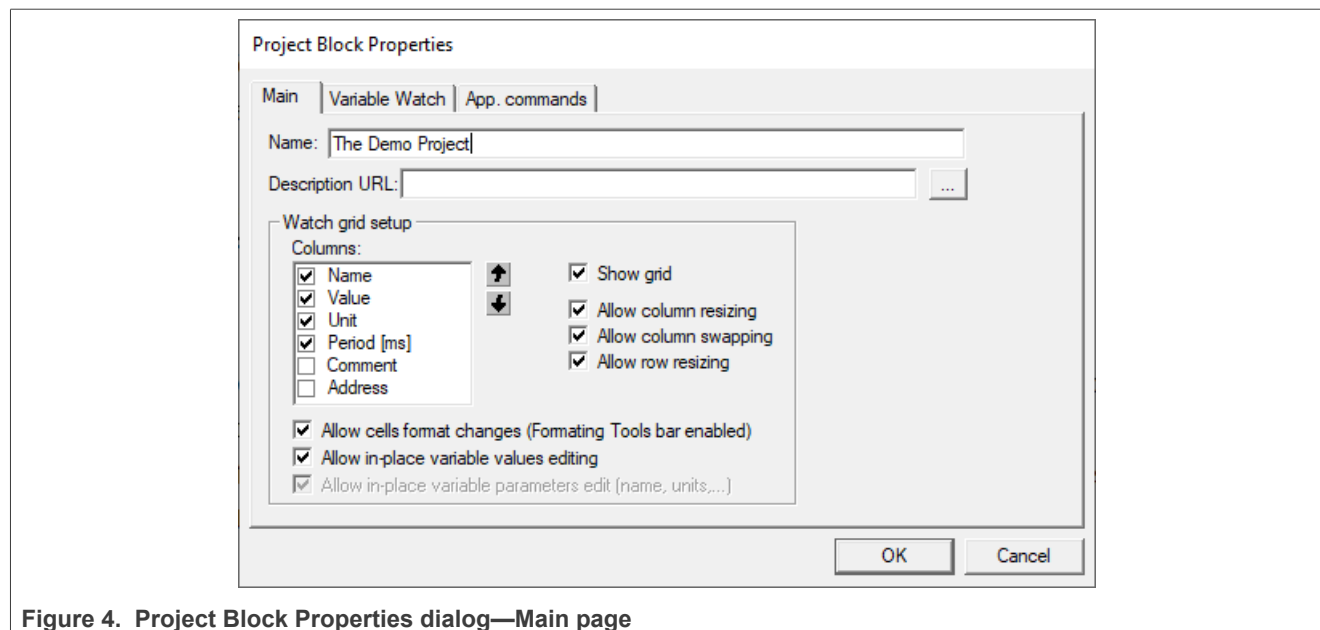
Document feedback

**11 / 142**

**Figure 4. Project Block Properties dialog—Main page**

The Main page contains the following user configuration items:

- Name = the name of the project block that is displayed in the Project Tree.
- Description URL = select a description URL or a path to the *.htm* or *.html* files to be shown in the Detail View pane under the Algorithm Block Description tab. With the demo application used as an example, the description page is left empty, causing the Algorithm Block Description tab to be hidden. See Section 4.1.2 for more details.
- Watch-grid setup = select the columns to display in the watch-grid (Name, Value, Unit, Period, Comment, Address), 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 the Toolbar (see Section 4.6.2), 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.
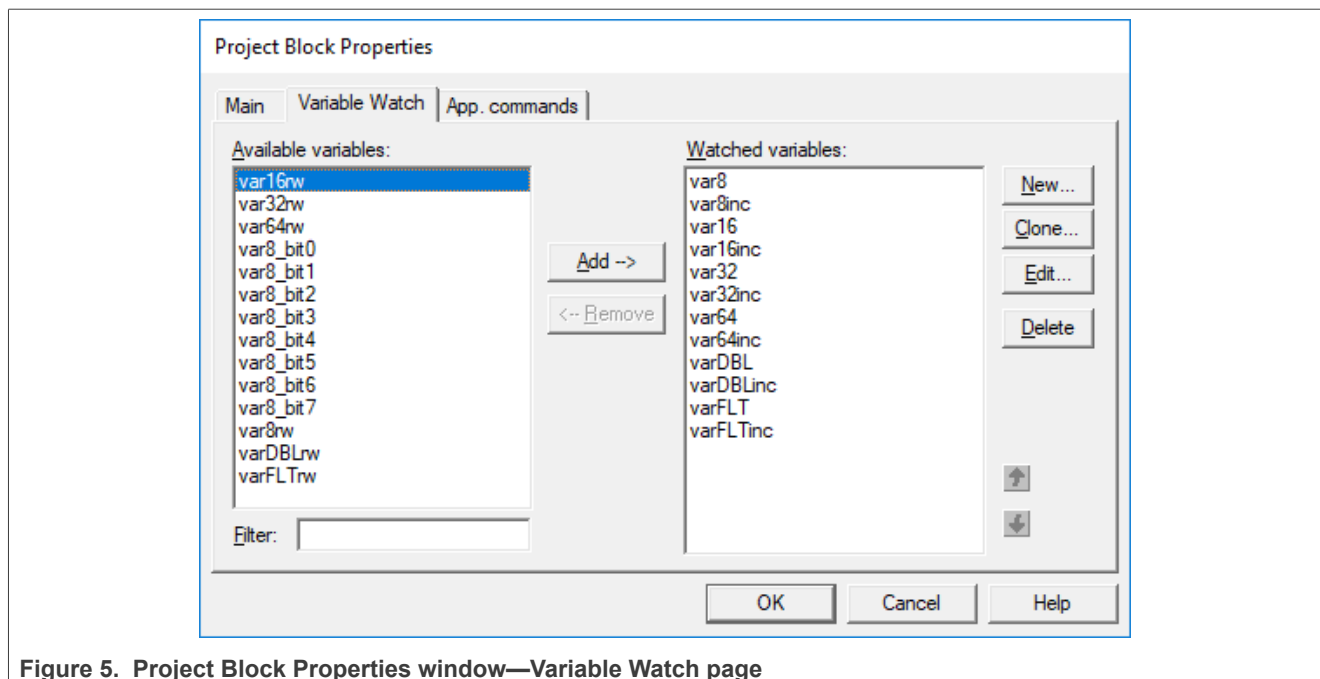
FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**12 / 142**

**Figure 5. Project Block Properties window—Variable 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. Use the following buttons:

- Add/Remove = moves variables into and out of the Watched variables window.
- New = creates a new variable (see Section 4.2).
- 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 = set 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 Section 4.3). 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 Commands pane in the context of this project block and also enables the management of application commands.
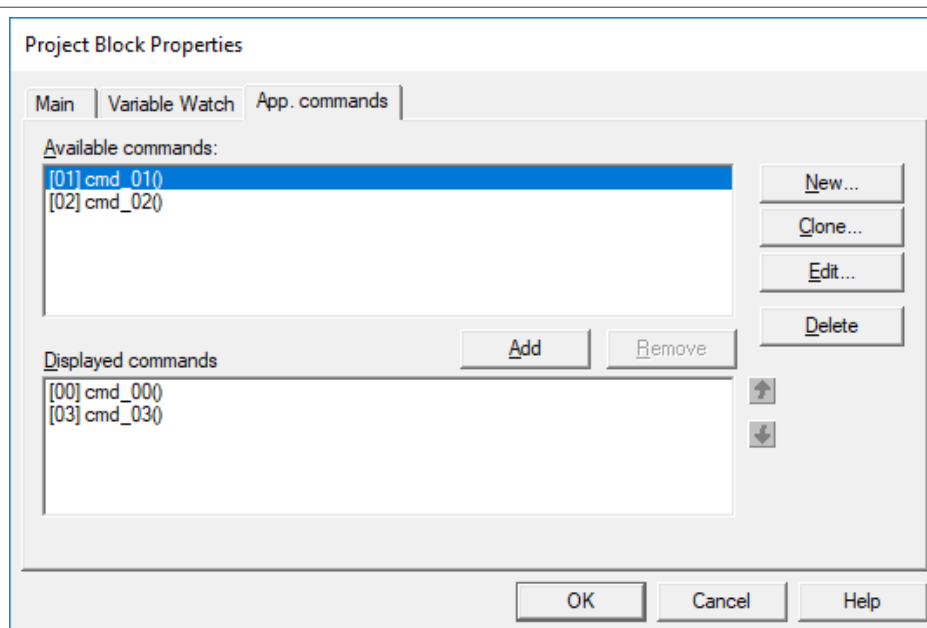
**Figure 6. Project Block Properties window—App. commands page**

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 Commands pane. Use the Remove button for reverse operation. Use the following buttons:

- New = creates a new command (see Section 4.3).
- 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 = set the display order of the commands in the Fast Access pane.
  **Note:** It is often more convenient to show all Application Commands in the Commands pane, regardless of what item is selected in the Project Tree. There is an All App. Commands Visible item in the View menu, which causes all commands to be visible always and unconditionally.

### 4.1.1.2 Oscilloscope

The Oscilloscope item in the Project Tree structure defines a real-time oscilloscope chart to be shown in the Detail View pane. The properties window (shown in the following figure) enables you to configure the appearance and characteristics of the oscilloscope chart.

The Main page contains these user configuration items:

- Name = the name of the Oscilloscope 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 document explains the chart variables and settings to the user.
- Oscilloscope properties = common properties for all oscilloscopes variables.
- Period = Oscilloscope sampling period.
- Buffer = the number of samples kept in the chart's local buffer. Enlarge this value to be able to scroll back to more recent points within the chart.
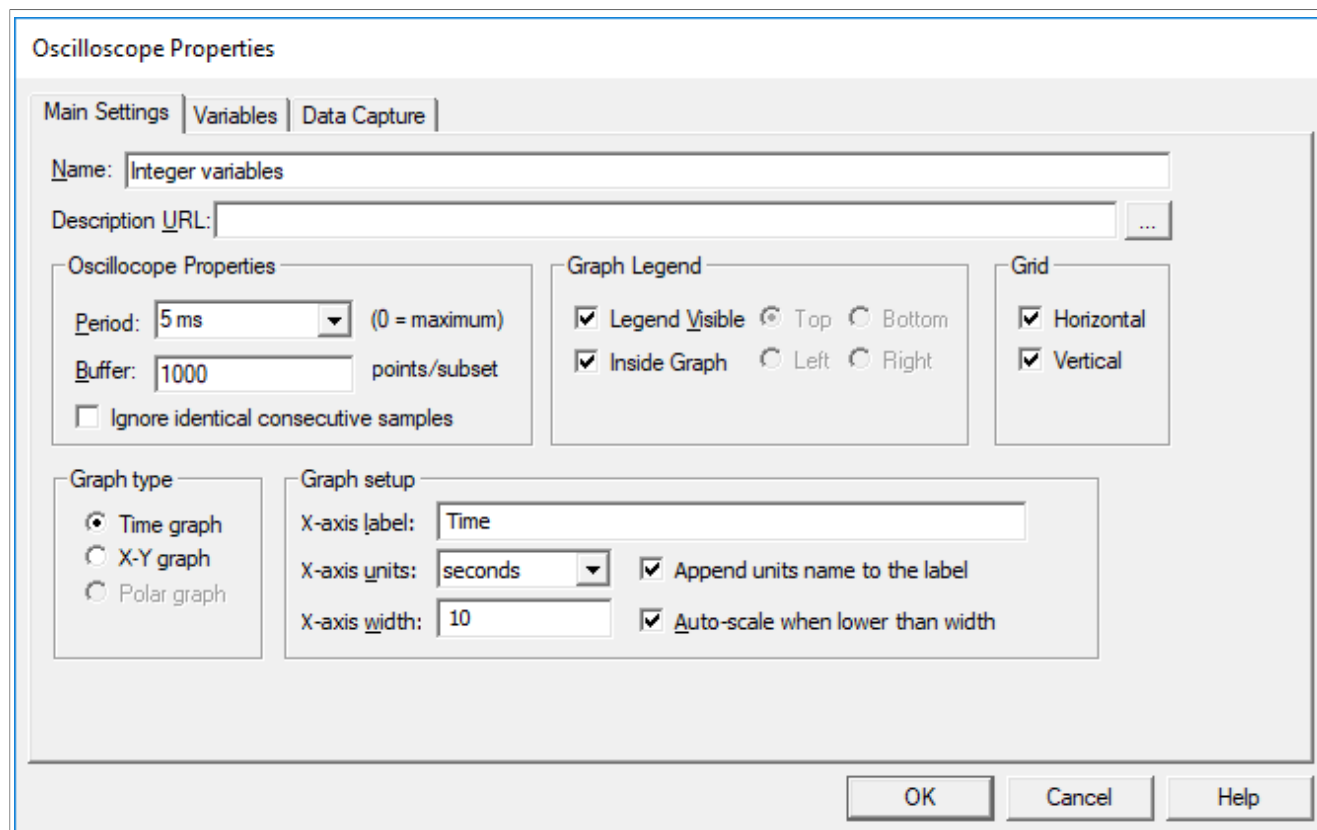- Legend location = set the visibility and location of the chart legend.

Figure 7.  Oscilloscope 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 Oscilloscope starts to operate 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 the X-axis values.
  – X-axis min = sets the X-axis lower limit value.
  – X-axis max = sets the X-axis upper limit value.

The Variables page is used to assign variables to be displayed in the Oscilloscope chart. The number of variables displayed is limited by FreeMASTER driver settings in the target microcontroller application. Use the Add Variable button to add a new slot in the chart and use the drop-down list to assign a variable into the slot.
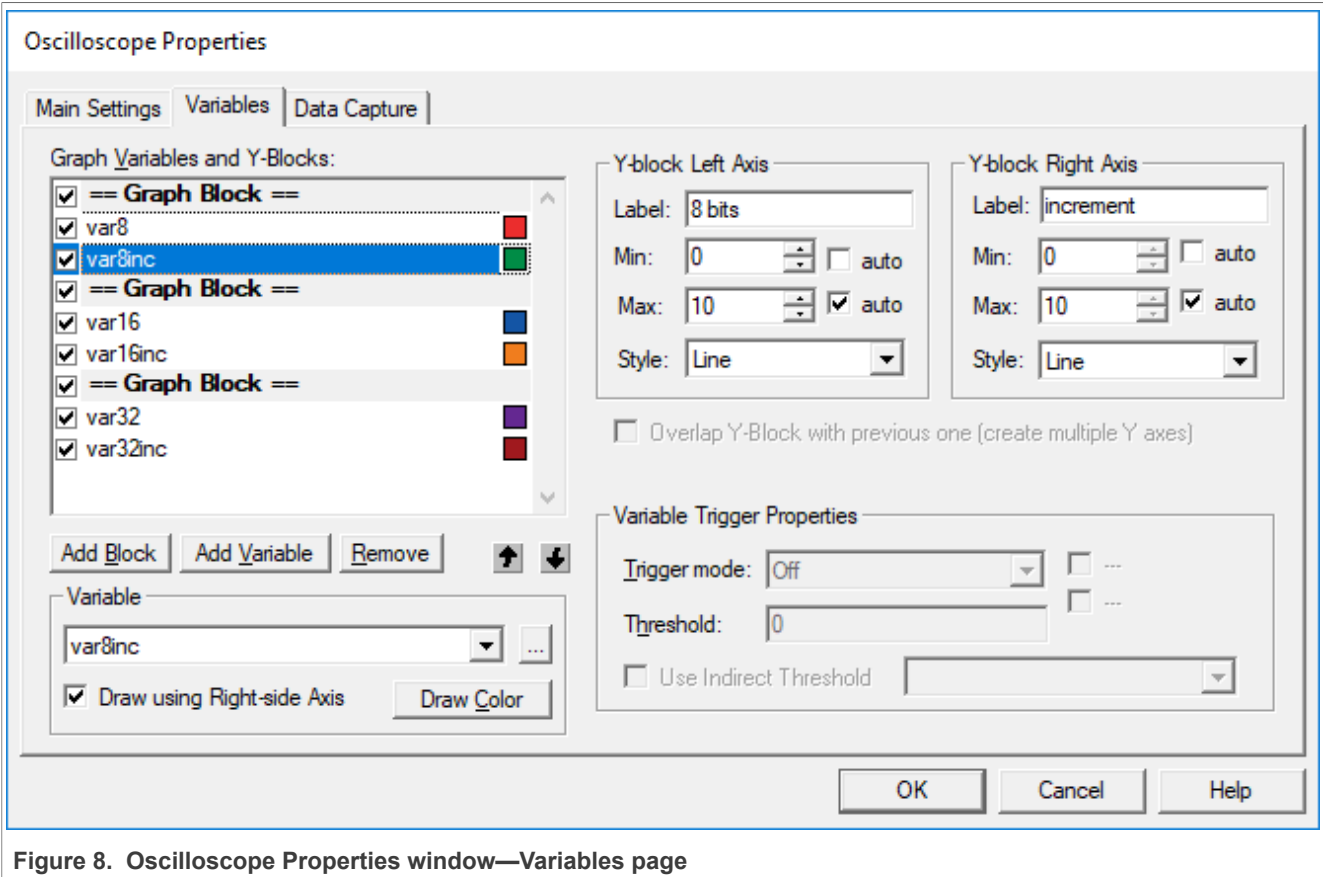
Figure 8.  Oscilloscope Properties window—Variables page

The Y block is a graph element represented by one left Y axis and, optionally, also one right Y axis. The Y blocks can be drawn separately, or overlapped in the graph. Use the Add Block button to create a new Y block separator and use the mouse to drag and drop it within the variable list. The separator items separate groups of variables which are put to a common Y block in the graph.

- In the Y-block Left Axis frame, set the axis name and range by specifying the minimum and maximum axis value, or tick 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.
- Tick the Draw using Right-side Axis option for any selected variable to assign the variable to the right axis of the Y-block. With this option checked, edit the right axis properties in the Y-block Right Axis frame.

The resulting oscilloscope chart is displayed in the following figure. Note that the var8inc, var16inc, and var32inc variables were assigned to the right axis.

FMSTERUG
All information provided in this document is subject to legal disclaimers.
© 2024 NXP B.V. All rights reserved.

**User guide**
**Rev. 4.8 — 18 September 2024**
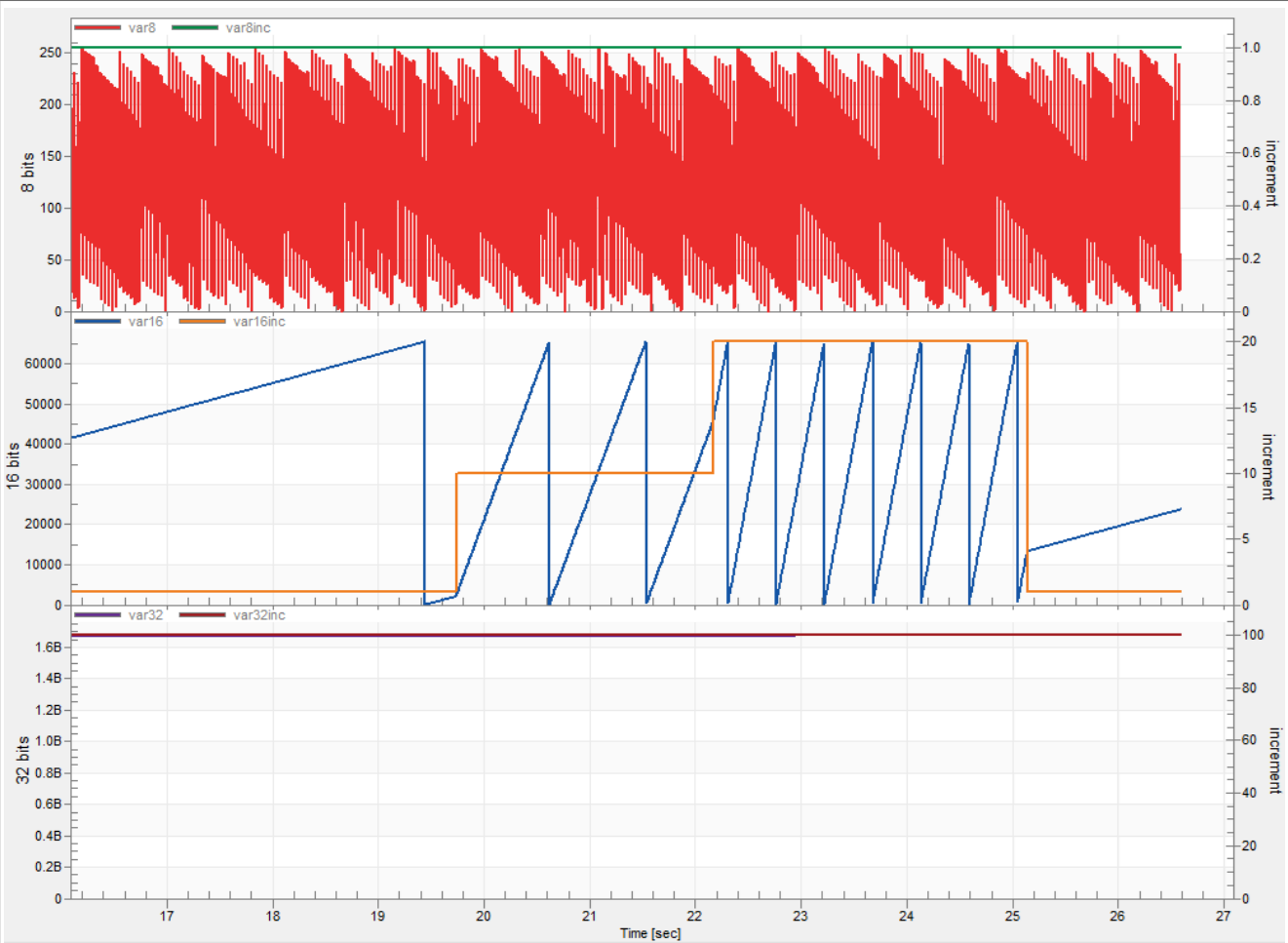Document feedback
**16 / 142**

**Figure 9. Basic oscilloscope chart**

The subsequent Y-blocks may overlap in the graph, causing multiple Y axes to be displayed. Use the Overlap Y-block with previous one option:
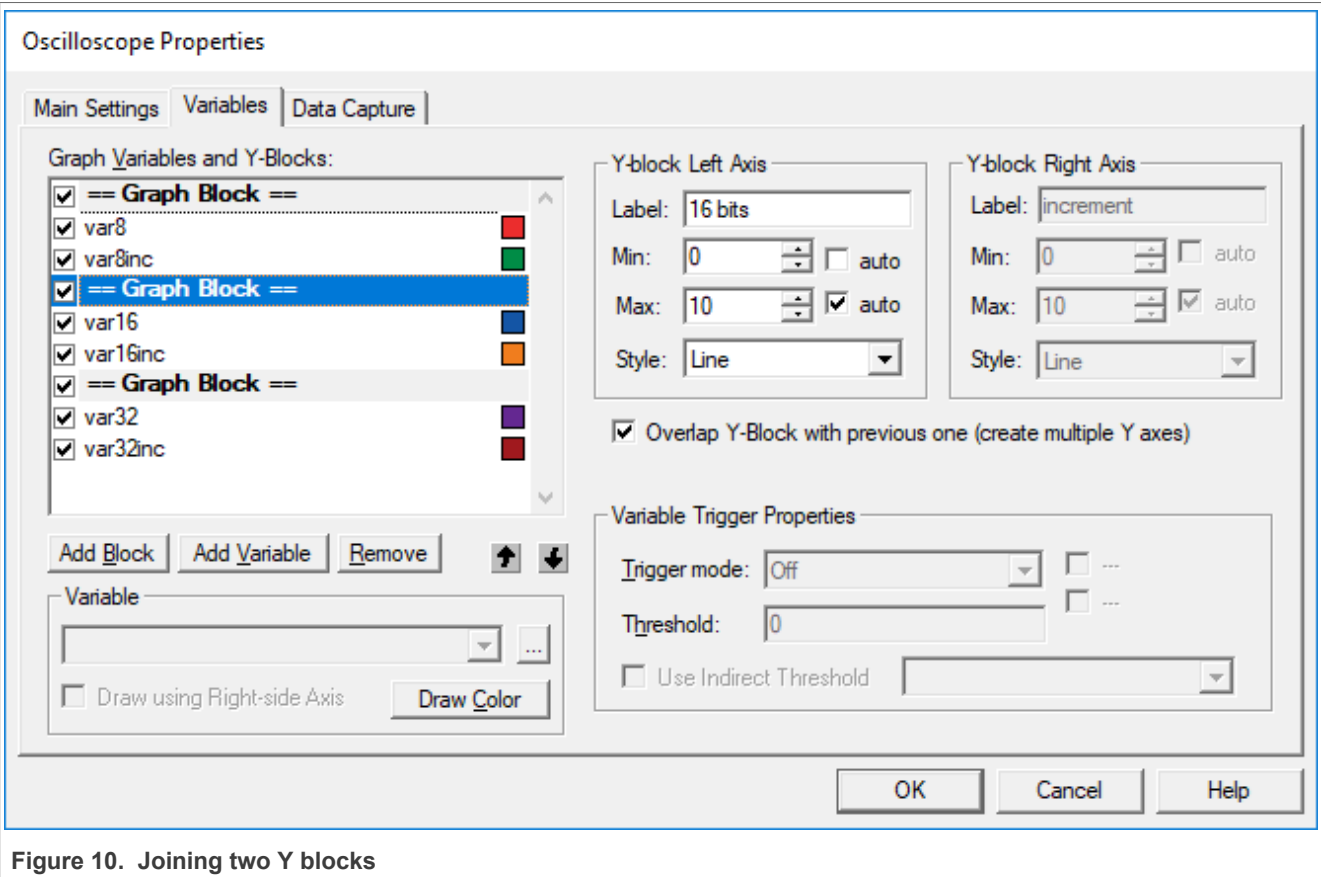
**Figure 10. Joining two Y blocks**

The resulting chart is shown in the following figure.
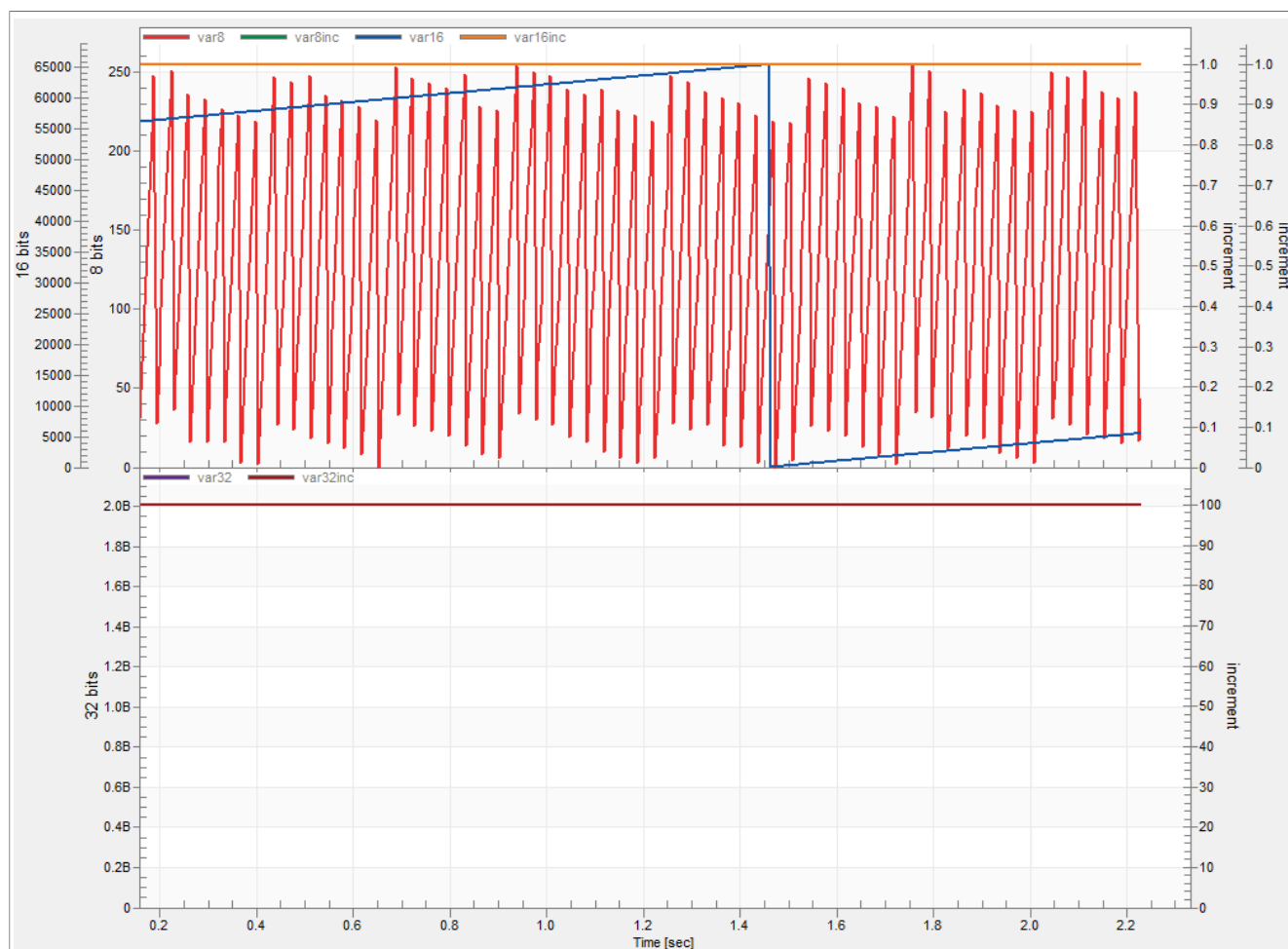
**Figure 11.  Joining two Y blocks in the graph**

### 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 Oscilloscope periodically reads variable values and plots them in real time, the Recorder runs on the target board, reads application variables, and sends them to the FreeMASTER tool in a burst mode. 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.

**Figure 12. Recorder Properties window—Main page**

The Main page contains the following user configuration items:

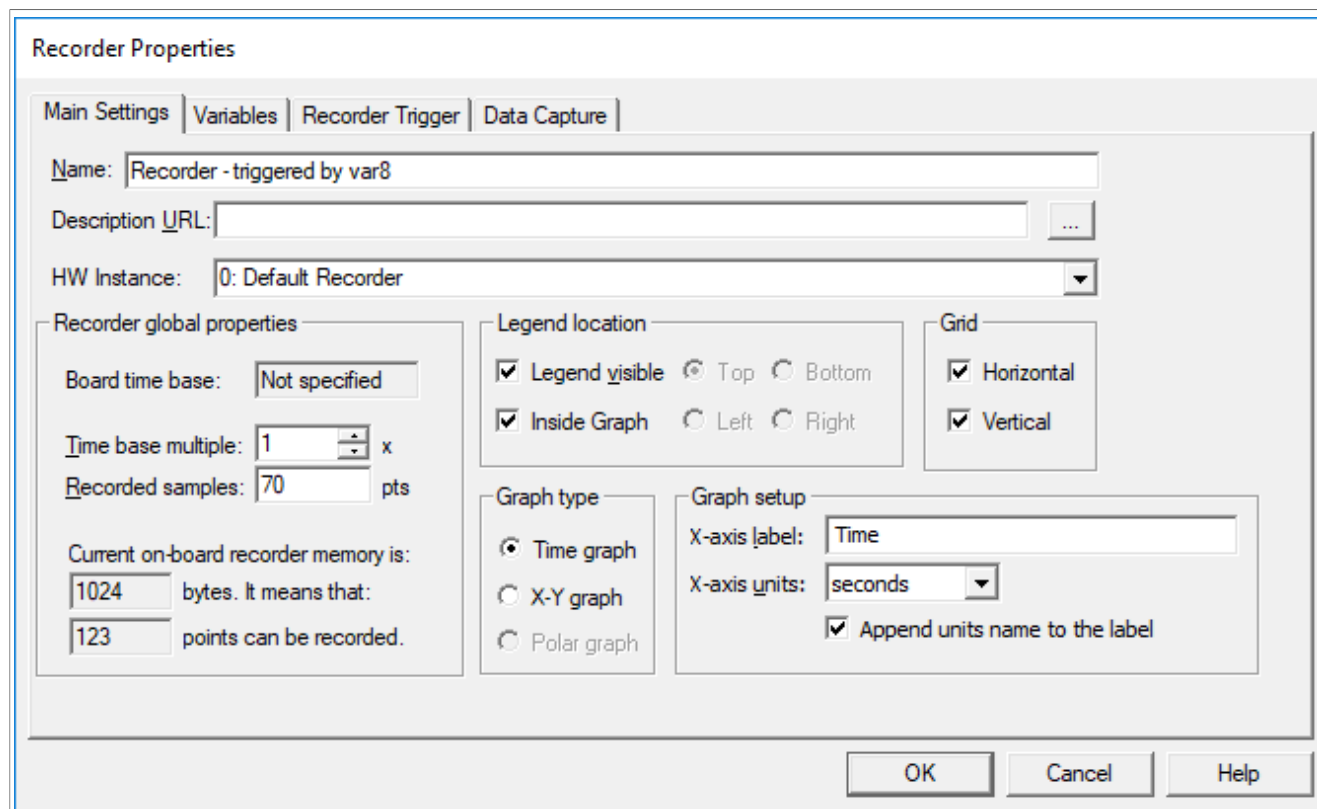- Name = the name of the Recorder item that is displayed in the Project Tree.
- Description URL = specify the document's URL or local path to a file to be shown in the Detail View pane under the current item help tab.
- Hardware Instance = identifier of the Recorder instance defined in the target microcontroller application. The application may define multiple recorders and let each of them be sampled in different functions. The application also assigns a name to each recorder instance, for example: Periodic-Timer Recorder, PWM-Interrupt Recorder, ADC Sampler Recorder, and so on. Multiple recorders of different hardware instances may run simultaneously in FreeMASTER.
- Recorder properties = common properties for all Recorder variables
  - Board timebase = a sampling period preset by the board application. Use the FMSTR_REC_TIMEBASE configuration option in the target microcontroller application to define the recording period. Leave this option at 0 when the recording base changes or cannot be specified.
  - timebase multiple = sets an integer multiple of the timebase to extend the sampling period used for the Recorder operation.
  - Recorded samples = the number of samples buffered for one recorded subset.
  - onboard recorder memory = displays the amount of onboard 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 memory is calculated and displayed. The Recorded samples' value set should be lower than this result.
- Legend location = sets the visibility and location of the chart legend.
- Grid = chooses the horizontal and/or vertical grid lines to be displayed in the chart.
- Graph type = selects the mode of Recorder operation.
  - Time graph = a variable (values versus time) 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.
– X-axis max = sets the upper limit value of the X-axis.

The Variables page of the Recorder properties dialog (shown in the following figure) looks exactly the same as the appropriate page of the Oscilloscope properties dialog. For more information about how to add variables to the Recorder chart and how to set up the chart itself, see Section 4.1.1.2.
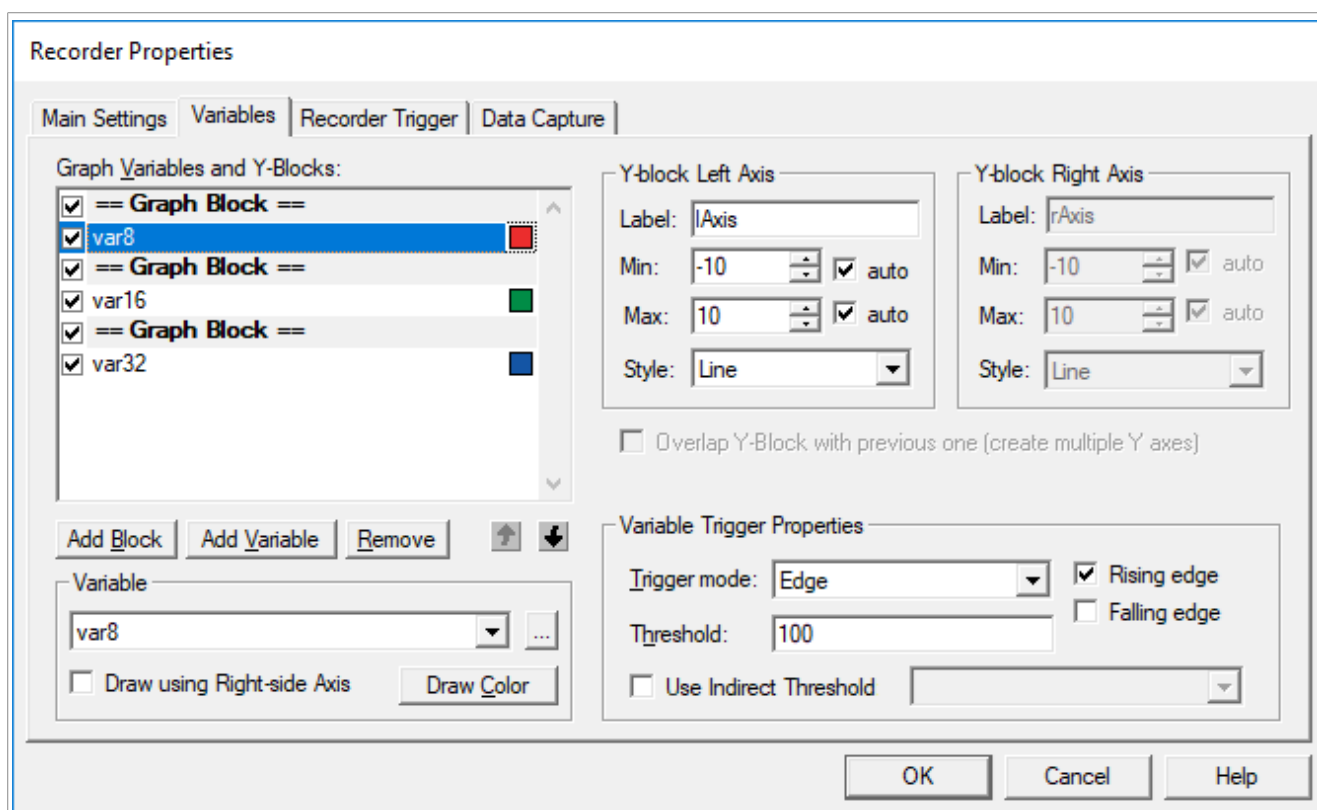


**Figure 13. Recorder Properties window—Variables page**

In addition to the general variable list configuration, the Recorder setup is extended by the Variable Trigger Properties options. One or more variables in the list may be configured to act as so-called "trigger" variables. When such a trigger variable value exceeds a defined threshold, the Recorder is set to stop recording after a defined period of time elapses. During this "stopping" period, the Recorder records the remaining data points which follow the trigger event. It also preserves the defined number of points sampled before the trigger event – so-called "pre-trigger" samples.

The triggering is configured in the Recorder Trigger tab.
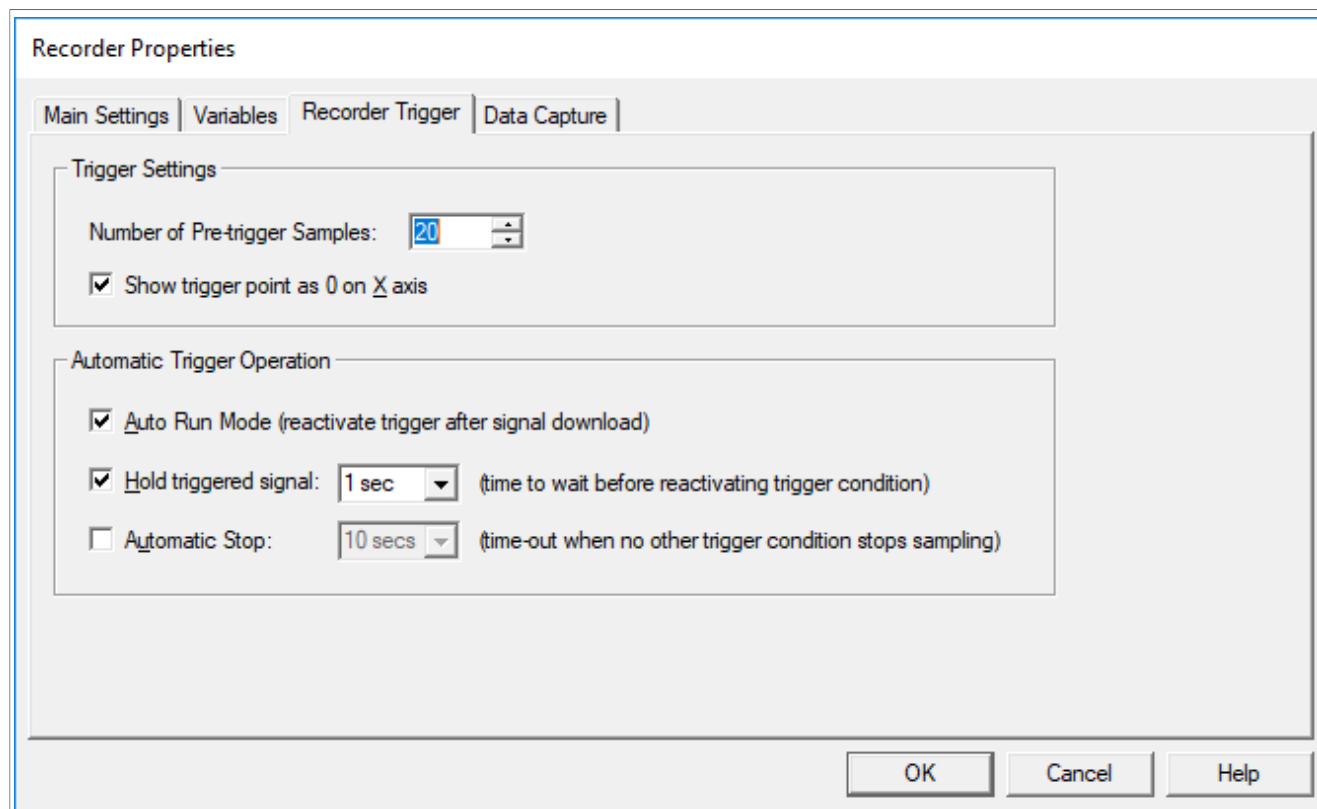
Figure 14. Recorder Properties window—Trigger page

- Number of Pre-trigger Samples = specify the number of samples to save and display before the trigger event.
- Show trigger point as 0 on X axis = when checked, the X-axis origin is put to the time of the trigger event.
- Automatic Trigger Operation = specify the conditions to reactivate the trigger for repeated recording.
  - Auto run mode = enables 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 = tick this box and specify how long to wait after a signal is displayed in the chart before reactivating the trigger.
  - Automatic Stop = tick this box and specify the maximum time period for detecting the trigger event. If there is no trigger event detected within the specified time, the sampling is unconditionally stopped and the actual buffer data is downloaded.

### 4.1.1.4 Array Viewer

As noted in the previous section, the Recorder requires quite sophisticated protocol logic and it must be supported by a driver running as a part of the target application. This rules out using the Recorder when FreeMASTER connects "passively" over JTAG or BDM interfaces, which only support direct memory read and write operations.

The Array Viewers may be viewed as a simpler and lighter replacement for the Recorder. The Array Viewer can read one or more arrays from the target memory and display the values in a graph. Another scalar variable is used as a trigger, causing the graph to refresh anytime it changes the value. FreeMASTER can optionally clear the trigger variable automatically after displaying the graph, providing an acknowledge to the target that it may prepare new data.

The primary benefit of the Array Viewer is that it only needs the Read Memory feature (Write in case of the trigger acknowledge). This is the reason why it can operate over any communication interface, including the direct-access JTAG or BDM interfaces. In many situations, displaying the whole arrays of values present in the target memory is more natural than using the Recorder which is oriented more toward capturing scalar variables' transitions in time.
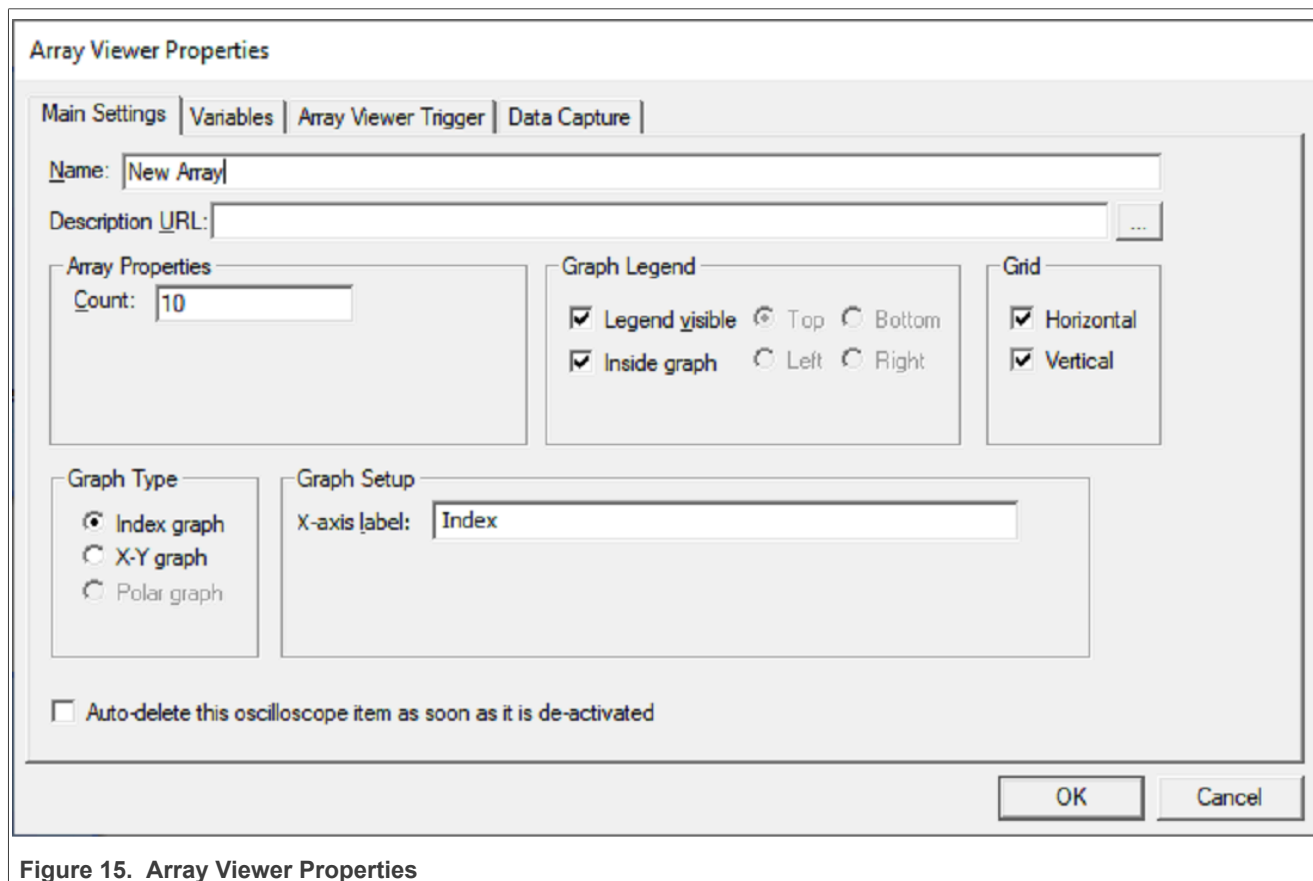


**Figure 15.  Array Viewer Properties**

The "Main Settings" tab defines the general Array Viewer parameters:

• **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.
• **Array Length** = specify the number of elements that are going to be fetched from each array (viewed arrays are listed in the Variables tab).
• **Graph Legend, Grid, Type,** and **Setup** = define the graph style in the same way as described for the Recorder in the previous section.

The **Variables** tab of the Array Viewer properties dialog is the same as the one used with the Recorder and Oscilloscope. It is used to define a list of arrays which are to be visualized. FreeMASTER does not use any special flag or option to identify a target variable as an array base. The Array Viewer will simply offer all variables which look like array elements (whose **name** ends with square brackets) to become elements that represent the whole array. For example, a variable named **arr16[0]** will be available to be used as a representing base element of the **arr16** array. The Array Viewer will use an address of such a representing element as a memory base to read the whole array. The length of the array is taken from the **Array Length** option specified in the *Main Settings* tab. The **size**, **type, format, bit-mask,** and post-processing **real-time transformation** formula defined in the representing variable definition will be also applied to all elements read from the array before putting the values into the resulting graph.
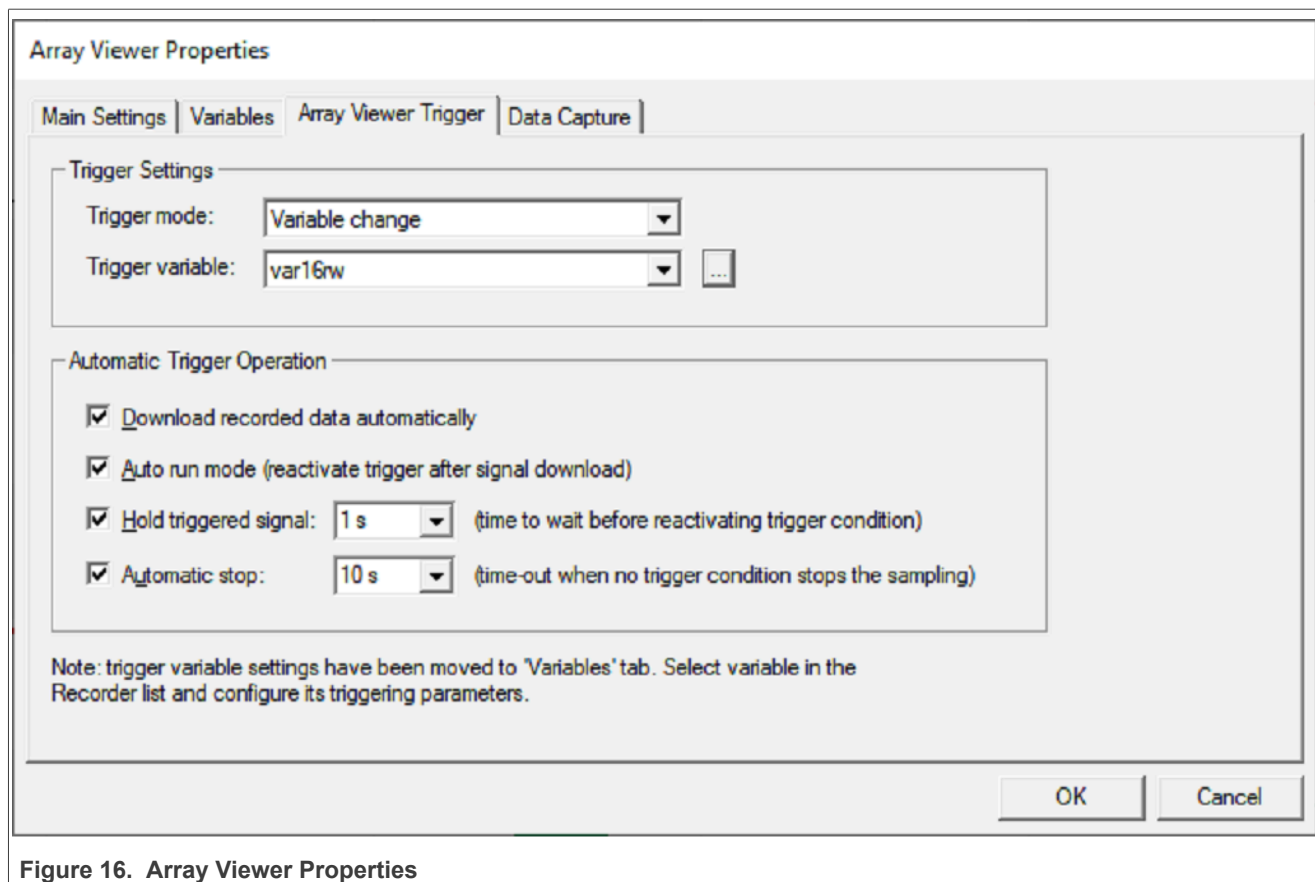
Figure 16.  Array Viewer Properties

The **Array Viewer Trigger** tab defines the viewer's trigger variable and other options, again quite similar to the Recorder object:

- **Trigger mode =** select the trigger condition which will cause the Array Viewer to activate, read all arrays, and display the resulting graph. This may be one of the following:
  - **Off =** no trigger is used. Only the manual button press or the *Automatic stop* time period expiration will trigger the viewer to show data.
  - **Variable change =** monitor the variable selected below and activate the viewer anytime the variable value changes.
  - **Variable change with acknowledge =** monitor the variable for any changes. When the viewer finishes the graph update, the variable is set to 0 by FreeMASTER.
- **Trigger variable =** the variable used as a trigger, depending on the *Trigger mode* selected.
- **Download recorded data automatically =** this is enabled in most cases. It causes the graph data to be downloaded and displayed automatically after a trigger is detected. When unchecked, the operation will wait for a button press.
- **Auto run mode =** when enabled, the trigger condition will be automatically re-activated after the graph is updated. This will cause the graph to update every time when a trigger variable changes again.
- **Hold triggered signal =** the minimum time the graph remains on the screen before the trigger is re-activated in the *Auto-run mode*.
- **Automatic stop =** thetimeperiod after which the viewer activates and updates the graph even without any trigger occurrences.

The **Data Capture** tab enables you to store the array data to be saved in a file or a sequence of files, again in an almost identical way to what the Recorder does.

**User guide**      **Rev. 4.8 — 18 September 2024**      Document feedback

**24 / 142**

### 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 (Section 5.4), all Project Tree items 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 Oscilloscope 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 Section 5.4.

The control page for the demo application used as an example is shown in the below figure. Despite its name, there are very few "control" features utilized in this simple application. Just three buttons setting a value of the var16inc variable which controls how much the var16 variable increments. The page also demonstrates the JavaScript scripting technique to display variable values directly in the HTML-coded page and a more advanced gauge object. For more details about the HTML coding and scripting, see Section 6.2.
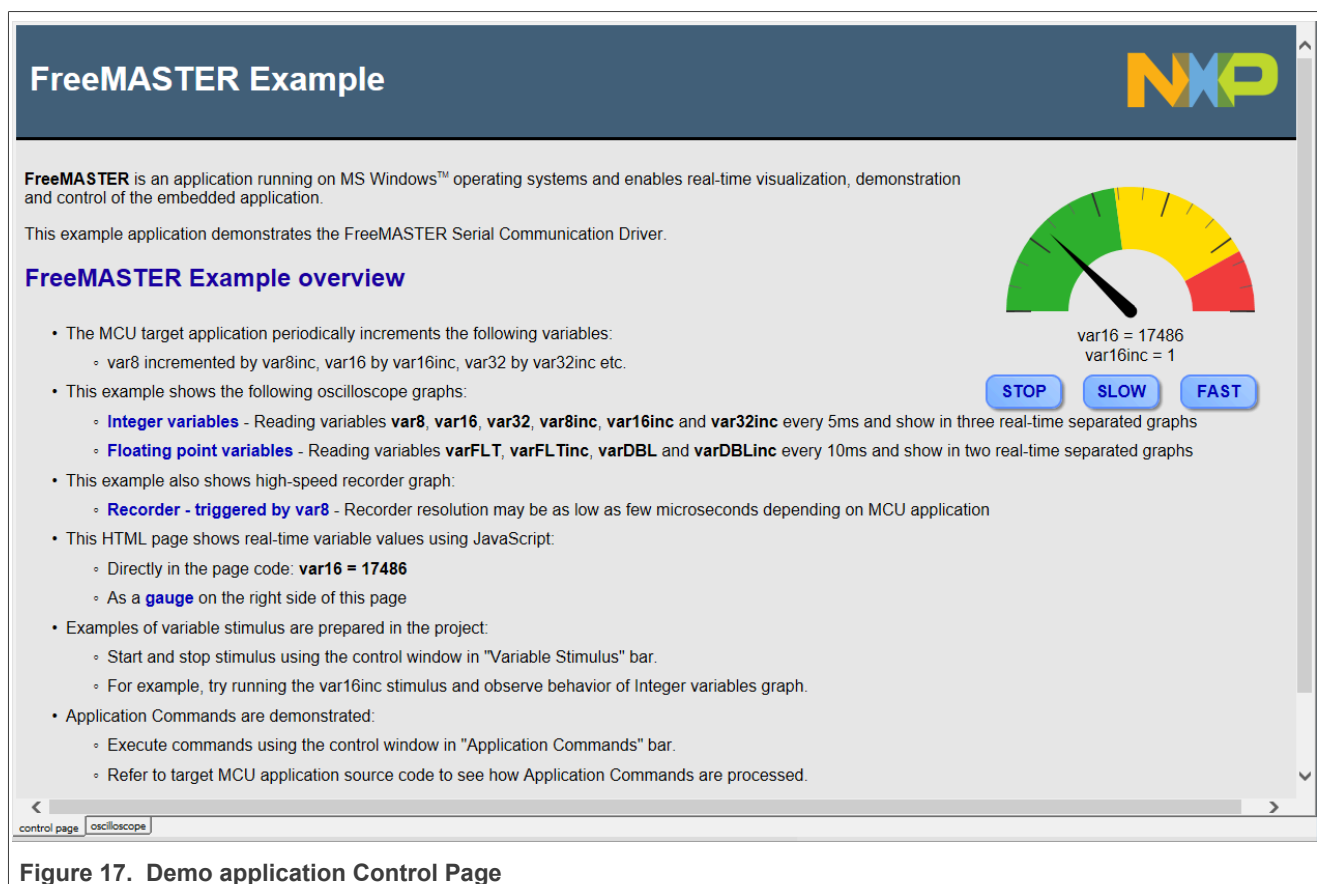


**Figure 17. Demo application Control Page**

By default, the demonstration control page uses a backward-compatible option with Internet Explorer and ActiveX interfaces, so the project works in all FreeMASTER versions (2.x and 3.x). FreeMASTER Version 3 enables modern JavaScript techniques to be used with the asynchronous JSON-RPC interface in pages

rendered by the Chromium or Microsoft Edge WebView2 web browser engines. The same pages can be used also in stand-alone Chrome or Edge browsers with an extra benefit of printing, editing, and debugging of the script code.

Using the Chromium or Edge rendering with the JSON-RPC interface is the recommended option when designing new FreeMASTER control pages. Such pages also operate well in a standalone Chrome browser connected to the FreeMASTER Lite service.

Older control page designs based on the obsolete Internet Explorer view and ActiveX technology may be migrated to Microsoft Edge view running inside the FreeMASTER application window. See the Internet Explorer migration guide in Section 6.10.

A few more sophisticated Control Page screenshots are shown below. The applications shown here use some third-party instrumentation components, inserted into the HTML code as embedded ActiveX objects.

**Figure 18. HTML control page examples**
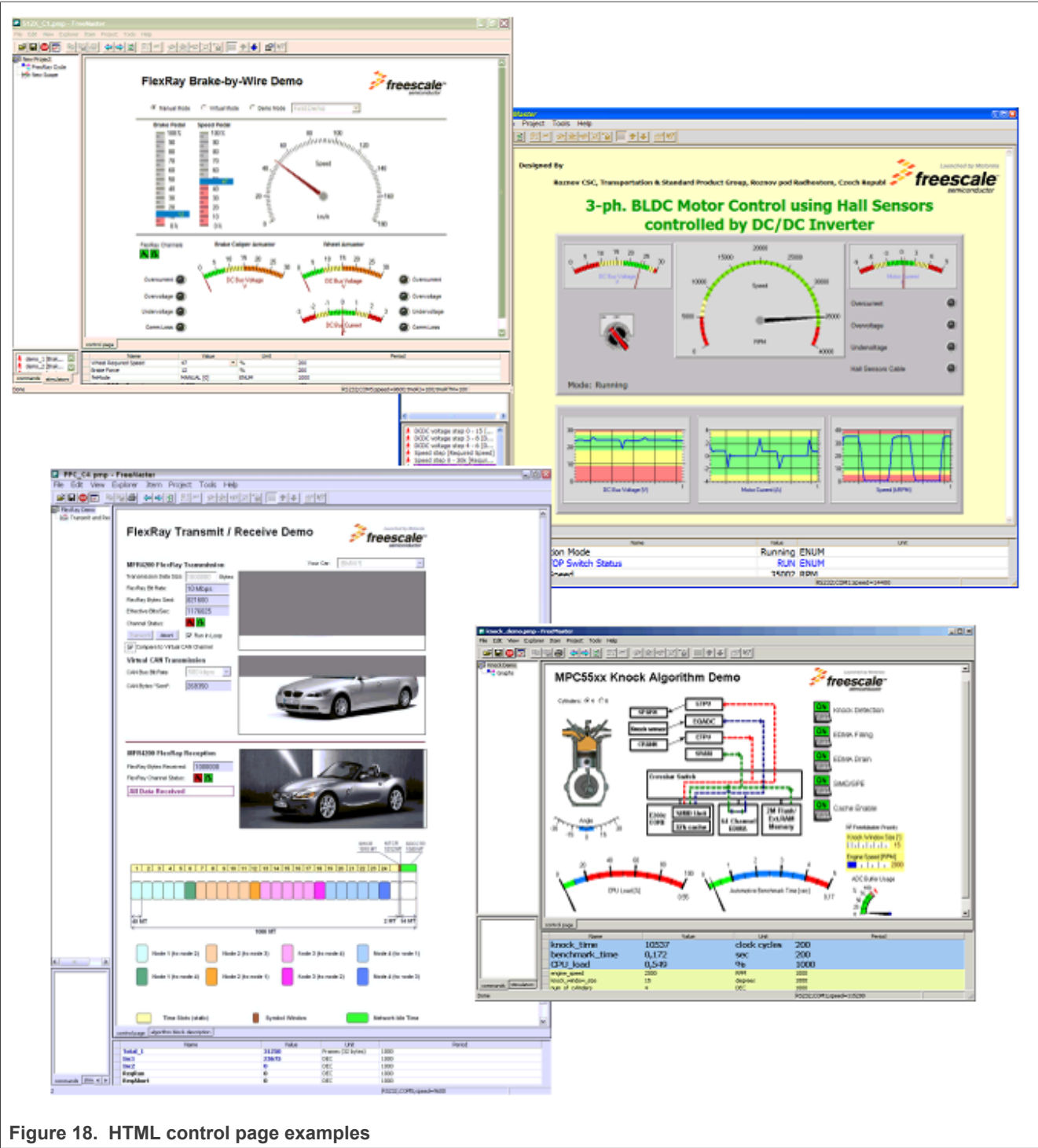
## 4.1.2.2 Algorithm Block Description

The Algorithm Block Description page in the Detail View pane contains 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 Section 4.1.1.1. This page is displayed when it is defined and when you select the appropriate block item or any of its child Oscilloscope or Recorder items.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**27 / 142**

As the standard HTML page, this page can also contain the scripts and other controls, but it is not a common practice. All control-specific features should be concentrated in the Control Page.

### 4.1.2.3 Current item help

The Current item help tab contains an HTML page which describes the Oscilloscope or Recorder selected. This page should contain such information as definitions or use instructions. It is specified as the description URL. The use of scripting or other control features is also not recommended on this page.

### 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 Section 4.1.1.2. Similarly, the Recorder page contains the chart created from the recorded data, as described in Section 4.1.1.3.

### 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 Formatting bar can be used to change the graphical look of a variable, including font type and size, foreground and background color, and alignment. See Section 4.6.2 for details.

The read-only variables can only be monitored. Variables with changes allowed can be altered from the Watch-Grid pane. For details about variables, see Section 4.2.

## 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, from Variables tab in the Oscilloscope Properties dialog).
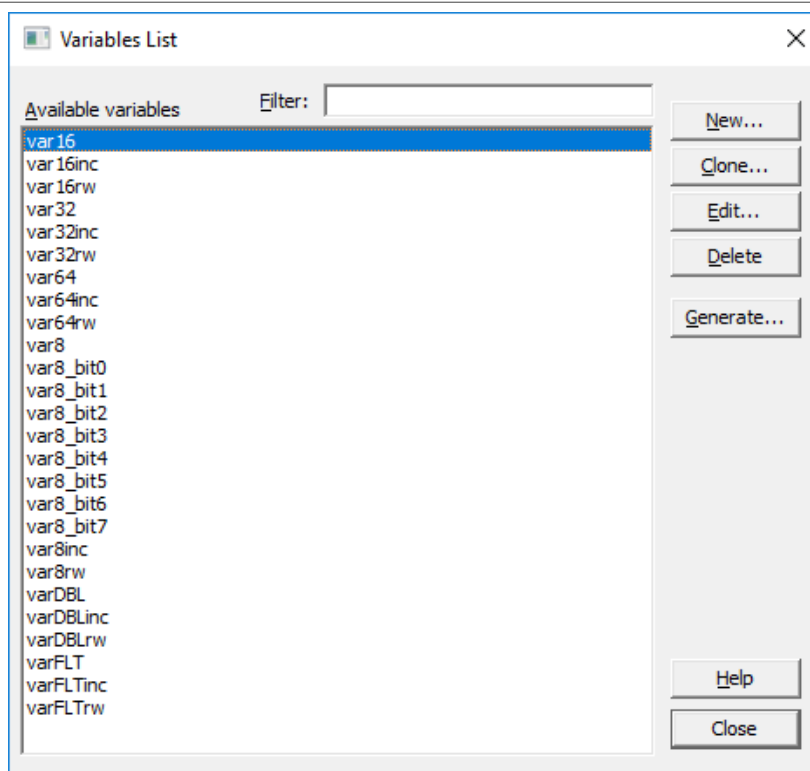
**Figure 19. Variables list dialog box**

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 Section 4.2.1. The loading of symbol files is described in Section 5.2.

*Variable Settings*

The Variable definition dialog has two tabs: Definition, shown in Figure 20, and Modifying, shown in Figure 21.
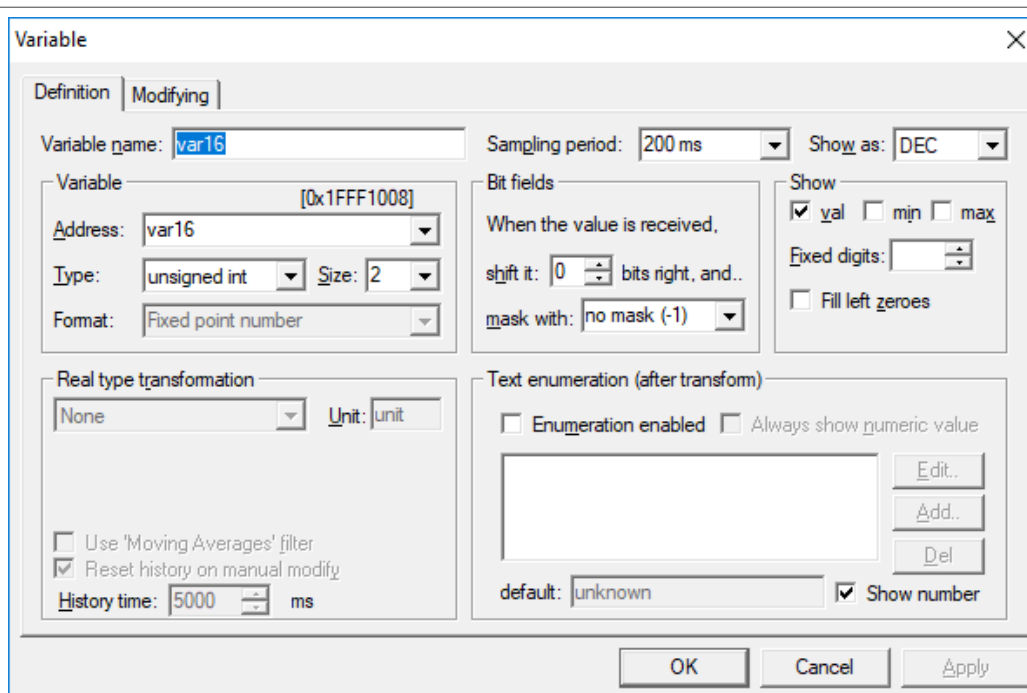
**Figure 20. Variable dialog box—Definition tab**

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 target application if a TSA feature is enabled or from embedded application executable file in the ELF format. You can also specify expressions in the address field which involve mathematical operations, sizeof() operator, and array-dereferencing operator [].
  - Type = select the variable type, as it is defined in the target application (unsigned fixed point, signed fixed point, floating point IEEE, fractional, unsigned fractional, or string).
  - Size = specify the size of the variable, as it is defined in the target application.
  - Format = extended binary format parameters must be provided to properly decode fractional types.
- 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 = tick 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 parameters 'a' and 'b', fill in the two coordinate points: (x1, y1) and (x2, y2). You can again specify the numeric values or variable names as the parameter values.
    hyp: d/(ax+b) + c: specify the parameters 'a', 'b', 'c', and 'd' of a hyperbolic transformation function.
  - Unit = the name of the 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 enables 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.

FMSTERUG
All information provided in this document is subject to legal disclaimers.
© 2024 NXP B.V. All rights reserved.

User guide
**Rev. 4.8 — 18 September 2024**
Document feedback
**31 / 142**
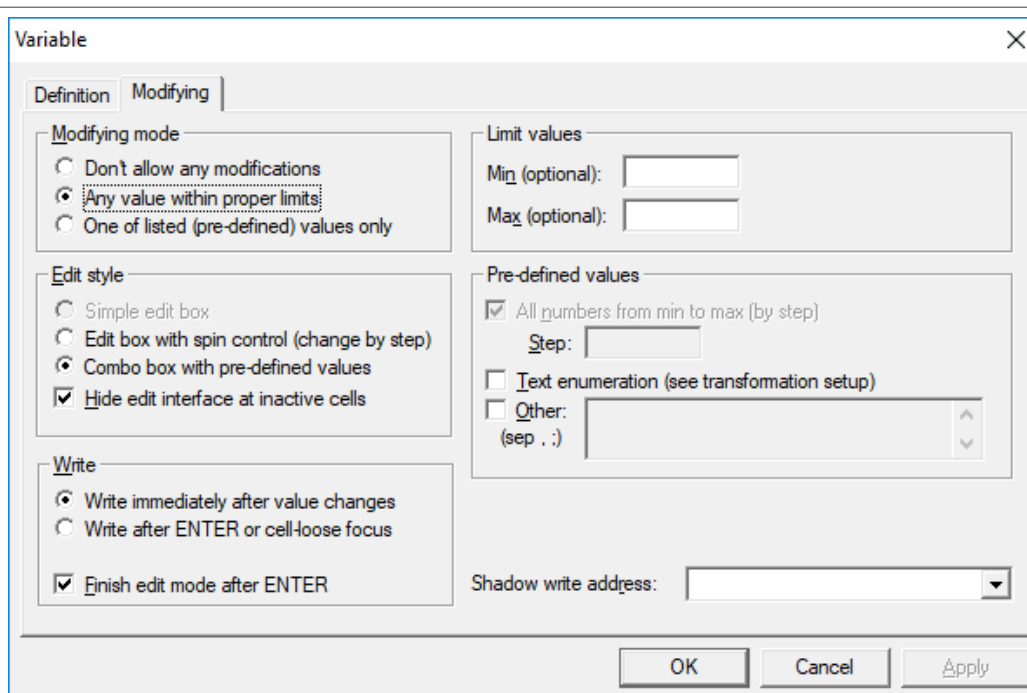
**Figure 21. Variable box—Modifying tab**

- 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 values. The value you enter into the watch window is then validated with the specified limits.
  - One of listed values only = when you specify 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 from 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 19) opens the Generate variables dialog box shown in the following figure.

In this dialog, you can automatically generate the variable objects for the variable objects for one or more symbols loaded from a connected embedded application (TSA), from its executable file (ELF), or a linker MAP file (see Section 5.2 for more information about symbol tables).
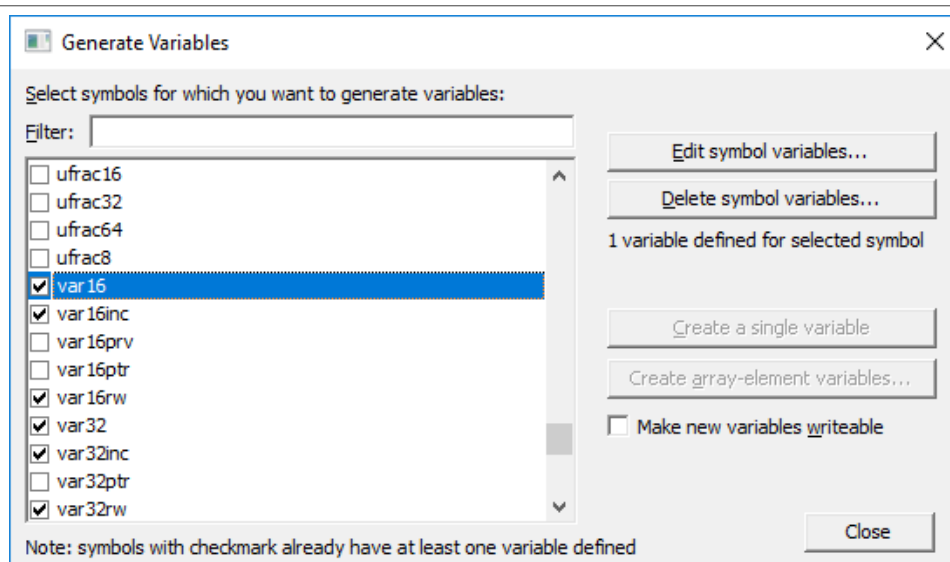


**Figure 22. Generating variables**

The list in the dialog shows all the symbols available in the project as they are read from the current symbol file. The symbols, for which the variables are already defined, are marked with a checkmark. Typically, you will want to create variables for symbols without the checkmark.

- Edit symbol variable = click this button to edit a variable bound to the selected symbol (if there is one).
- Delete symbol variable = click this button to delete a variable bound to the selected symbol(s).
- Create a single variable = for all selected symbols, this option generates (creates) a new variable with the same name as the symbol and with proper address, type, and size settings. After creation, you can use the Edit symbol variable to see and change parameters of newly created variable objects.
- Generate array-element variables… = enables you to generate a set of variables encapsulating individual elements of an array.

### 4.2.1.1 Generating array-element variables

There are two symbols representing arrays in the FreeMASTER symbol list. One symbol represents the whole array while the other represents the first (or any other) array element.

The demo microcontroller application declares an array of short integers as follows:

```
short int arr16[10];
```

In FreeMASTER, the `arr16` symbol represents the 20 bytes of memory area used by the array as a whole. In addition to that, a symbol named `arr16[0]` represents the first 2-byte element of the array.
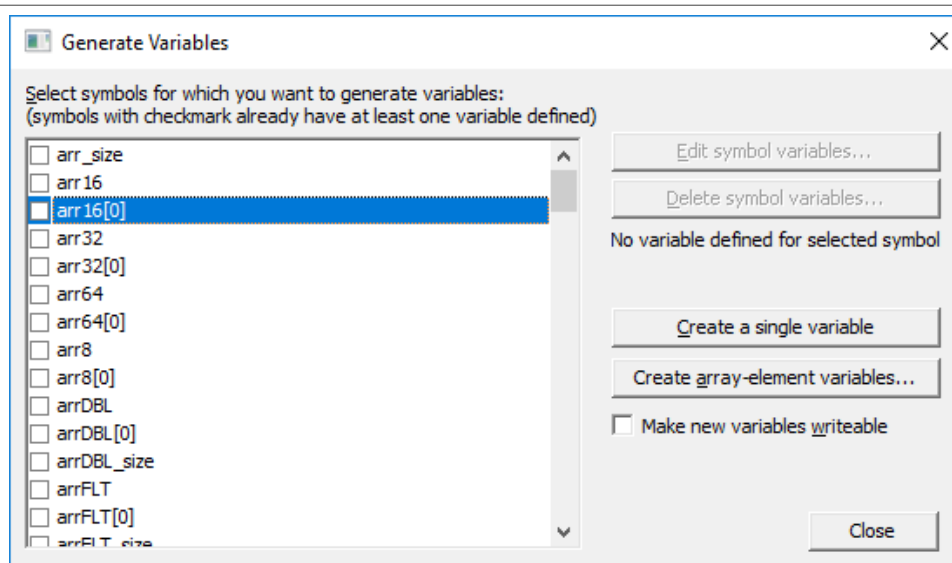
**Figure 23. Generating array-element variables**

When defining a single variable, `arr16[0]` and any other indexed element (like `arr16[1]`, `arr16[2]`, and so on) can be used as a variable address. Mass-creating several array element variables may be automated in the *Generate Variables* dialog described in the previous section. Select an array element symbol (for example, `arr16[0]`) and click the *Create array-element variables…* button.

A simple dialog opens and enables to specify a range of array indexes for which the variables are to be generated.
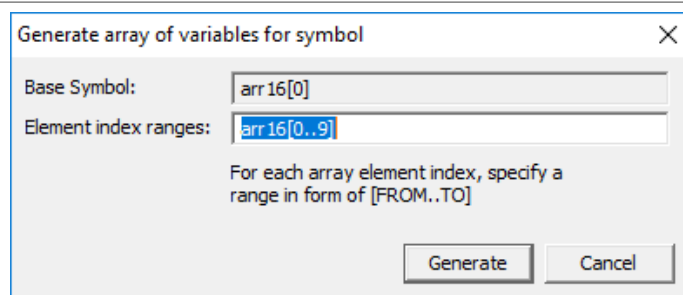


**Figure 24. Generating array of variables for symbol**

## 4.3 Commands

The list of Application Commands defined in the project can be opened by selecting the Project / Commands menu and it 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.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**34 / 142**
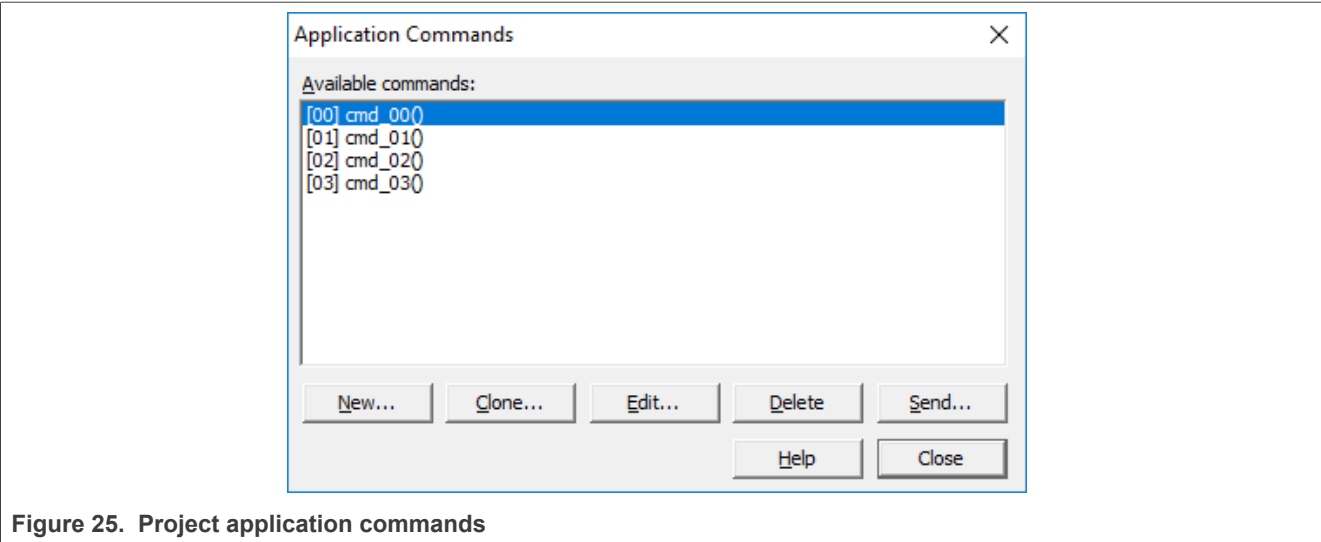
**Figure 25.  Project application commands**

In the Sending application command dialog box (which shows up 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 26. For each argument, you can define the help message, which is displayed in this dialog when typing the argument value, as shown in Figure 27.
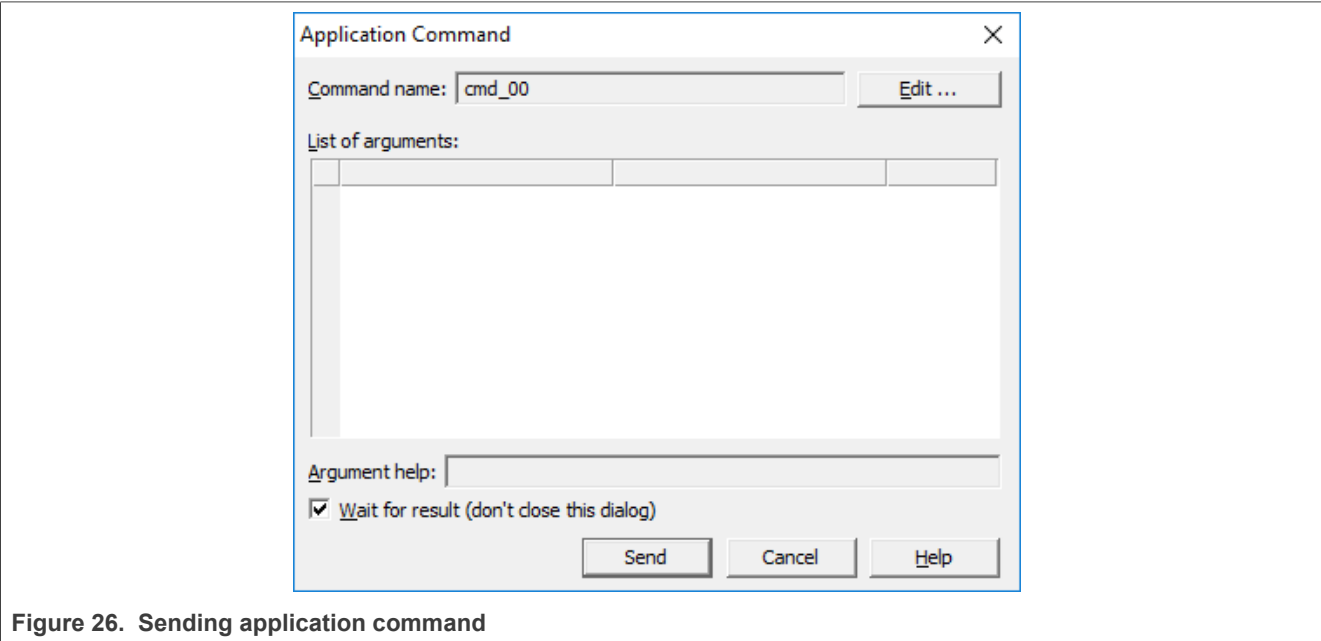


**Figure 26.  Sending application command**

If you want to wait for data to be returned from the board (a command result) without closing the dialog, tick 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 the three pages of the dialog is shown in the following figure.
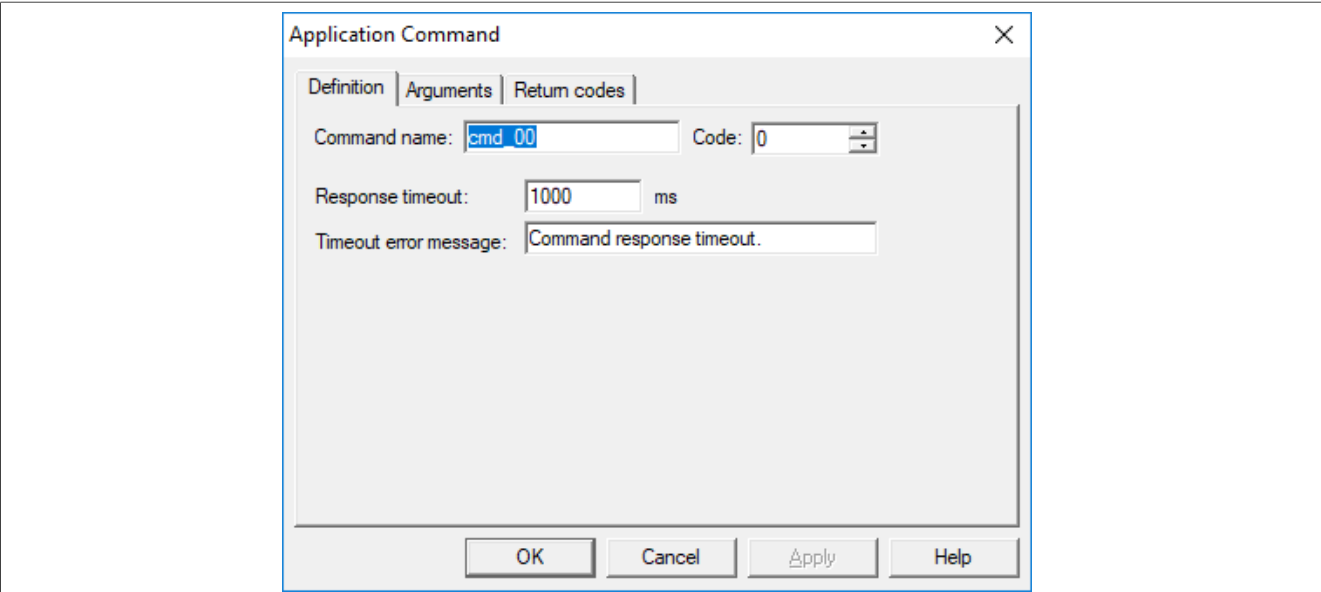
**Figure 27.  Application Command window—Definition tab**

In the Definition tab, enter the Command name used in the project and specify the Code one-byte command 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 a response from the board application. If the embedded application does not acknowledge the command and neither responds to it before this timeout occurs, the text entered into the Timeout error message field appears in the alert window.
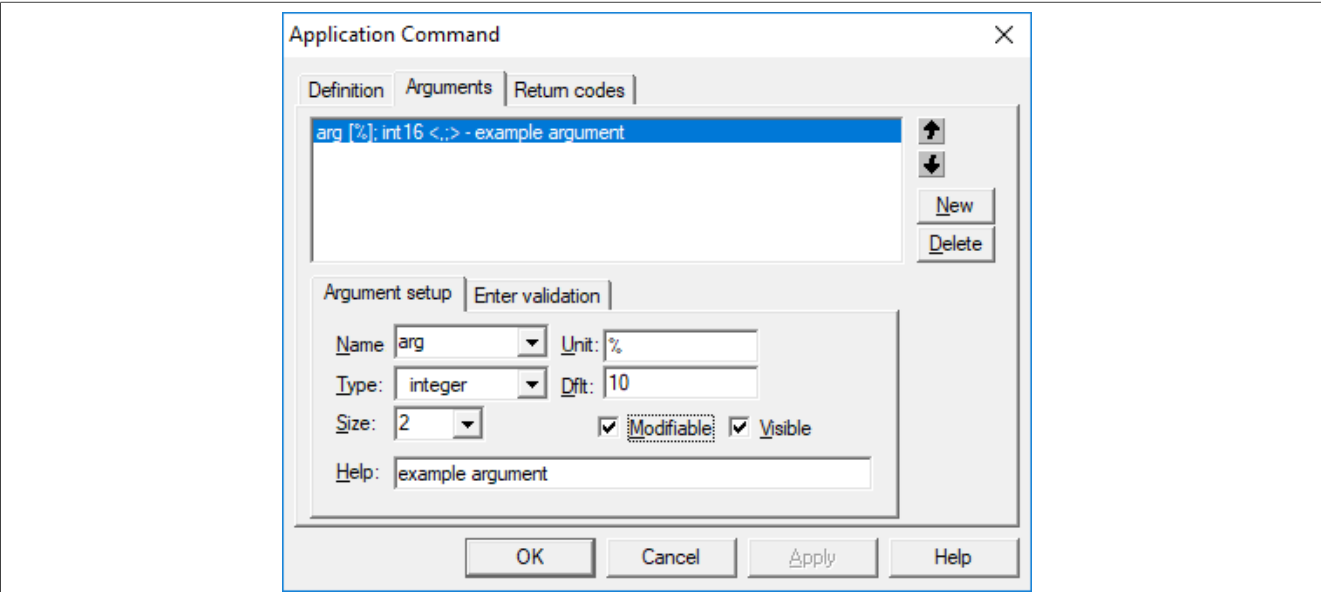


**Figure 28.  Application Command window—Arguments tab (page 1)**

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.

In 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 ticked, 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 ticked, 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.
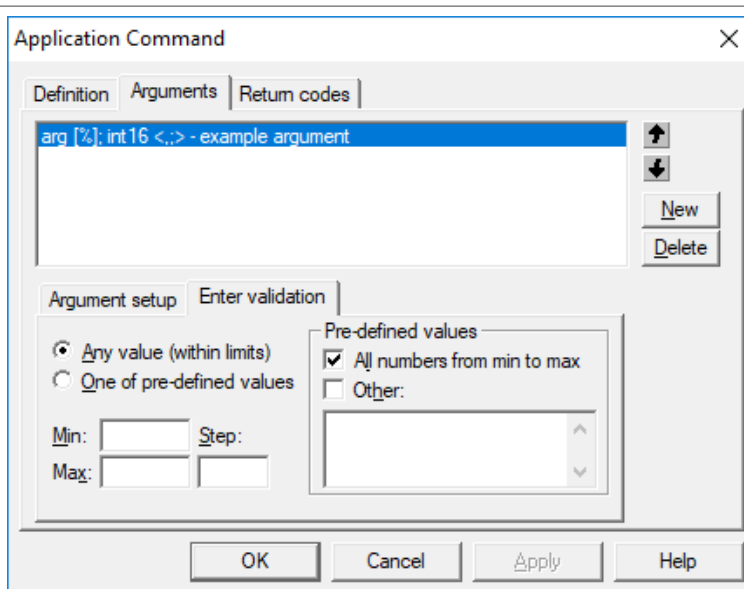


**Figure 29. Application Command window—Arguments tab (page 2)**

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 ticked, all numbers between the Min and Max limits (incremented by Step) are considered to be valid for the argument value.
  - Other = tick this box and specify the list of other values (separated by a comma) that are valid as the argument value.

 Document feedback

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 this procedure 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. Then it appears in the message dialog, together with the text of the message.
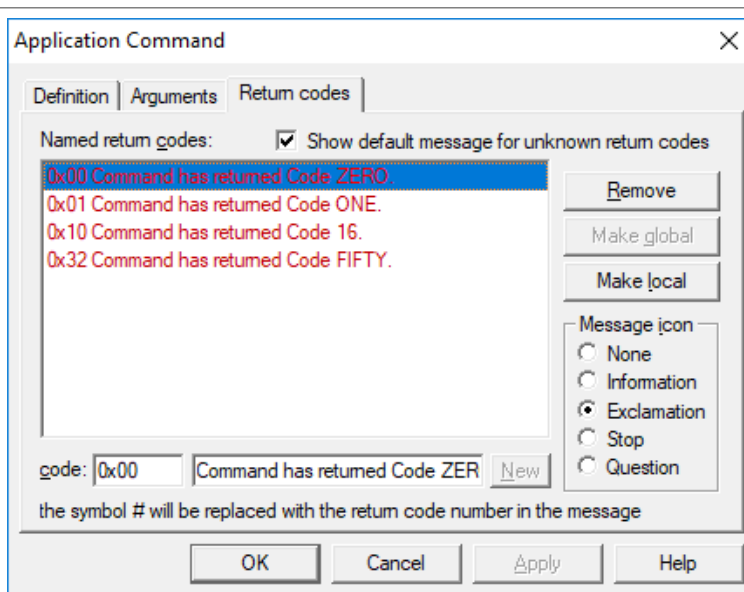


**Figure 30. Application Command window—Return codes tab**

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, oscilloscope, 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 <u>Figure 31</u>. 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 the 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.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**38 / 142**

- 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.
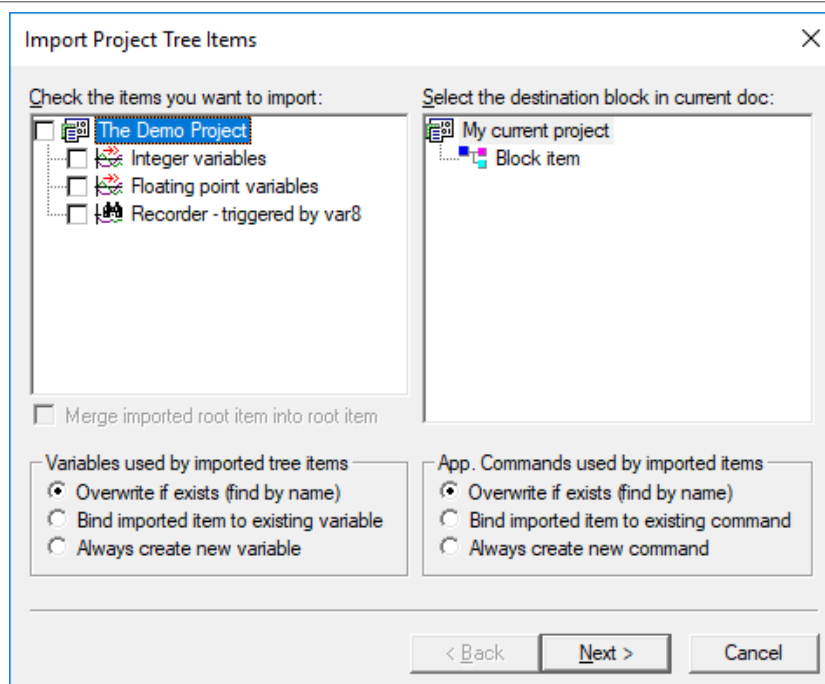


**Figure 31. Import Project Tree Items window**

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 32. The three check-box lists contain the objects found in the source project file:

- Variables = tick each variable you want to import. You don't have to import variables that are referenced from the tree items selected in the previous dialog from Figure 31; such variables are always unconditionally imported.
- App. commands = tick each application command definitions you want to import. As with the variable objects, you do not have to tick the commands that are referenced from the block tree items selected in the previous dialog.
  – Global return messages = if this box is ticked, 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 checkmark next to each variable stimulator you want to import.
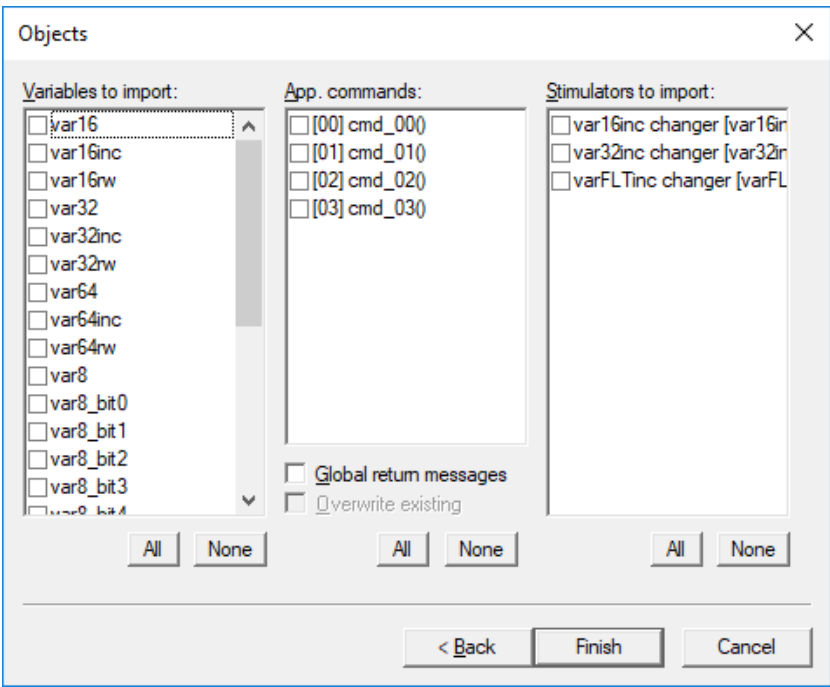
**Figure 32.  Import project objects**

## 4.5  Menu description

The following sections describe the commands available in the FreeMASTER main application menu. Note that each menu command with an icon assigned is also available in one of the toolbars for faster and more convenient access.

### 4.5.1  File menu



**Figure 33.  File menu**

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**40 / 142**
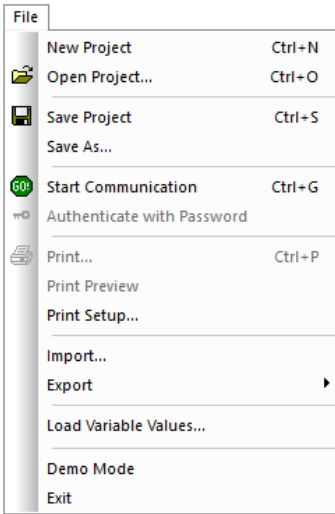
Selecting New Project creates a new empty project, while Open Project opens an existing project file, which has the *.pmp* extension.

Save Project saves the open project into the current file, while Save As enables you to specify a new filename for the current project.

Start or Stop Communication starts or pauses the communication. When started, the TSA symbol table and symbol file are automatically checked for changes. If a difference is found, the user can choose to apply new address variables.

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

Import... enables you to import selected objects (Project Tree items, variables, commands, stimulators) from an existing project file (*.pmp* or *.pmpx* file) into the current project. Importing is also possible from the FreeMASTER Lite configuration file with an *.fmcfg* extension.

Export enables to save current project settings and definition of variable objects into a FreeMASTER Lite Service configuration file. Note that when exporting to a file which already exists, the export only modifies the relevant entries in the JSON output file, while the other entries are preserved.

Load variable values enable you to load project variable values specified in a JSON file. This option is also available from the Watch-Grid context menu, where variables' values can be exported for the whole Watch-Grid or selected variables only.

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

The 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.
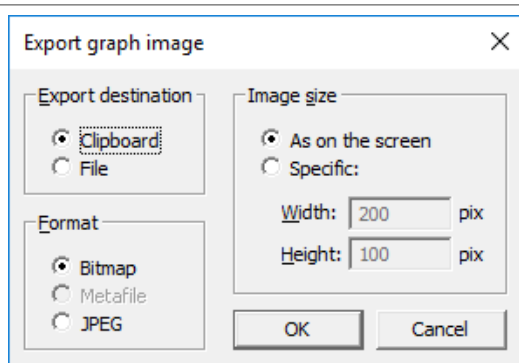


**Figure 34.  Export graph image dialog**

### 4.5.3  View menu

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

FMSTERUG

**User guide**

All information provided in this document is subject to legal disclaimers.

**Rev. 4.8 — 18 September 2024**

© 2024 NXP B.V. All rights reserved.

Document feedback

**41 / 142**
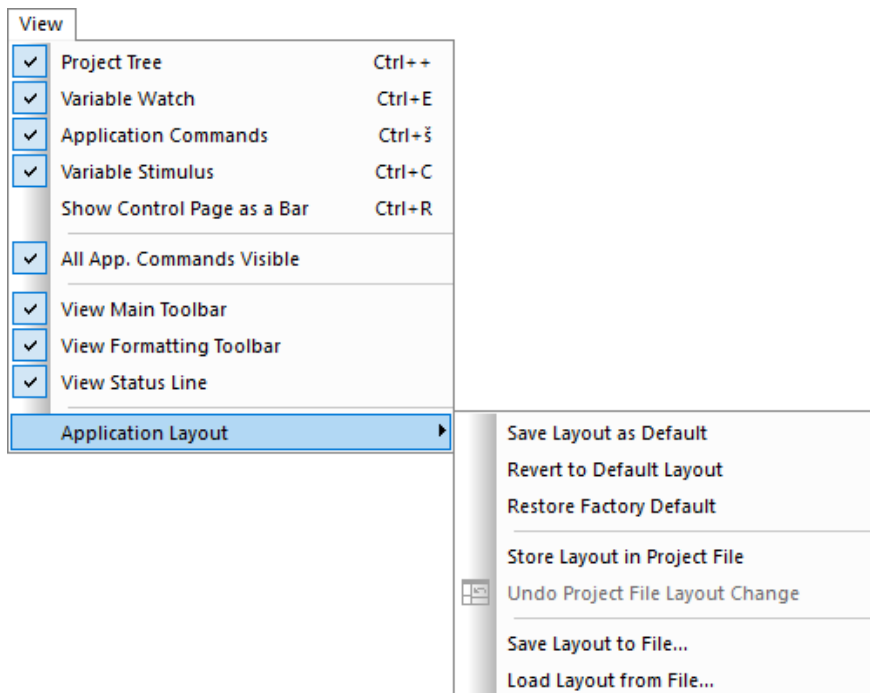
**Figure 35. View menu**

- Show Control Page as a Bar enables to undock the Control Page view from the main tabbed area in the application window and makes it a floating window, which may be optionally docked just like the Project Tree, Variable Watch, and other views.
- All App. Commands Visible makes all Application Commands defined in the project to be always displayed in the Commands view. When not ticked, only the Commands selected in the Project Block Properties is visible whenever a given block item is selected in the Project Tree.
- Application Layout enables the user to save immediate layout of bars and view panes or to restore them again.
  - Store Layout in Project File may be used to get the window layout saved to a project file whenever it is saved the next time. When such a project file with layout is loaded, the stored window layout is applied.
  - Undo Project File Layout Change is available to users who load the project with the window layout stored. This option gives users a chance to revert the restored layout back to the recently used one.
    When creating FreeMASTER projects targeted to other users, you should never assume that a certain window layout that you store to a project is going to be actually applied.

### 4.5.4 Explorer menu

The Explorer sub-menu is available when an HTML page (Control page or other) is displayed in the Detail View. The menu is the same for Internet Explorer and Chromium rendering modes.
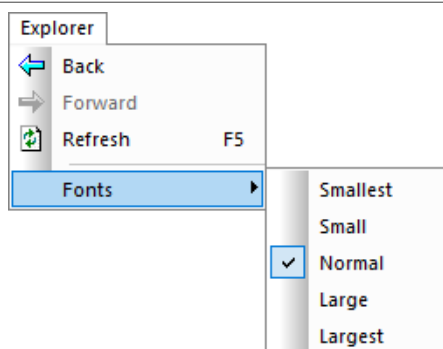
**Figure 36.  Explorer menu**

The Back, Forward, and Refresh items represent the commands for the embedded web browser window. They are used to move through previously-visited pages and refresh the page contents.

Fonts sets the font size for the current window.

### 4.5.5  Oscilloscope menu
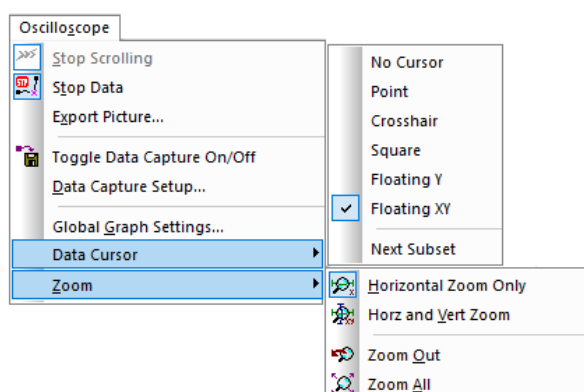


**Figure 37.  Oscilloscope menu**

- The Stop Scrolling and Stop Data commands cause 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 the 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.
- Global Graph Settings enable you to set font sizes, point size (for point-based graphs), and line width (for line-based graphs). Specified settings apply for all item types (oscilloscope, recorder, and so on).
- Data cursor enables you to select the style of data cursor to examine the chart values. The Next Subset sub-item selects the next chart line to be examined. Note that the data cursor is controlled using the arrow keys on the keyboard. Use the mouse to click any graph data point (when the cursor changes to a "hand" shape) to position the cursor.
- The Zoom sub-menu consists of zooming modes and commands. The Horizontal Zoom Only mode allows x-axis-only zooming. The Horizontal and Vertical Zoom allows free rectangle zooming.

**Figure 38.  Global Graph Settings**
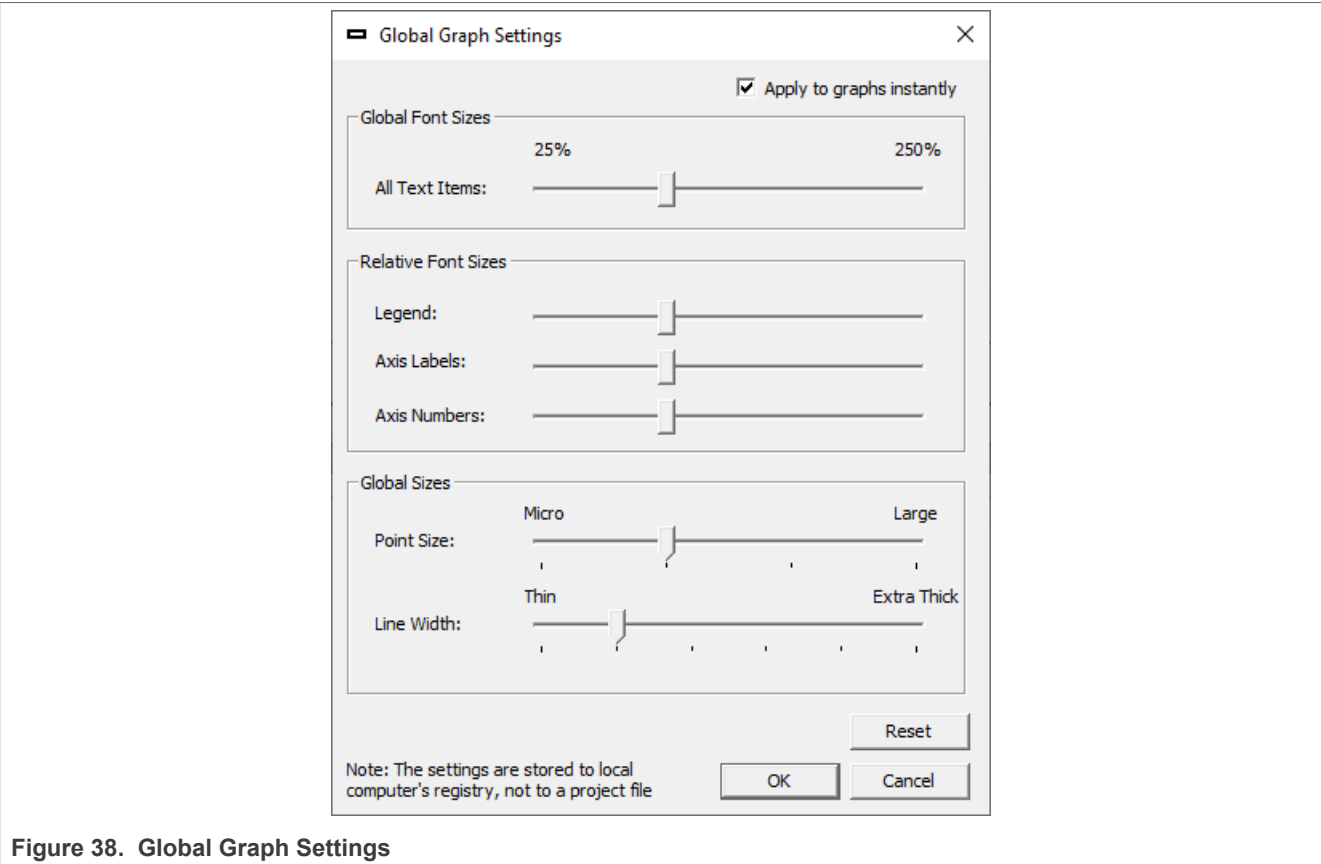
### 4.5.6  Project menu

The Project menu items are used to access project settings, define the Variable and Command objects, and manage other project resources.
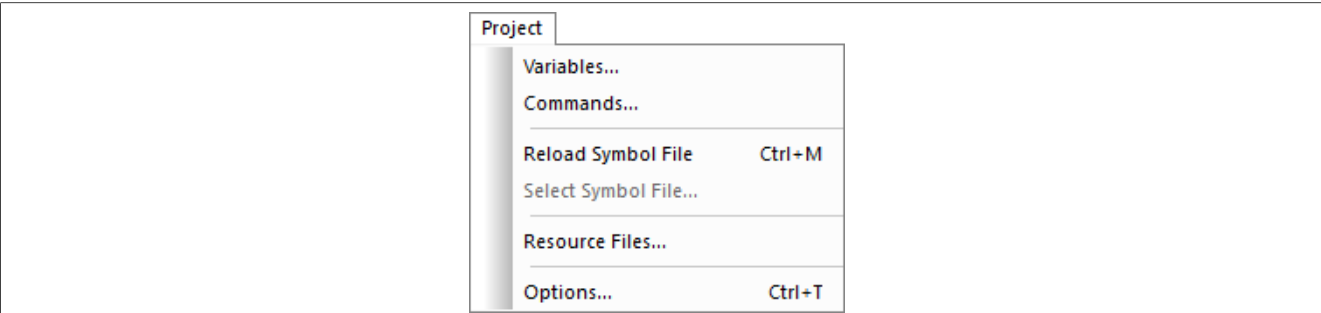


**Figure 39.  Project menu**

- Variables opens the Variables List dialog box, where you can manage all project variables, create new variables, or mass-generate variables from loaded symbols.
- Commands opens the Application Commands dialog box, where you can manage project commands.
- The Reload Map File command updates the physical addresses of the board application variables from the currently selected ELF or MAP files. Note that the TSA-loaded symbols are updated automatically whenever a board is connected.
- Select Symbol File… is available when multiple ELF or MAP files are specified in Project Options. It lets you select what file to load the symbols from.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**44 / 142**

- Resource Files... enables to review what HTML, ELF, or MAP files are referenced by the project. This may be useful before distributing the project file to other users to verify if the resources are only accessed using relative paths. Always avoid using absolute paths, because they become invalid when a project is opened on a different host computer.
- Options… opens the main dialog box with all project options and settings.

## 4.6 Toolbars

### 4.6.1 Main Toolbar

Main Toolbar allows quick access to the most commonly used menu commands.



**Figure 40.  Main Toolbar**
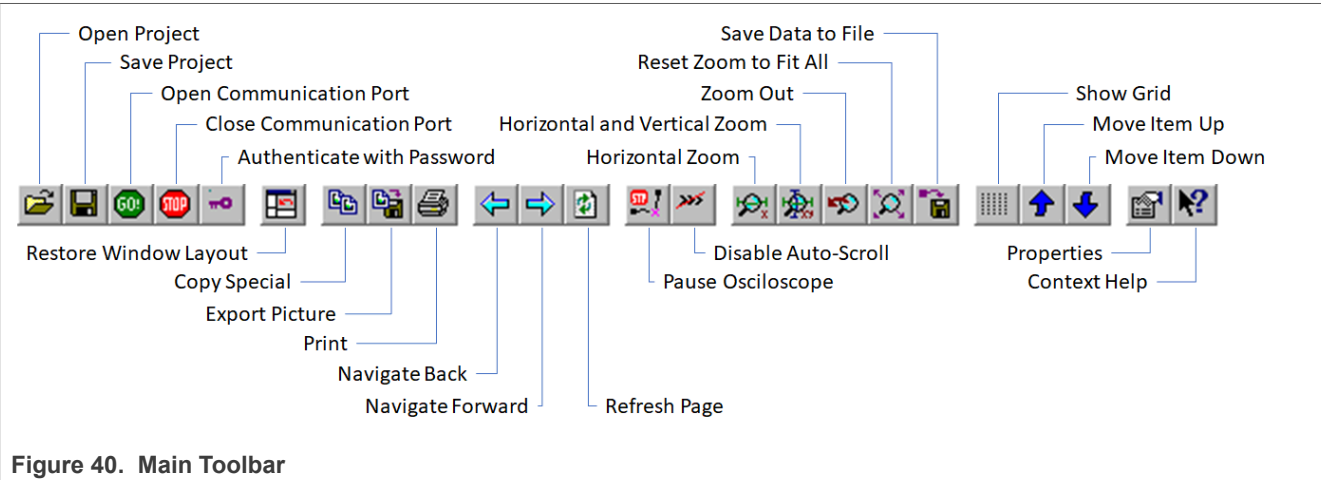
### 4.6.2 Formatting Bar

Formatting Bar is available when the Watch-Grid pane, Pipe pane, or some other view that enables to format fonts and colors has the input focus.



**Figure 41.  Formatting Bar**
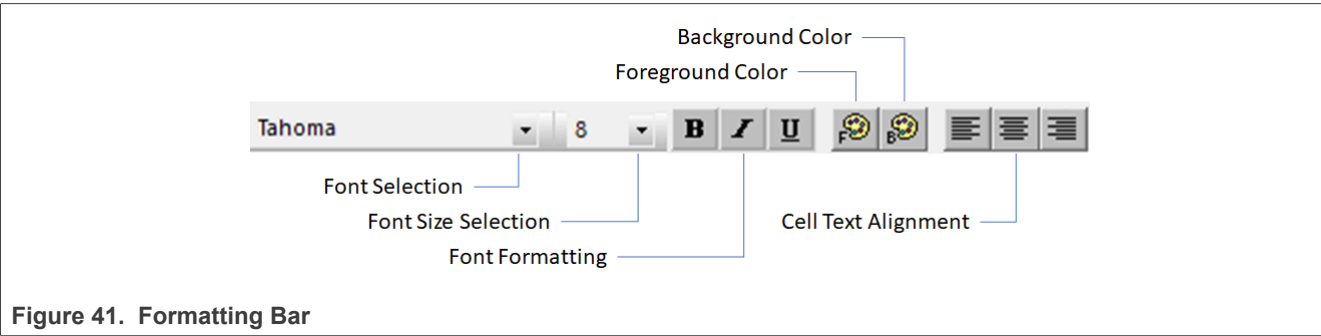
# 5  Project options

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.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**45 / 142**

## 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 42.

- Communication
  - RS-232 = the standard serial interface (3-wire RS-232 cable) is used to connect to the target application.
  - Port selects the serial port on which the board is accessed. Either specify the port system name (like "COM1") or use any word from the associated description text, as displayed next the the selection drop-down list. Some USB-to-serial cables tend to be mapped to different COM ports each time they are connected. Using the name from their description may help to simplify the connection process.
  - Speed = select or specify the communication baud rate. This speed must correspond to the embedded application settings.
  - Press the Timeouts button to set the time-out values related to the serial communication on the host PC.
  - 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 it must be registered in the system registry database. See the protocol and communication library documentation for more information. A set of plugins is installed along with the FreeMASTER application, which enables the communication over the DCOM (or other network protocol), CAN, or a BDM/JTAG debugger module.
  The drop-down list contains all plugin modules 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 click the Build button to open the module-specific configuration dialog.
  - Save settings to project file = if ticked, 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 are restored and used instead of the defaults.
  Be aware that the communication port and options valid on your computer are not necessarily valid on other computers. Use this option with care.
  - Save settings to registry = if ticked, the settings are stored in the local computer registry database. These settings are used by default when the application is started the next time.
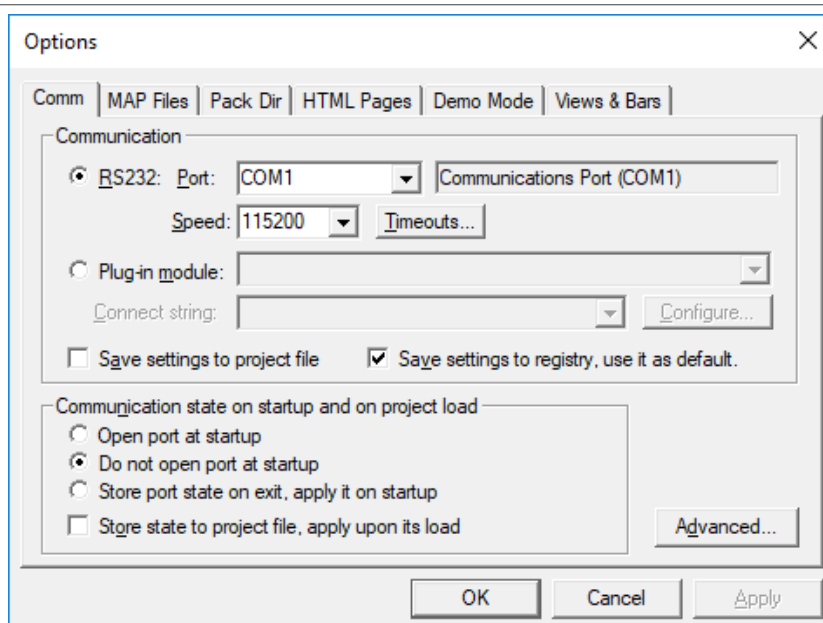


Figure 42. Communication options

*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 with 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. The remote server must be properly installed also on the remote side. If you have 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 start up.
  - Do not open port at start up.
  - Store port status on exit, apply it on start up.
  - Store status to a project file, apply it on its load = if ticked, 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.
- Press the Advanced button to open the dialog with communication thread priority settings. However, these settings typically do not need to be changed.

## 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 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 that 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 an embedded application tested on an evaluation 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.

*Note:*  *In addition to loading the application executable or a symbol file, the feature called Target-Side Addressing (TSA) may be enabled in the target MCU driver. The TSA enables the application programmer to define the so-called TSA tables which describe the data types and variables to be displayed in FreeMASTER. FreeMASTER reads the TSA tables and uses the information automatically when an MCU board is connected. The TSA information takes a precedence if both the TSA application and the symbol file are used. A great benefit of using the TSA are safety and security reasons. The variables described by TSA tables may be read-only, so even if FreeMASTER attempts to write the variable, the value is actively denied by the target MCU side. The variables not described by any TSA tables may also become invisible and protected even for read-only access.*

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

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**47 / 142**
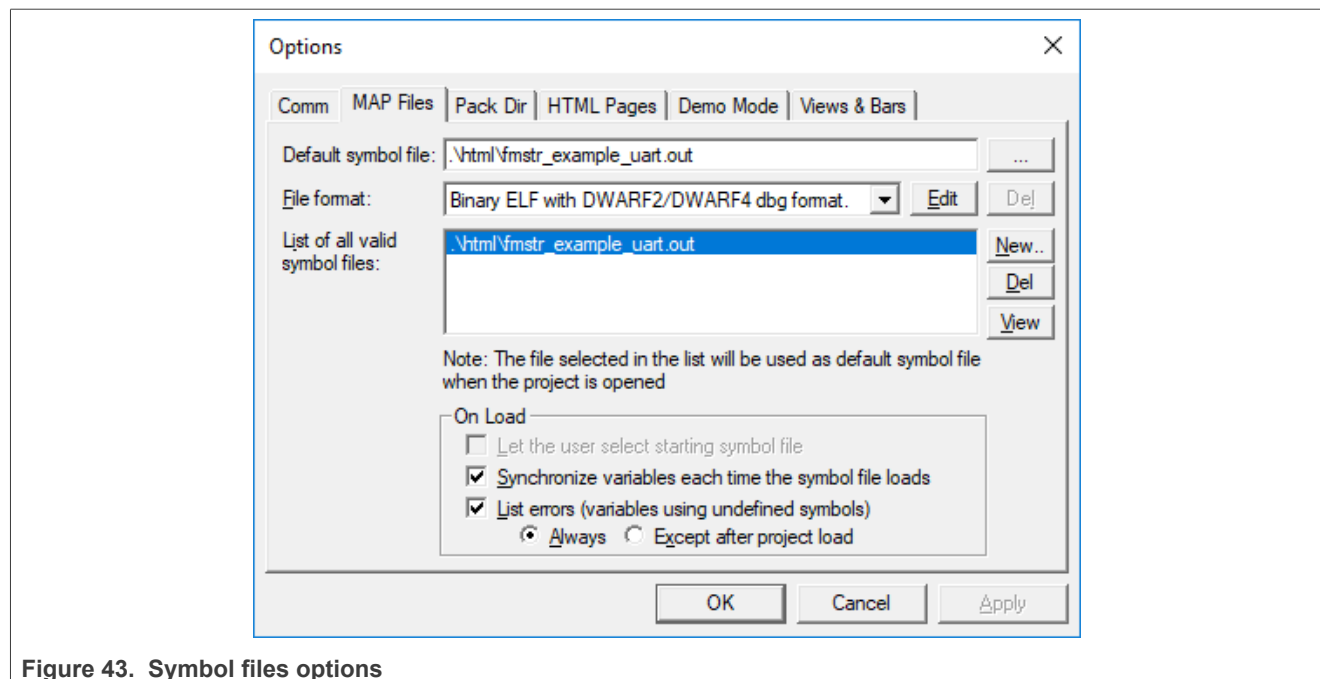
**Figure 43. Symbol files options**

- 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 file format.
  - 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 option to define a new text file parser based on the regular expression pattern matching; see Section 5.2.1.
- 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 most suitable for project deployment to other computers; see Section 5.3 for more information.*

- The 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 ticked, you are prompted to select the initial symbol file when the project is loaded.
  - Synchronize variables each time the map file loads = when ticked, 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 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. However, 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 Compiler and Linker for the NXP DSC56F800E processor family.

The example of the xMap line, which describes the global symbol length, may 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 that describe the symbol parameters, put them in round parentheses:

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

Now 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.
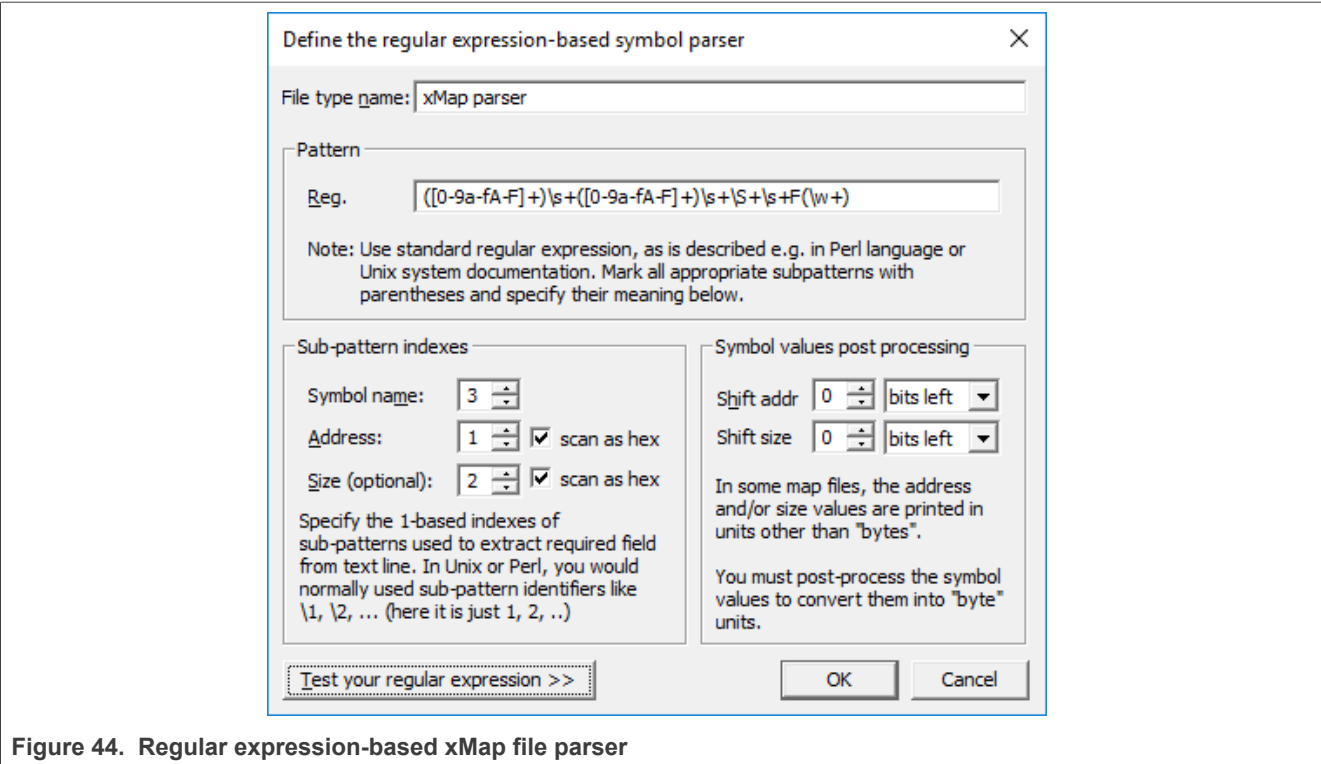
**Figure 44. Regular expression-based xMap file parser**

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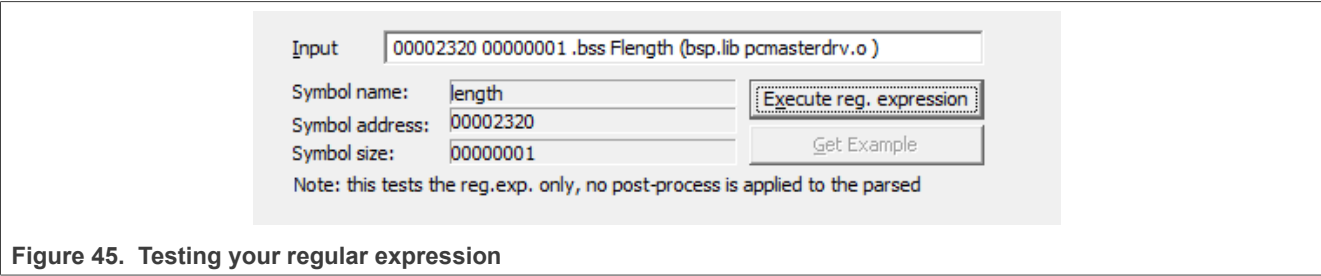


**Figure 45. Testing your regular expression**

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

To finish the example, set the 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 this value is needed 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 a 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

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**50 / 142**

deploy or distribute a project to different computers. Using the Pack Dir options shown in the below figure, you can choose to pack the resource files into the project file when it is saved.
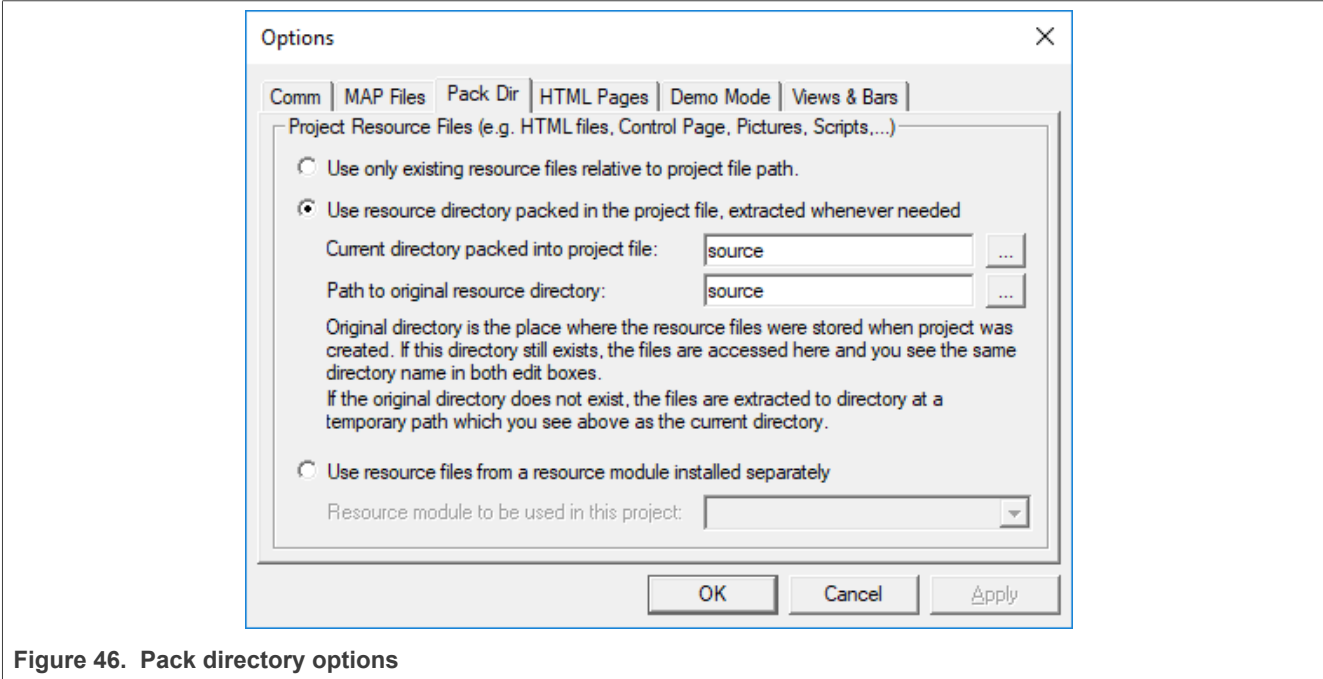


**Figure 46. Pack directory options**

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 should 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 it still remembers the path to the original resource directory and checks its existence any time the project opens in the future.
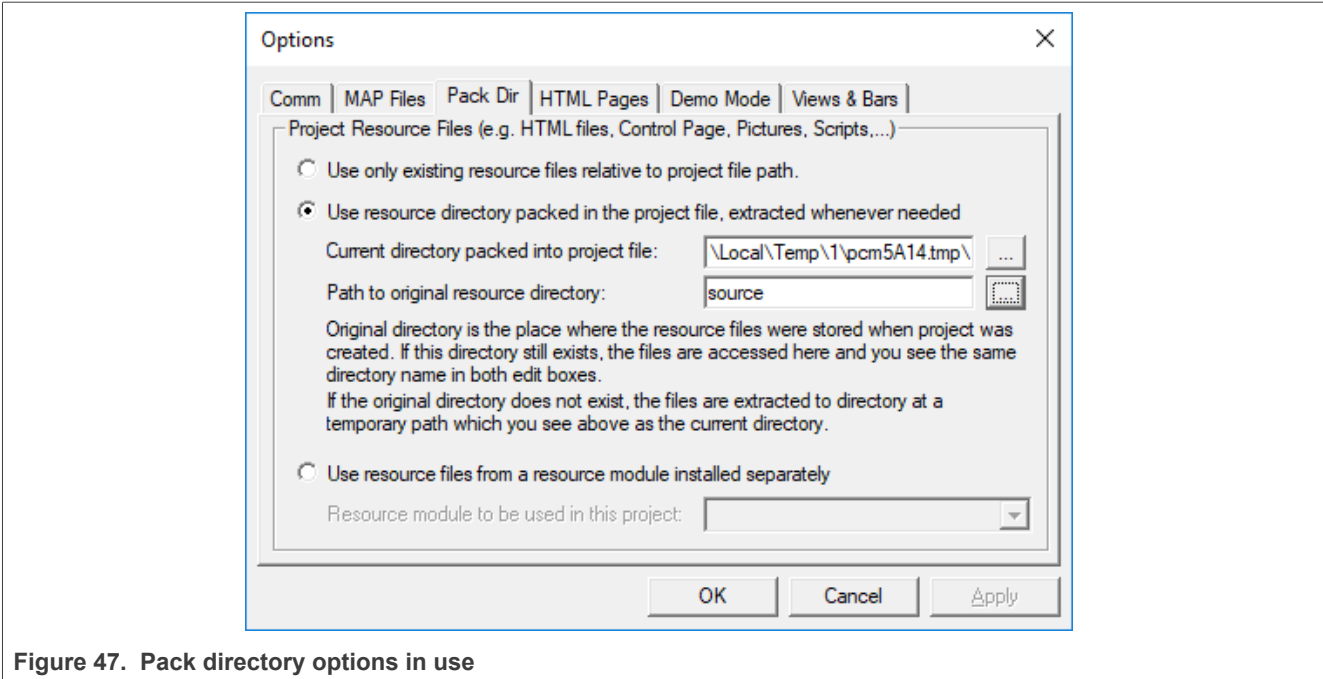
**Figure 47. Pack directory options in use**

### 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.
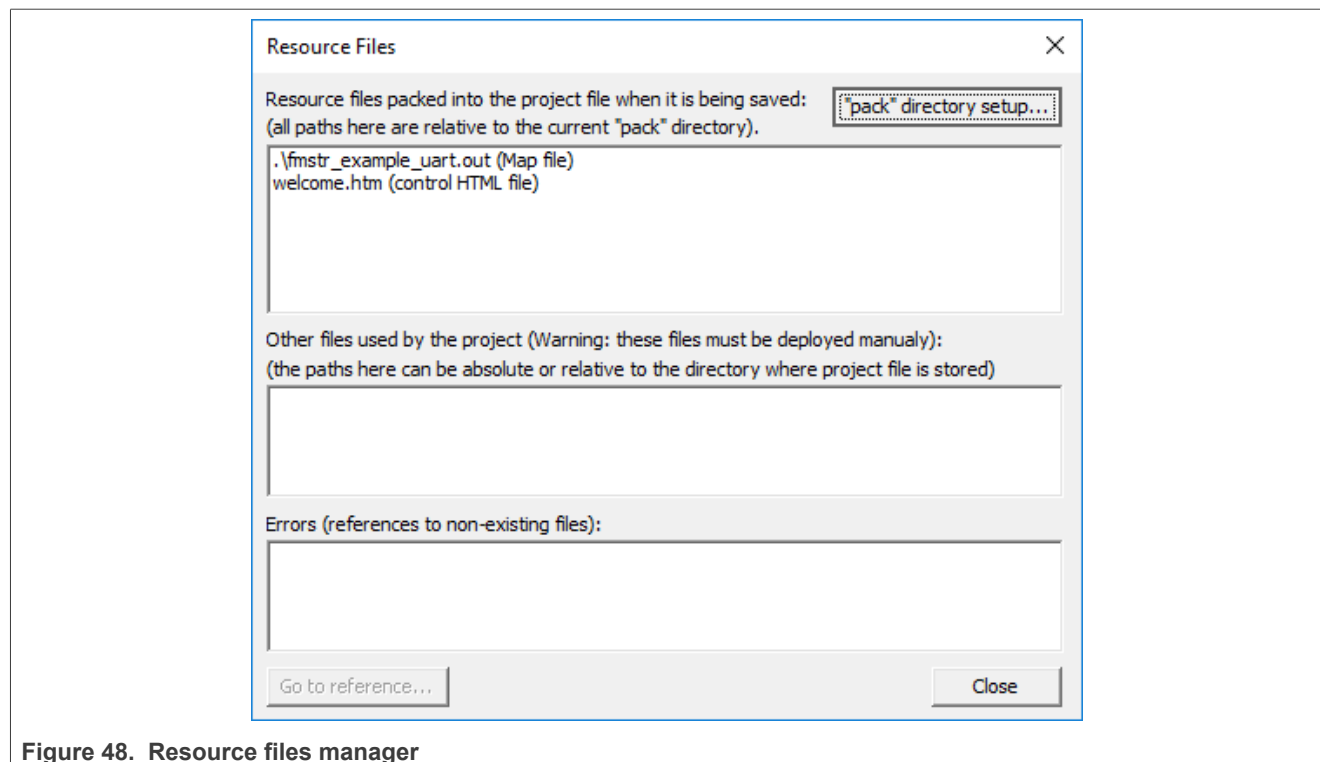
**Figure 48.** Resource files manager

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 the Properties dialog for a project tree item or the 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. In 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.

**User guide**                          **Rev. 4.8 — 18 September 2024**                               Document feedback

53 / 142
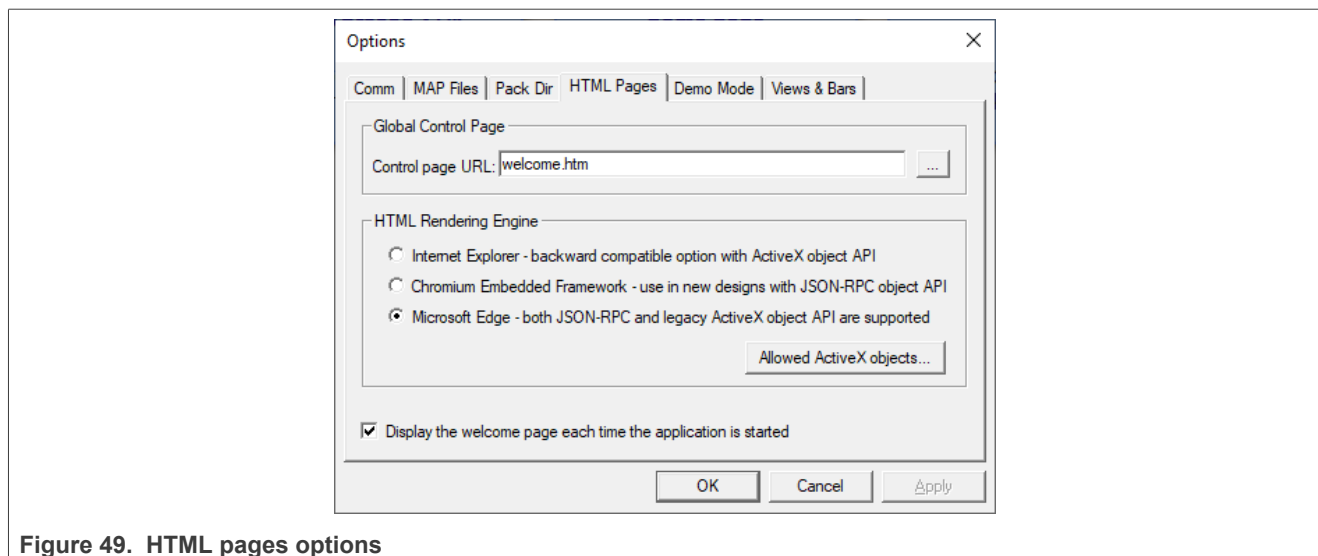
**Figure 49.  HTML pages options**

Set the Control page URL to the name of the HTML file used as a general control page in the FreeMASTER project. The control page is accessible all the time from the main FreeMASTER window. It is typically used to host a custom user interface related to the target microcontroller application.

The control page typically contains graphical control elements (like push-buttons or gauges) and also embeds the JavaScript code that implements the elements' logic and behavior. There are three HTML rendering engines supported by FreeMASTER:

- Internet Explorer = this option is backward compatible with older FreeMASTER releases. Scripts should use the ActiveX technology to communicate with the FreeMASTER application object and get access to the target microcontroller resources.
- Chromium Embedded Framework = this option is available since FreeMASTER 3.0 and uses an embedded Chromium view to render the HTML pages and run the JavaScript code. Use the JSON-RPC protocol to communicate with the FreeMASTER application object.
- Microsoft Edge WebView2 = added in FreeMASTER 3.2 as a new browser view compatible with both the JSON-RPC interface as well as COM+ and ActiveX technology. See Section 6.10.

*Note:  When using a script in the Microsoft Edge view to dynamically create ActiveX objects, specify a list of enabled objects by pressing the "Allowed ActiveX Objects" button.*

See the related information in Section 4.1.2.1 and Section 6.

## 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 to the Demo Mode. See the following figure for details of the Demo Mode tab.
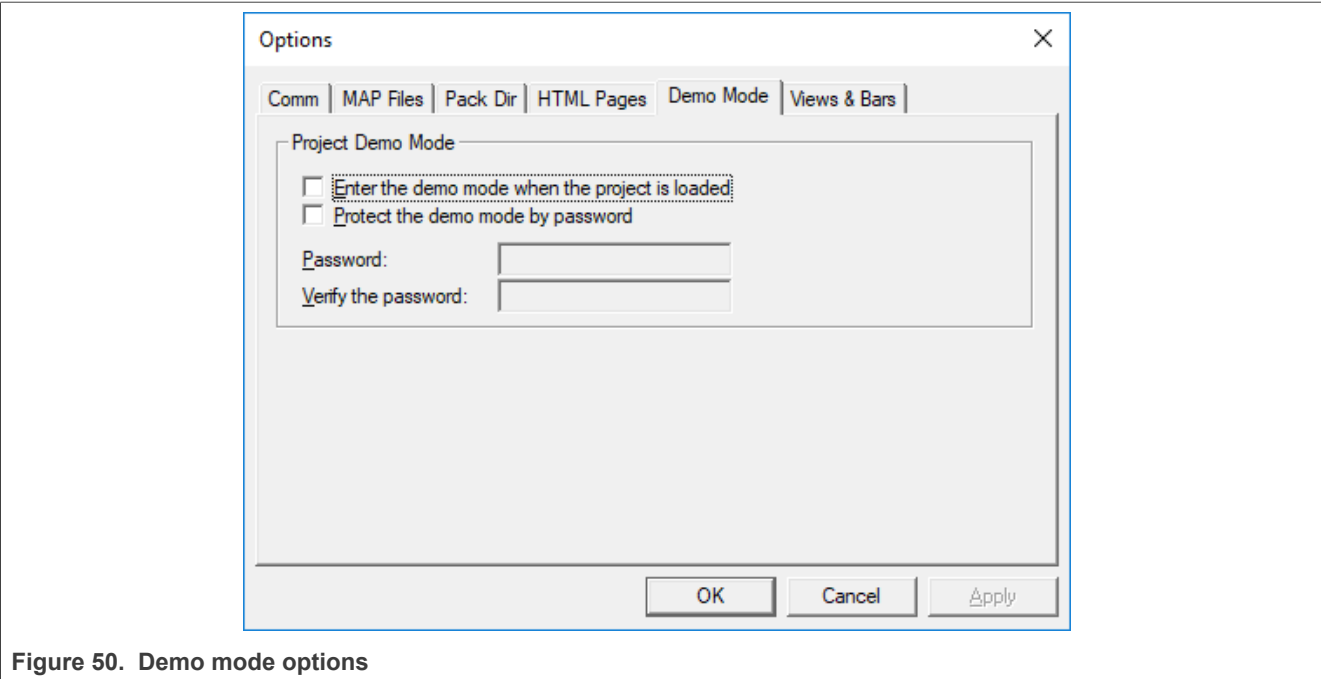
**Figure 50.  Demo mode options**

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.
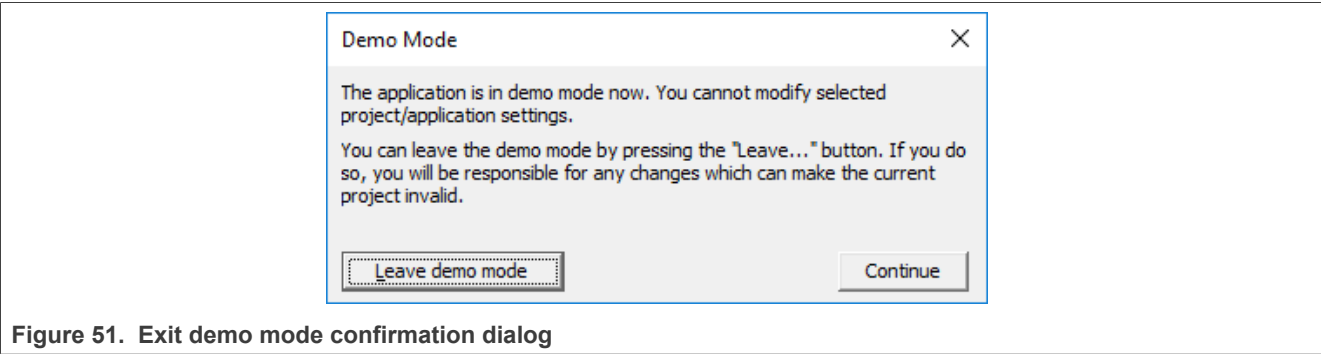


**Figure 51.  Exit demo mode confirmation dialog**

# 6  HTML and scripting

FreeMASTER version 2.x renders all HTML pages using the standard Microsoft Internet Explorer component. FreeMASTER 3.0 adds an option to switch from Internet Explorer to the Chromium rendering engine. FreeMASTER 3.2 further extends the options by adding support for the Microsoft Edge WebView2 component.

All three ways enable the user to create HTML pages extended by JavaScript code which may access the FreeMASTER features. Using this approach, a powerful user interface with custom logic, data processing, and graphical controls can be created. This user interface may be hosted directly inside the FreeMASTER main window, so it appears as an integral part of the FreeMASTER application to the end users.

FMSTERUG

**User guide**

All information provided in this document is subject to legal disclaimers.

**Rev. 4.8 — 18 September 2024**

© 2024 NXP B.V. All rights reserved.

Document feedback

**55 / 142**

Choose Internet Explorer based the following criteria:

- It is backward compatible, so it works with older FreeMASTER versions (2.x) as well.
- It supports ActiveX objects, so it is the only option to use when your graphical design relies on third-party objects.
- The communication with the FreeMASTER ActiveX interface is synchronous, which simplifies the script design. On the other hand, synchronous waiting for the ActiveX method completion may cause the user interface to be less responsive.

Benefits of the Chromium engine:

- Supports asynchronous JavaScript programming and other modern scripting technologies.
- Communicates with both FreeMASTER 3.0 and FreeMASTER "Lite" using the JSON-RPC protocol and a fully asynchronous promise-like interface.
- Pages can be developed and tested in a standalone Chrome browser, supported by a powerful JavaScript debugger.

When to use the Microsoft Edge WebView2 engine:

- It is fully compatible with the Chromium view and asynchronous JavaScript programming techniques.
- It supports the FreeMASTER and FreeMASTER Lite JSON-RPC interfaces.
- When running embedded inside the FreeMASTER main window, this view also partially supports COM+ and ActiveX object creation. It may be used as a good modern replacement for the outdated Internet Explorer option.
- It is an automatically activated option when the Internet Explorer component is disabled in the system.
- Some script and HTML code changes may be needed when switching from the Internet Explorer view to the Microsoft Edge view. See the Internet Explorer migration guide at the end of this section.

The following text is recommended primarily for Control Page authors experienced with the HTML page design and scripting.

## 6.1  Special HTML hyperlinks

When creating HTML pages that are to be displayed in the FreeMASTER environment, you can use special hyperlinks to let the users navigate in the project tree or to invoke selected application commands.

- "HREF=pcmaster:selitem:itemname:tabname" selects a project tree item named "itemname", displays the selected tab in the detail view, and sets the other view content exactly as if you had selected the item manually.
- The valid "tabname" identifiers are:
  - ctl = Control Page tab
  - blk = Algorithm Block Description tab
  - info = Current Item Help tab
  - osc= Oscilloscope tab
  - rec = Recorder tab

The application command invocation hyperlinks include:

- "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: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.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**56 / 142**

## 6.2 FreeMASTER ActiveX interface

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.
- SaveProject saves the current project or opens a "Save As" dialog.
- StartStopComm, StartComm, and StopComm open or close the communication port.
- IsCommPortOpen and IsBoardDetected can be used to determine the connection status.
- GetHtmlDocument retrieves the HTML DOM object from the FreeMASTER window.
- SendComman sends a FreeMASTER-defined command.
- SendCommandDlg opens a command dialog and sends a FreeMASTER-defined command.
- ReadVariable and ReadMultipleVariables read a value from FreeMASTER-defined variables.
- WriteVariable and WriteMultipleVariables write a value to FreeMASTER-defined variables.
- ReadMemory and "ReadMemoryHex" read a block of memory from a specified location.
- ReadIntArray, ReadUIntArray, ReadFloatArray, and ReadDoubleArray read numeric arrays from a specified location.
- WriteIntArray, "WriteUIntArray, WriteFloatArray, and WriteDoubleArray write numeric arrays to a specified location.
- ReadIntVariable, ReadUIntVariable, ReadFloatVariable, and ReadDoubleVariable read a single value from a specified location.
- WriteIntVariable, WriteUIntVariable, WriteFloatVariable, and WriteDoubleVariable write a single value to a specified location.
- GetCurrentRecorderData retrieves the data currently displayed in the recorder or array chart.
- GetCurrentRecorderSeries retrieves one data series from the currently displayed recorder or array chart.
- StartCurrentRecorder starts the currently displayed recorder or array.
- StopCurrentRecorder stops the currently displayed recorder or array.
- GetCurrentRecorderState retrieves the current recorder or array status code and text.
- LocalFileOpen and LocalFileClose enable you 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, GetStructMemberInfo, and GetAddressInfo retrieve information about symbols in the target application.
- DefineSymbol and DeleteAllScriptSymbols manipulate the symbolic information on top of the ones provided by the target application.
- SubscribeVariable and UnSubscribeVariable enable the script to be notified when a variable value changes.
- SelectItem selects the project item in the FreeMASTER project tree view.

- DefineVariable, DefineOscilloscope, DefineRecorder, DefinePipe, DefineArrayViewer, and DefineWatchBlock are complex functions which can be used to dynamically create or modify FreeMASTER variable objects or Oscilloscope/Recorder graphs.
- GetVariableInfo, GetRecorderDefinition, GetOscilloscopeDefinition, GetPipeDefinition, GetArrayViewerDefinition, GetWatchBlockDefinition can be used to retrieve complete information about the project items.
- IsBoardWithActiveContent, EnumHrefLinks, and 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 and EnumSymbols enumerate the variables and symbols loaded in the current project.
- GetDetectedBoardInfo retrieves information about the connected board.
- GetCommPortInfo retrieves information about the current communication port settings.
- FireCustomEvent sends a notification event to all connected ActiveX and JSON-RPC clients.

These callback events are implemented:

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

## 6.3 FreeMASTER JSON-RPC interface

When the FreeMASTER application starts, it initializes the JSON-RPC server listening on port number 41000. The server accepts plain TCP or WebSocket connections from the clients. If more FreeMASTER instances are started simultaneously, ports 41001 to 41016 are automatically used unless you specify the port manually using the "/rpcs" command-line option when running the executable.

There are many ways how the client script connects and how it calls the JSON-RPC methods remotely. The way recommended and demonstrated in the default FreeMASTER examples uses a JavaScript object wrapping the functionality of a popular "simple-jsonrpc-js" implementation available on GitHub. The wrapper object is named PCM (for sentimental and legacy reasons) and it is implemented in the "freemaster-client.js" file available in the FreeMASTER installation folder. This file is also instantly available to all clients of the FreeMASTER Lite web server.

With the PCM wrapper object, calling the remote JSON-RPC methods is as simple as calling general asynchronous JavaScript methods locally with standard arguments. All the methods available in the legacy ActiveX interface are also available in the PCM objects with the same order of parameters.

Establish the JSON-RPC connection by calling:

```
rpcs_addr = "localhost:41000";
pcm = new PCM(rpcs_addr, on_connected, on_error, on_error);
```

Pass the "on_connected" and "on_error" callback methods, which handle successful connection or errors. The "rpcs_addr" parameter should be set to "localhost:41000" unless the server port is overridden by the "/rpcs" command-line option. FreeMASTER version 3.1.3 adds a new possibility to retrieve the valid address from a JScript object named "FreeMASTER", which is defined by including a special script file:

```
<!-- Get the information object named "FreeMASTER" -->
<script type="text/javascript" src="fmstr://localapp/info.js"></script>
```

The statement fetches a virtual "info.js" file from the current "hosting" FreeMASTER instance of the Chromium HTML rendering engine. Be aware that this file is only available inside a running FreeMASTER application.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 4.8 — 18 September 2024

Document feedback

58 / 142

When an HTML/JScript code runs standalone in a web browser and for backward compatibility, always use the default value of "localhost:41000".

Any result data returned by the JSON-RPC methods are accessible in the "data" member of the response object. The format of the data member depends on the method called. See Section 6.5 for more details. The response object returned by the FreeMASTER desktop application may also contain the "xtra" object member with additional information that is not part of the FreeMASTER Lite responses.

For example, there is a basic JSON-RPC method to retrieve the variable value:

```
promise = pcm.ReadVariable("variable_name");
```

When the read is successful, both the FreeMASTER and FreeMASTER Lite servers return a response object (a so-called "Promise" object), which resolves to the "data" member with the returned variable value. The FreeMASTER desktop application also returns the "xtra.formatted" value with the same value, formatted according to the variable object settings. See the code example in Section 6.8.2 for more information about using the JSON-RPC interface.

## 6.4 FreeMASTER Lite

FreeMASTER Lite was first introduced with FreeMASTER 3.0. It is a service-like application that can be started on the host computer to enable the remote access from various clients to the target microcontroller board. Clients access the service using the JSON-RPC protocol with an API that is almost the same as the one provided by the desktop FreeMASTER application. The major difference from the desktop application is that FreeMASTER Lite does not have any user interface. It is configured using a local JSON configuration file which specifies the physical connection parameters, describes the ELF file to load symbols from, and also defines the Variable objects that are accessible to clients. When the service starts, it runs silently on the host computer and listens for incoming client connections.

FreeMASTER Lite also runs a classic web server which supplies HTML pages and other local files to remote clients. This radically simplifies the deployment of graphical control pages to mobile devices like phones or tablets. The mobile device simply starts a web browser and opens the internet address of the computer that runs the FreeMASTER Lite service.

Here are typical FreeMASTER Lite use cases:

- A back-end engine of the FreeMASTER Control Page applications running solely in a web browser; either on a remote mobile device (like a tablet) or on the same computer using a localhost connection.
- A service that enables standalone, custom, third-party applications to connect to the target microcontroller application. Any application that supports scripting and network communication can connect to FreeMASTER Lite. For example, Matlab, LabView, and others.
- A service that enables custom JavaScript or Python scripts to connect to a microcontroller board. This may be useful for testing scripts and other automated environments.
- It provides FreeMASTER connectivity on Windows and Linux operating systems.

### 6.4.1 FreeMASTER vs. FreeMASTER Lite

The following list highlights the major differences between FreeMASTER Lite and the desktop FreeMASTER application.

**FreeMASTER Lite is cross-platform.**

- The desktop FreeMASTER only runs on the Windows operating system.
- FreeMASTER Lite runs on Windows and Linux operating systems.

**The JSON-RPC API is similar, but not identical.**

- Both APIs can be accessed using the JavaScript PCM wrapper object which encapsulates the JSON-RPC remote calls. See the "freemaster-client.js" file in the FreeMASTER installation for the implementation details. This file is also served by FreeMASTER Lite's built-in web server.
- The desktop FreeMASTER has additional methods which only make sense in this application. For example, SelectItem, OpenProject, or Exit enable the manipulation with the user interface and are not available in FreeMASTER Lite.
- The desktop FreeMASTER has also more information about the "visual" aspect of variables as defined by the user (like print formatting, value transformation, or value-to-text enumeration). Therefore, reading a variable by FreeMASTER Lite only returns the variable value. The desktop FreeMASTER also returns an 'xtra' object with additional information that may be the formatted text-based value.
- Unlike the Lite service, the desktop FreeMASTER runs its own background processing and may provide its own timing. This enables the variable Stimulator to be started using a JSON-RPC call after which the stimulator keeps running independently. With FreeMASTER Lite, the client application handles the whole periodic or time-based processing.
- Similarly, the desktop FreeMASTER monitors the connection to the target board and generates the so-called events. The events are asynchronous JSON-RPC notification messages generated by the server and handled on the client side. The events are: OnBoardDetected, OnRecorderDone, OnVariableChanged (for subscribed variables), and so on.

## 6.5 ActiveX and JSON-RPC methods

The following sections may be used as a reference of the FreeMASTER ActiveX methods and their counterparts from the JSON-RPC API.

The full reference of the JSON-RPC methods supported by the FreeMASTER Lite server is available separately in a form of hypertext document generated automatically from the script source code. See the *html/index.html* file in the FreeMASTER Lite installation folder for more details. You can also launch the service executable either from the installation folder or created shortcut to automatically open the mentioned page in the system default web browser.

Note that the method declarations described below use the input and output parameters as well as the return value. Such a full declaration is only usable in the VB script, Visual Basic for Applications, and other languages that support returning by-reference "output" parameters. Other languages (like JavaScript) only get the general return value, but they cannot get the output parameter values.

There are two ways to access the output values:

- When using the ActiveX interface, the output parameters are accessible after the call returns by using the LastResult, LastRetMsg, and other LastXXX properties of the main interface object.
- When using the JSON-RPC interface, the response object contains the "data" return value and also the "xtra" object with additional members containing the remaining output values. Note that the "xtra" object is not returned by the FreeMASTER Lite server.

*Note: The true value returned by the methods of the PCM object, which wraps the JSON-RPC calls, is a standard Promise object representing a remote call. If a remote call is successful, the Promise object resolves the response object with the "data" and "xtra" members. To signal a successful execution, the response object also contains the "succeeded" member set to the "true" value. When the remote call fails due to communication issues or when the response object contains "succeeded" set to "false", the Promise object is rejected.*

*It is strongly recommended to study rich Internet sources to become familiar with the Promise interface and asynchronous JavaScript programming concepts before starting to code scripts using the JSON-RPC interface.*

The following API reference pages contain these markings in the method prototype specification:

- [in] - denotes the required input parameter.
- [in, optional] - denotes an optional input parameter; the value may be omitted when calling a method.

- [out] - denotes the by-reference output parameter. See the "Outputs" tables in the below text to determine how to access the output value. All output parameters are also optional.

Each method is also marked with these compatibility signs:

- ActiveX - when checked ✓, the method is available in the ActiveX interface.
- JSON-RPC - when checked ✓, the method is available in the JSON-RPC interface implemented by the desktop FreeMASTER application. Use the PCM object which wraps the JSON-RPC remote calls using the *freemaster-client.js* file.
- Lite - when checked ✓, the method is available in the JSON-RPC interface implemented by FreeMASTER Lite. When not checked, or marked with ✗, the function is not supported by FreeMASTER Lite.

## 6.5.1 GetAppVersion

**Prototype:**

```
GetAppVersion([out] retMsg)
```

**Description:**

Gets the FreeMASTER application version.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite - APIs not fully compatible

**Inputs:**

None

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value (ActiveX only) | LastResult | response.xtra.retval | Version number as a scalar numeric value. |
| retMsg | LastRetMsg | response.data | Version number formatted as a text. This value is returned by the JSON-RPC method. |

## 6.5.2 OpenProject

**Prototype:**

```
OpenProject ([in] project)
```

**Description:**

Opens the specified project file.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| project | String containing the project file path. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | Boolean return value, true if project open was successful. |

### 6.5.3  SaveProject

**Prototype:**

```
SaveProject([in] saveAs);
```

**Description:**

Saves the current project or opens a "Save As" dialog.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| saveAs | When a boolean value is passed: when false, the project is silently saved to the last known path. When true or when the project is new, the SaveAs dialog opens. |
| | When a string value is passed: the project is saved as a new file name. If a relative path is provided, it is evaluated relative to an existing saved project. The new file name must contain either a **pcm** or **pcmx** extension. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned when the project is saved. |

### 6.5.4  StartStopComm

**Prototype:**

```
StartStopComm ([in] start)
```

**Description:**

Opens or closes the FreeMASTER communication port selected in the currently loaded project.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| start | Boolean value |
| | • **true** - start communication |
| | • **false** - stop communication |

**Outputs:**

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**62 / 142**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | Boolean return value if port operation was successful. |

**Remarks:**

The FreeMASTER Lite interface contains distinct StartComm and StopComm methods which are also recommended to use with the desktop FreeMASTER application in new designs.

### 6.5.5 StartComm

**Prototype:**

```
StartComm ([in] name)
```

**Description:**

Opens the FreeMASTER communication port identified by the name or by the full connection string. This function should be used instead of StartStopComm. It is useful when working with FreeMASTER desktop application when a calling script must specify all parameters of the new connection instead of using the settings configured within the FreeMASTER project.

The connect string should be in a form of "RS232" or {CLSID} in curly brackets followed by a semicolon-delimited optional arguments.

The following short names can be used instead of full plug-in CLSID identifiers:

```
"DAP"      {1BB3C904-F1F4-4652-92EE-368716FE1D10}
"EONCE"    {5A129680-897E-4014-916D-9B1FE13DE156}
"PEMICRO"  {225E034C-1AA6-4EC5-9A6C-A9FA9E890373}
"JLINK"    {8C575DB3-9769-4B66-AB60-C83CEA683E01}
"ISYSTEM"  {80F17965-EDBD-41F1-9182-3C44E04E5794}
"PDBDM"    {30AB7AA1-493A-4DD7-9509-C9C91EFFF0FD}
"CAN"      {C10A92C3-7D47-4FDC-94B6-64B8E5C85E01}
"LIN"      {C10A92C4-7D47-4FDC-94B6-64B8E5C85E01}
"NET"      {6D13CD9D-9F2D-4066-B655-45B54CA7494B}
```

**Examples of connection strings:**

```
RS232;port=COM1;speed=115200
CAN;drv=kvaser;port=1;bitrate=500000;cmdid=0x7aa;rspid=0x7aa;tmo=500
DAP;devid=0;dapid=0;DAP_Port=SW;DAP_SWJ=Y;DAP_Clock=1000000
PEMICRO;drv=6;ptype=3;pnum=1;devid=;devlock=0;jtagspd=500;target=NXP_K6x_K64FN1M0M12
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| name | String identifying the connection name in the FreeMASTER Lite configuration file. Use a special name "preset" to identify the default connection built in the FreeMASTER desktop application. In both FreeMASTER and FreeMASTER Lite cases, the name can also be specified as a full connection string, so it allows to open serial or plug-in connections with all configuration parameters. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | Boolean return value if port operation was successful. |
| retMsg | LastRetMsg | response.xtra.message | When an error occurs, this value contains an error message. |

### 6.5.6  StopComm

**Prototype:**

```
StopComm ()
```

**Description:**

Closes the currently open FreeMASTER communication port.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

None

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | Boolean return value if port operation was successful. |

### 6.5.7  IsCommPortOpen

**Prototype:**

```
IsCommPortOpen ()
```

**Description:**

Returns the current state of the communication port.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

None

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.data | Boolean value representing the current communication port open state. |

FMSTERUG
All information provided in this document is subject to legal disclaimers.
© 2024 NXP B.V. All rights reserved.

**User guide**
**Rev. 4.8 — 18 September 2024**
Document feedback
**64 / 142**

### 6.5.8 IsBoardDetected

**Prototype:**

```
IsBoardDetected ()
```

**Description**

Returns the state of a connection to the target board.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

None

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.data | Boolean "true" value if the board is currently detected. |

### 6.5.9 IsBoardWithActiveContent

**Prototype:**

```
IsBoardWithActiveContent ()
```

**Description:**

This function can be used to determine if Active Content is present in the currently connected target MCU application. Active Content is enabled using the Target-Side Addressing (TSA) tables in the MCU application and includes HTML pages, project files, hyperlinks, and other file-like resources. See the FreeMASTER Serial Driver documentation for more details about TSA.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

None

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.data | Boolean "true" value if the board is currently detected and has TSA Active Content available. |

**Remarks:**

See the JavaScript example of using this function in the source code of the FreeMASTER Welcome page which is displayed as soon as FreeMASTER starts. The Welcome page detects and displays the hyperlinks and project files embedded in the target MCU application code.

### 6.5.10 GetHtmlDocument

**Prototype:**

```
GetHtmlDocument ([in] winId, [in] activate)
```

**Description:**

Get a handle to the Document Object Model (DOM) object representing the HTML document in the Internet Explorer rendering engine.

**Compatibility:**

✓ ActiveX, ✗ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| winId | A number identifying the HTML pane to access. Use one of these values:<br>• **0** - Control page<br>• **1** - Block description page<br>• **2** - Detailed description page (for Oscilloscope and Recorder items) |
| activate | Boolean value which selects whether to activate the selected HTML view. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| Object dispatch handle | LastResult | N/A | The returned handle can be used in the script to access the HTML Document object. |

### 6.5.11 SendCommand

**Prototype:**

```
SendCommand ([in] cmd,  [in] wait, [out] retMsg)
```

**Description:**

Sends the FreeMASTER Application Command.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| cmd | 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)`". |
| wait | Boolean value indicating that the process shall wait until the target returns a command result or until a timeout occurs. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned if the command was sent without errors. "False" is returned if the command could not be found in the FreeMASTER project. |
| retMsg | LastRetMsg | response.xtra.message | Text returned after the command invocation. When an error occurs, this value contains an error message. |
| | LastAppCommand _ret Code | response.xtra.retCode | Command return code. This parameter is only valid when the **wait** argument is "true". |

### 6.5.12 SendCommandDlg

**Prototype:**

```
SendCommandDlg ([in] cmd, [out] retMsg)
```

**Description:**

Invokes the FreeMASTER *Send Application Command* dialog.

**Compatibility:**

✓ ActiveX, ✗ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| cmd | 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. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | N/A | A boolean "true" value is returned if the command was sent without errors. "False" is returned if the command could not be found in the FreeMASTER project. |
| retMsg | LastRetMsg | N/A | Text returned after the command invocation. When an error occurs, this value contains an error message. |

### 6.5.13 ReadVariable

**Prototype:**

```
ReadVariable ([in] var, [out] numValue, [out] textValue, [out] retMsg)
```

**Description:**

Read a value of a FreeMASTER variable. The variable must be defined in the current FreeMASTER project or in the FreeMASTER Lite configuration file.

**For the desktop FreeMASTER application only:** This method may return the "cached" value of the variable if it is newer 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 ("!") before the variable name in the *var* parameter.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| var | String value with the name of the variable to be read. The variable must be defined in the Free MASTER project that is currently open or in the FreeMASTER Lite configuration file. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A Boolean "true" value is returned if the variable was read without errors. "False" is returned if the specified variable was not found in the Free MASTER project or if a communication error occurred. |
| numValue | LastVariable_vValue | response.data | Returns a numeric representation of the variable *var*. |
| textValue | LastVariable_tValue | response.xtra. formatted | Returns a string value which represents the variable format and units necessary for displaying the variable value. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message; otherwise, it is empty. |

### 6.5.14 WriteVariable

**Prototype:**

```
WriteVariable ([in] var, [in] value, [out] retMsg)
```

**Description:**

Write a value to a FreeMASTER variable. The variable must be defined in the current FreeMASTER project or in the FreeMASTER Lite configuration file.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| var | A string value with the name of the variable to be written. The variable must be defined in the Free MASTER project that is currently open or in the FreeMASTER Lite configuration file. |
| value | The value to write to the variable. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|-----------|----------------|-----------------|-------------|
| return value | LastResult | response.xtra.retval | A boolean "true" value 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. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

### 6.5.15 ReadMemory

**Prototypes:**

```
ReadMemory ([in] addr, [in] size, [out] data, [out] retMsg)
ReadMemoryHex ([in] addr, [in] size, [out] textData, [out] retMsg)
```

**Description:**

Reads a block of memory from the target application.

- *ReadMemory* returns data compatible with any scripting environment.
  - ActiveX: data are returned as a safe array of variants, which can be used in scripting languages, such as VBScript, and in compiled languages, such as Visual Basic. For JavaScript, the safe array must be converted to JavaScript array using the `toArray()` method.
  - JSON-RPC: data are returned as a native JSON array.

For ActiveX only, there are also other optional functions that can be used in different scripting environments:

- *ReadMemory_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.
- *ReadMemoryHex* returns data in the string value. Each byte is represented by two characters in a hexadecimal format.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|----------|-------------|
| addr | 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 complex expression which involves the symbol name, offset value additions, multiplication, or using an array de-referencing operator. |
| size | The size of the memory block to be read. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|-----------|----------------|-----------------|-------------|
| return value | LastResult | response.xtra.retval | A boolean "true" value 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. |
| data | LastMemory_data | response.data | The return array of values. |
| textData (ReadMemory Hex only) | LastMemory_hexstr | response.xtra.hex | The return data in the string format. |

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message; otherwise, it is empty. |

**Remarks:**

This function is provided for backward compatibility only. Use ReadUIntArray with the element size set to 1 byte instead.

### 6.5.16 WriteMemory

**Prototype:**

```
WriteMemory ([in] addr, [in] size, [in] data, [out] retMsg)
```

**Description:**

Write a block of bytes to the target application's memory.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| addr | 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 complex expression which involves the symbol name, offset value additions, multiplication, or using an array de-referencing operator. |
| size | The size of the memory block to be written. |
| data | The safe array of bytes to be written. It must contain at least the *size* elements. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "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. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

**Remarks:**

This function is provided for backward compatibility only. Use WriteUIntArray with the element size set to 1 byte instead.

### 6.5.17 ReadXxxArray

**Prototypes:**

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

```
ReadIntArray ([in] addr, [in] size, [in] elemSize, [out] data, [out] retMsg)
```

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

```
ReadUIntArray ([in] addr, [in] size, [in] elemSize, [out] data, [out] retMsg)
```

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

```
ReadFloatArray ([in] addr, [in] size, [out] data, [out] retMsg)
```

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

```
ReadDoubleArray ([in] addr, [in] size, [out] data, [out] retMsg)
```

**Description:**

Reads a block of memory from the target application and returns it to the caller as an array of integer or floating-point numbers.

- ReadXxxArray returns data compatible with any scripting environment.
  - ActiveX: data are returned as a safe array of variants, which can be used in scripting languages, such as VB script, and in compiled languages, such as Visual Basic. Because the VB script 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. For JavaScript: the safe array must be converted to the JavaScript array using the `toArray()` method.
  - JSON-RPC: data are returned as a native JSON array.

For ActiveX only, there are also other optional functions that can be used in different scripting environments:

- ReadXxxArray_v is the same as ReadXxxArray, except that it does not perform the VB script 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.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| addr | 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 complex expression which involves the symbol name, offset value additions, multiplication, or using an array de-referencing operator. |
| size | The number of elements to be read from the array. |
| elemSize | The size of an array element in bytes. The total number of bytes read from the target can be calculated as *size* * *elemSize*. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value 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. |
| data | LastMemory_data | response.data | The return array of the values. |

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

### 6.5.18 WriteXxxArray

**ActiveX Prototypes:**

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

```
WriteIntArray ([in] addr, [in] size, [in] elemSize, [in] data, [out] retMsg)
```

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

```
WriteUIntArray ([in] addr, [in] size, [in] elemSize, [in] data, [out] retMsg)
```

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

```
WriteFloatArray ([in] addr, [in] size, [in] data, [out] retMsg)
```

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

```
WriteDoubleArray ([in] addr, [in] size, [in] data, [out] retMsg)
```

**JSON-RPC Prototypes:**

JSON-RPC functions do not have the "size" argument; array size is inferred from data automatically.

```
WriteIntArray ([in] addr, [in] elemSize, [in] data, [out] retMsg)
WriteUIntArray ([in] addr, [in] elemSize, [in] data, [out] retMsg)
WriteFloatArray ([in] addr, [in] data, [out] retMsg)
WriteDoubleArray ([in] addr, [in] data, [out] retMsg)
```

**Description:**

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.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite - APIs not fully compatible

**Inputs:**

| Argument | Description |
|---|---|
| addr | 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 complex expression which involves the symbol name, offset value additions, multiplication, or using an array de-referencing operator. |
| size (ActiveX only) | The number of elements to be written to the target memory. Set it to 0 to write all array elements as passed in the *data* parameter. |
| elemSize | The size of an array element in bytes. The total number of bytes to be written to the target can be calculated as *size * elemSize*. |
| data | The array of the values to be written. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value 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. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

### 6.5.19 ReadXxxVariable

**Prototypes:**

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

```
ReadIntVariable ([in] addr, [in] size, [out] value, [out] retMsg)
```

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

```
ReadUIntVariable ([in] addr, [in] size, [out] value, [out] retMsg)
```

To access 4-byte floating-point variable:

```
ReadFloatVariable ([in] addr, [out] value, [out] retMsg)
```

To access 8-byte floating-point variable:

```
ReadDoubleVariable ([in] addr, [out] value, [out] retMsg)
```

**Description:**

Reads a memory from the target application and returns 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 compatible with any scripting environment.
  - ActiveX: a value is returned as a variant which can be used in scripting languages, such as VB script, and in compiled languages, such as Visual Basic. Because the VB script 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.
  - JSON-RPC: a value is returned as a native JSON value.

For ActiveX only, there is also another optional function that can be used in different scripting environments:

- ReadXxxVariable_v is the same as ReadXxxVariable, except that it does not perform the VB script translation for 2 and 4-byte unsigned integer types.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| addr | 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 complex expression which involves the symbol name, offset value additions, multiplication, or using an array de-referencing operator. |

FMSTERUG
All information provided in this document is subject to legal disclaimers.
© 2024 NXP B.V. All rights reserved.

**User guide**
**Rev. 4.8 — 18 September 2024**
Document feedback
**73 / 142**

| Argument | Description |
|----------|-------------|
| size | The size of the variable to read. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|-----------|----------------|-----------------|-------------|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned if the memory was read without errors. "False" is returned if the specified address was invalid or if a communication error occurred. |
| value | LastVariable_vValue | response.data | The return value. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

### 6.5.20 WriteXxxVariable

**Prototypes:**

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

```
WriteIntVariable ([in] addr, [in] size, [in] value, [out] retMsg)
```

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

```
WriteUIntVariable ([in] addr, [in] size, [in] value, [out] retMsg)
```

To write 4-byte floating-point number:

```
WriteFloatVariable ([in] addr, [in] size, [in] value, [out] retMsg)
```

To write 8-byte floating-point number:

```
WriteDoubleVariable ([in] addr, [in] size, [in] value, [out] retMsg)
```

**Description:**

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

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|----------|-------------|
| addr | 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 complex expression which involves the symbol name, offset value additions, multiplication, or using an array de-referencing operator. |
| size | The size of the target variable. |
| value | The value to be written. |

**Outputs:**

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**74 / 142**

| Parameter | ActiveX access | JSON-RPC access | Description |
|-----------|----------------|-----------------|-------------|
| return value | LastResult | response.xtra.retval | A boolean "true" value 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. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

### 6.5.21 ReadMultipleVariables

**Prototype:**

```
ReadMultipleVariables ([in] arrNames, [out] outputJSON)
```

**Description**

Use this function to read multiple FreeMASTER-defined variables in a single call. This call is functionally equal to a sequential execution of several ReadVariable calls. Use it to minimize the overhead time consumed by ActiveX or JSON-RPC communication and other framework-related processing, which may take up to several milliseconds in some cases.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|----------|-------------|
| arrNames | Array of variable names to read. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|-----------|----------------|-----------------|-------------|
| return value | LastResult | response.xtra.retval | Number of variables which were successfully read during the call. |
| outputJSON | LastMultipleVariables_json | response.data | ActiveX: string with a JSON-formatted array of result objects. JSON-RPC: the array of result objects. There is one array element for each variable name passed as an input when calling the function. Each array element is an object with the following properties: <br>• **name** - variable name <br>• **status** - boolean result of the read operation <br>• **value** - numeric variable value |

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| | | | • **formatted** - string formatted value<br>• **retMsg** - error message |

### 6.5.22  WriteMultipleVariables

**Prototype:**

```
WriteMultipleVariables ([in] arrValues, [out] outputJSON)
```

### Description

Use this function to write multiple FreeMASTER-defined variables in a single call. This call is functionally equal to a sequential execution of several WriteVariable calls. Use it to minimize the overhead time consumed by ActiveX or JSON-RPC communication and other framework-related processing, which may take up to several milliseconds in some cases.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| arrValues | Array of variable names and values to be written. Each array element should be an object with the following properties:<br>• **name** - variable name<br>• **value** - variable value to be written |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | Number of variables which were successfully written during the call. |
| outputJSON | LastMultipleVariables_json | response.data | ActiveX: string with a JSON-formatted array of result objects.<br>JSON-RPC: array of result objects.<br>There is one array element for each variable value passed as an input when calling the function.<br>Each array element is an object with the following properties:<br>• **name** - variable name<br>• **status** - boolean result of write operation<br>• **retMsg** - error message |

### 6.5.23 GetCurrentRecorderData

**Prototype:**

```
GetCurrentRecorderData ([out] data, [out] serieNames, [out] timeBaseSec, [out] retMsg)
```

**Description:**

Retrieves the data currently displayed in the recorder or array chart.

- GetCurrentRecorderData returns two-dimensional array data compatible with any scripting environment. The first dimension is the series index, the second dimension is the value index.
  - ActiveX: data are returned as two-dimensional safe array of variants, which can be used in scripting languages, such as VB script, and in compiled languages, such as Visual Basic. Because the VB script 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. For JavaScript: the safearray must be converted to a JavaScript array using the `toArray()` method.
  - JSON-RPC: data are returned as a native JSON array of arrays, one for each recorder data series.

For ActiveX only, there are also other optional functions that can be used in different scripting environments:

- GetCurrentRecorderData_t returns data in the safe array 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.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

None.

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned if the data was retrieved successfully from the recorder or array item. "False" is returned if there is no valid data in the recorder or array or if there is no currently active recorder or array. |
| data | LastRecorder_data | response.xtra.data | Returns two-dimensional array of values; the first dimension is the series index, the second dimension is the value index. |
| serieNames | LastRecorder _serie Names | response.xtra.names | Returns an array with the names of series on appropriate indexes. |
| timeBaseSec | LastRecorder _time BaseSec | response.xtra. base RateSec | Returns the time between individual recorded samples in seconds; this value can be 0 if the target does not supply such value. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

**Remarks:**

FreeMASTER Lite does not support this method as there is no concept of a "currently selected" recorder or array item as is common for the desktop application. FreeMASTER Lite provides methods to control and read the recorder or array directly.

The method is not supported in the Edge ActiveX as the Edge does not support 2-dimensional arrays. Use GetCurrentRecorderSeries for a specific series instead.

### 6.5.24 GetCurrentRecorderSeries

**Prototype:**

```
GetCurrentRecorderSeries ([in] serieName, [out] data, [out] timeBaseSec, [out] retMsg)
```

**Description:**

Retrieve one data series from the currently displayed recorder or array chart.

- GetCurrentRecorderSeries returns one-dimensional array data compatible with any scripting environment.
  - ActiveX: data are returned as a safe array of variants, which can be used in scripting languages, such as VB script, and in compiled languages, such as Visual Basic. Because the VB script 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. For JavaScript: the safearray must be converted to a JavaScript array using the `toArray()` method.
  - JSON-RPC: data are returned as a native JSON array

For ActiveX only, there are also other optional functions that can be used in different scripting environments:

- GetCurrentRecorderSeries_t returns data in the safe array of "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.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| serieName | String with the name of the data series (which is a recorded variable name) whose data is to be retrieved. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" is returned if the data was retrieved successfully from the recorder or array item. "False" is returned if there is no valid data in the recorder or array or if there is no recorder or array currently active. |
| data | LastRecorder_data | response.xtra.data | Return array of the values. |
| timeBaseSec | LastRecorder _time BaseSec | response.xtra. base RateSec | Returned time between individual recorded samples in seconds; this value can be 0 if the target does not supply such value. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

**Remarks:**

FreeMASTER Lite does not support this method as there is no concept of a "currently selected" recorder or array item as it is common for the desktop application. FreeMASTER Lite provides methods to control and read the recorder or array directly.

## 6.5.25 StartCurrentRecorder

**Prototype:**

```
StartCurrentRecorder ([out] retMsg)
```

**Description:**

Starts the recorder or array which is currently displayed in the FreeMASTER window.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

None

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned if the recorder or array was started successfully. "False" is returned if there is no recorder or array currently active. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

**Remarks:**

FreeMASTER Lite does not support this method as there is no concept of a "currently selected" recorder or array item as is common for the desktop application. FreeMASTER Lite provides methods to control and read the recorder or array directly.

## 6.5.26 StopCurrentRecorder

**Prototype:**

```
StopCurrentRecorder ([out] retMsg)
```

**Description:**

Manually stops the recorder or array which is currently displayed in the FreeMASTER window.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

None

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned if the recorder or array was successfully stopped. "False" is returned if there is no recorder or array currently active. |

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**79 / 142**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

**Remarks:**

FreeMASTER Lite does not support this method as there is no concept of a "currently selected" recorder or array item as is common for the desktop application. FreeMASTER Lite provides methods to control and read the recorder or array directly.

### 6.5.27 GetCurrentRecorderState

**Prototype:**

```
GetCurrentRecorderState ([out] state, [out] retMsg)
```

**Description:**

Retrieve the current recorder or array status code and the assigned status text.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

None

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned if the recorder or array state was read successfully. "False" is returned if there is no recorder or array currently active. |
| state | LastRecorder_state | response.xtra.state | Returns a numeric value identifying the current recorder or array state. The valid state codes are:<br>• **0** - idle<br>• **1** - starting<br>• **2** - running<br>• **3** - downloading results<br>• **4** - holding received signal<br>• **5** - error<br>• **6** - manually stopping<br>• **7** - not initialized<br>• **8** - holding in error<br>• **9** - ready to download |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it contains a text description of the current recorder state. |

**Remarks:**

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 4.8 — 18 September 2024

Document feedback

80 / 142

FreeMASTER Lite does not support this method as there is no concept of a "currently selected" recorder or array item as is common for the desktop application. FreeMASTER Lite provides methods to control and read the recorder or array directly.

### 6.5.28 RunStimulators

**Prototype:**

```
RunStimulators ([in] stimNames)
```

**Description:**

Starts one (or more) FreeMASTER variable stimulators.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| stimNames | A semicolon-delimited list of stimulators to be started. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | The number of stimulators actually started. This number may be equal or lesser than the number of semicolon-delimited stimulator names passed in the *stimNames* parameter. The return count does not include the stimulators not found by name and the stimulators that are already running. |

### 6.5.29 StopStimulators

**Prototype:**

```
StopStimulators ([in] stimNames)
```

**Description:**

Halt one (or more) FreeMASTER variable stimulators.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| stimNames | A semicolon-delimited list of stimulators to be stopped. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | The number of stimulators actually stopped. This number may be equal or lesser than the number of semicolon-delimited stimulator names passed in the *stimNames* parameter. The return count does not include stimulators not found by name and stimulators that are not running. |

### 6.5.30 LocalFileOpen

**Prototype:**

```
LocalFileOpen ([in] fileName, [in] openMode)
```

**Description:**

Open a file stored locally in the project area and return a handle to the file for subsequent operations.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| fileName | 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:<br><br>`.txt, .xml, .htm, .html, .c, .cpp, .asm, .h, .hpp`<br>The relative path can start with one of the virtual folder names:<br>• `FMSTR_PROJECT_PATH` - location of the current project file.<br>• `FMSTR_PACKDIR_PATH` - location of the current resource module folder. |
| openMode | Specifies the access mode of the open file:<br>• **'r'** – open file for reading (default).<br>• **'w'** – create or open file for writing, original file gets truncated to zero length if it already exists.<br>• **'a'** – create or open file for writing in append mode, original file is not truncated if it already exists. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.data | Numeric handle to file or a "false" boolean value in case the file could not be opened. |

### 6.5.31 LocalFileClose

**Prototype:**

```
LocalFileClose ([in] handle)
```

**Description:**

Close the file identified by the *handle* parameter as returned by a previous LocalFileOpen call.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| handle | Handle to the file to be closed. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned if the file was successfully closed. "False" is returned in case the file handle is not valid. |

### 6.5.32 LocalFileWriteString

**ActiveX prototype:**

```
LocalFileWriteString ([in] handle, [in] data, [in, optional] charsToWrite, [in, optional] unicode)
```

**JSON-RPC prototype:**

JSON-RPC function does not have the *charsToWrite* argument, the string size is inferred from data automatically.

```
LocalFileWriteString ([in] handle, [in] data, [in, optional] unicode)
```

**Description:**

Write data to a file identified by the *handle* parameter as returned by a previous [LocalFileOpen](#) call.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite - APIs not fully compatible

**Inputs:**

| Argument | Description |
|---|---|
| handle | Handle to open the file. |
| data | Text to be written to the file. |
| charsToWrite (ActiveX only) | The number of characters to be actually written. If this parameter is omitted or set to 0, the whole text passed in the *data* parameter is written. |
| unicode | "True" - write the data in unicode format.<br>"False" - write data in the ASCII format. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.data | Returns the number of characters written to the file. |

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**83 / 142**

### 6.5.33  LocalFileReadString

**Prototype:**

```
LocalFileReadString ([in] handle, [in] charsToRead, [in, optional] unicode, [out] retString)
```

**Description:**

Reads data from a file identified by the *handle* parameter as returned by a previous [LocalFileOpen](#) call.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| handle | Handle to open the file. |
| charsToRead | The number of characters to read. |
| unicode | "True" - read data in unicode format.<br>"False" - read data in ASCII format. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | Returns the number of characters actually read from the file. |
| retString | LastLocalFile_string | response.data | The return data containing the characters read from the file. |

### 6.5.34  GetSymbolInfo

**ActiveX prototype:**

```
GetSymbolInfo ([in] symbol, [out] symAddr, [out] symSize, [out] retMsg, [in] useVBScriptTypes)
```

**JSON-RPC prototype:**

```
GetSymbolInfo ([in] symbol)
```

**Description:**

Returns the information about the symbol as parsed from the application's ELF or MAP files or obtained in the runtime using the TSA feature.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| symbol | Name of the symbol. |
| useVBScriptTypes (ActiveX only) | Use "true" to return the address and size as floating-point numbers compatible with the VB script scripting language. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned if the symbol exists and the information was retrieved. "False" is returned when the symbol was not found. |
| symAddr | LastSymbolInfo_addr | response.xtra.addr | Address of the symbol. |
| symSize | LastSymbolInfo_size | response.xtra.size | Size of the symbol. |
| N/A | LastSymbolInfo_type | response.xtra.type | Data type name. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

### 6.5.35 GetStructMemberInfo

**ActiveX prototype:**

```
GetStructMemberInfo ([in] typeName, [in] member, [out] membOff, [out] membSize, [out] retMsg, [in]
  useVBScriptTypes)
```

**JSON-RPC prototype:**

```
GetStructMemberInfo ([in] typeName, [in] member)
```

**Description:**

Returns the information about the structure data type member as parsed from the application's ELF file or obtained in the runtime using the TSA feature.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| typeName | Name of the structure or union data type. |
| member | Name of the structure or union member. |
| useVBScriptTypes (ActiveX only) | Use "true" to return the address and size as floating-point numbers compatible with the VB script scripting language. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned if the type exists and information was retrieved. "False" is returned when the structure type was not found. |
| membOff | LastMemberInfo_offset | response.xtra.offset | Returns the offset of the structure member. |
| membSize | LastMemberInfo_size | response.xtra.size | Returns the size of the structure member. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

### 6.5.36 GetAddressInfo

**Prototype:**

```
GetAddressInfo ([in] addr, [in] size, [out] isExactMatch, [out] symbolName)
```

**Description:**

Returns the information about the memory location as parsed from the application's ELF file or obtained in the runtime using the TSA feature. All global and static target symbols are evaluated to see if they overlap at least partly 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.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| addr | Memory address. |
| size | Size of the memory area to look up. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned if the memory area was mapped to any global or static symbol. A "false" value is returned when the memory area was not mapped to any symbols. |
| isExactMatch | LastAddressInfo_exact | response.xtra.isExact Match | Returns the "true" value if the returned symbol exactly matches the memory address and size. |
| symbolName | LastAddressInfo_name | response.xtra.symbol Name | Returns the best-matching name of the memory area. |
| N/A | LastAddressInfo_type | response.xtra.symbol Type | Data type name. |

### 6.5.37 GetXxxDefinition

**Prototypes:**

```
GetOscilloscopeDefinition([in] name, [out] defString, [out] retMsg)
GetRecorderDefinition([in] name, [out] defString, [out] retMsg)
GetArrayViewerDefinition([in] name, [out] defString, [out] retMsg)
GetPipeDefinition([in] name, [out] defString, [out] retMsg)
GetWatchBlockDefinition([in] name, [out] defString, [out] retMsg)
GetStimulatorDefinition([in] name, [out] defString, [out] retMsg)
GetEnumDefinition([in] name, [out] defString, [out] retMsg)
```

**Description:**

These functions may be used to obtain the JSON definition record of each respective kind of item. The JSON record can later be used in the DefineOscilloscope, DefineRecorder, DefineArrayViewer, DefinePipe, DefineWatchBlock, DefineStimulator, and DefineEnum methods described above to create a new project item with modified properties and settings.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|----------|-------------|
| name | Name of the item to get definition object of. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|-----------|----------------|-----------------|-------------|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned when the object was found and a definition record is provided. |
| defString | LastDefinition_string | response.xtra.def | ActiveX: string with a JSON-formatted definition. JSON-RPC: definition object itself. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

### 6.5.38  DefineSymbol

**Prototype:**

```
DefineSymbol ([in] symbol, [in] addr, [in] typeName, [in] size, [out] retMsg)
```

**Description:**

This function can be used to extend the symbol information normally obtained by parsing the target application executable file (ELF or MAP files) or obtained in the runtime using the TSA feature. 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 for the symbol in the desktop application interface or using the [DefineVariable](#) function.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|----------|-------------|
| symbol | Symbol name. It can also contain special characters, such as dot `.`, `->`, or brackets `[]`, when defining complex symbols mapping structure members. |
| addr | Symbol address. |
| typeName | 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 not specified or if an empty string is passed, no type is assigned to the symbol. |
| size | Symbol size. You may also use special expressions, such as `sizeof(TYPE)` or mathematical expressions. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|-----------|----------------|-----------------|-------------|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned if the symbol was defined with the type information as |

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**87 / 142**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| | | | specified. "False" is returned when the type name was specified, but no such type was found in the type information loaded from the application executable file. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

### 6.5.39 DeleteAllScriptSymbols

**Prototype:**

```
DeleteAllScriptSymbols()
```

**Description:**

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

**Compatibility:**

✔ ActiveX, ✔ JSON-RPC, ✗ Lite

**Inputs:**

None.

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | This method returns the "true" value. |

### 6.5.40 SubscribeVariable

**Prototype:**

```
SubscribeVariable ([in] varName, [in] interval, [out] retMsg)
```

**Description:**

Enables event-driven processing of variable value changes. When a variable is "subscribed", the script receives the OnVariableChanged notification event whenever FreeMASTER detects that the variable value changed. All subscribed variables are periodically sampled by FreeMASTER at a specified time interval.

**Compatibility:**

✔ ActiveX, ✔ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| varName | Variable name; this must be a valid FreeMASTER variable name not just a symbol name. |
| interval | An interval in which the variable is tested for changes. The OnVariableChanged event is only called once per this interval value, even if the variable changes more frequently. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra. subscriptionId | 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 On VariableChanged event callback.<br><br>Zero value when the variable could not be subscribed. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

**Remarks:**

For the JSON-RPC interface, call the EnableEvents function to activate notification sending.

### 6.5.41 UnSubscribeVariable

**Prototype:**

```
UnSubscribeVariable ([in] nameOrId, [out] retMsg)
```

**Description:**

Cancels the variable subscription and stops the OnVariableChanged event calls for the variable identified by the first parameter.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| nameOrId | Variable name used to subscribe the variable (or subscription ID) which was returned from the SubscribeVariable call. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned if the variable was unsubscribed. "False" is returned if an error occurred. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

### 6.5.42 SelectItem

**Prototype:**

```
SelectItem ([in] name, [in] tabPage)
```

**Description:**

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

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| name | Name of the item, as it appears in the FreeMASTER "Project Tree" view. This can be the name itself or a fully qualified path of all parent block names and the item name, separated by a slash character. |
| tabPage | Name of the FreeMASTER view tab which is to be activated after the tree item is selected. Use one of these values: <br> • **control** - Control Page tab <br> • **blkinfo** - Parent Block description page <br> • **iteminfo** - Item description page <br> • **scope** - Oscilloscope graph tab <br> • **recorder** - Recorder graph tab |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned when the item was found and selected. "False" is returned otherwise. |

### 6.5.43 DeleteItem

**Prototypes:**

```
DeleteItem([in] name, [out] retMsg)
```

**Description**

Deletes the item in the project tree.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| name | Name of the item, as it appears in the FreeMASTER "Project Tree" view. <br> This can be the name itself or a fully qualified path of all parent block names and the item name, separated by a slash character. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | Boolean "true" value is returned when the item was found and deleted. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**90 / 142**

### 6.5.44 FireCustomEvent

**Prototype:**

```
FireCustomEvent([in] arg)
```

**Description**

Use this function to fire a notification event (OnCustomEvent) in all clients attached to this server instance. The event is fired in all clients of both the ActiveX and JSON-RPC types. The JSON-RPC clients must subscribe to events by calling EnableEvents.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| arg | A variant-type (any script type) event argument which is passed to the OnCustomEvent handler. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | Boolean "true" value is returned when the event was fired. |

### 6.5.45 DefineVariable

**ActiveX prototype:**

```
DefineVariable ([in] name, [in] defString, [out] retMsg)
```

**JSON-RPC prototype:**

```
DefineVariable ([in] defString)
```

**Description:**

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 that are not defined in JSON are assigned a default value when creating a new object.

The following JSON example gives a brief reference of the definition record. It can also be retrieved by calling GetVariableInfo(). Note that the same record is used as a as "var_def" object within the clipboard text format when copying a variable in the "Variable Watch" view.

```
{
    // general information
    "name": "var16",           // variable name
    "period": 200,             // sampling period in milliseconds
    "comment": "",             // comment visible in Watch view
    "description": "",         // internal description
    "unit": "unit",            // physical units of REAL mode
    // variable type
    "treat_as": 0,             // 0=uint, 1=int, 2=float, 3=fract, 4=ufract, 5=string
    "type_fmt": "",            // fract/ufract type description (e.g. UQ1.15)
    // physical location and type in embedded application
    "address": "var16",        // C symbol name or an address expression
    "byte_size": 2,            // size of the variable
```

FMSTERUG
All information provided in this document is subject to legal disclaimers.
© 2024 NXP B.V. All rights reserved.

**User guide**
**Rev. 4.8 — 18 September 2024**
Document feedback
**91 / 142**

```
        "bit_shift": 0,               // bit-field shift
        "bit_mask": "-1",             // bit-field mask
        // virtual variable special settings
        "virtual": 0,                 // non-zero to set as virtual
        "virtual_default": "0",       // default virtual variable value
        // enumeration type
        "enum_enabled": 0,            // non-zero to enable enumerations
        "enum": "",                   // name of the enum type
        "enum_num_always": 0,         // display numeric value always
        "enum_default_val": "unk",    // default text label
        "enum_num_with_default": 1,   // numeric value visible at default text
        // visualization in the Watch view
        "show_as": 0,                 // 0=DEC,1=HEX,2=BIN,3=ASCII,4=REAL,
        "show_val": 1,                // value visible
        "show_max": 0,                // maximum visible
        "show_min": 0,                // minimum visible
        "show_num_fixed_digs": 0,     // fixed number of digits
        "show_num_lzero_fill": 0,     // use 0 to fill fixed number of digits
        "show_num_afp_digs": 0,       // number of digits after decimal point
        "show_num_exp": 0,            // non-zero to show in exponential format
        "show_ascii_zterm": 0,        // display as zero-terminated string (ASCII)
        "show_ascii_hex": 1,          // display non-printable chars as HEX
        "show_ascii_chwdt": 1,        // character width in bytes
        // numeric filter
        "filt_enabled": 0,            // non-zero to enable averaging filter
        "filt_reset_onmod": 1,        // reset filter when value is modified
        "filt_time": 5000,            // filter period in milliseconds
        // variable modification properties
        "modif_mode": 1,              // 0=disabled, 1=any value, 2=pre-defined values
        "modif_edit_mode": 2,         // 1=edit box, 2=drop-down list
        "modif_set_mode": 0,          // 0=write after any change, 1=wait for Enter key
        "modif_auto_fin": 1,          // finish edit mode after Enter key
        "modif_auto_hide": 1,         // hide edit interface if cell is not focused
        "modif_predefs_modes": 1,     // predefined values: 0x1=range, 0x2=enum, 0x4=other
        "modif_predefs_other": "",    // "other" pre-defined values (when 0x4 flag is used)
        "modif_set_min": "",          // minimum value for range-based predef. values
        "modif_set_max": "",          // maximum value for range-based predef. values
        "modif_set_step": "",         // iteration step for range-based predef. values
        // advanced parameters
        "modif_address": "",          // shadow write address
}
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite - APIs not fully compatible

**Inputs:**

| Argument | Description |
|---|---|
| name (ActiveX only) | Name of the FreeMASTER variable to be created or modified. |
| defString | ActiveX: String with a JSON-formatted definition.<br>JSON-RPC: definition object itself. The new variable name may also be specified in this definition object as the "name" value. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned when the object was created or modified. "False" is returned if an error occurred. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

**JavaScript example:**

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

```
    "byte_size" : size,
    "period" : period,
  };
  ok = pcm.DefineVariable(name, JSON.stringify(def));
}
```

### 6.5.46 DefineOscilloscope

**Prototype:**

```
DefineOscilloscope ([in] name, [in] defString, [out] retMsg)
```

**Description:**

Defines the FreeMASTER Oscilloscope object with all its properties. The script must use a JSON notation to describe all properties of the object. The Oscilloscope object is created or modified if it already exists. All properties that are not defined in JSON are assigned a default value when creating a new object.

The following JSON example gives a brief reference of the definition record. It can also be retrieved by calling GetXxxDefinition. Note that the same record is used as a "tree_item" object within the clipboard text format when copying an oscilloscope item in the "Project Tree" view.

```
{
    "name": "var16 scope",       // oscilloscope item name
    "href": "",                  // description page URL
    // oscilloscope settings
    "scope_period": 2.5e-002,    // sampling period in seconds
    "scope_ignore_duplicity": 0, // ignore samples which are identical to previous
    "scope_x_width": 5,          // x-axis total width
    "scope_x_autoscale": 0,      // autoscale x-axis until width is reached
    // graph settings
    "graph_buffer": 2000,        // graph buffer size in points
    "graph_type": 0,             // graph type: 0=Time, 1=XY
    "graph_legend_loc": 0,       // nonzero if legend is visible
    "graph_legend_vis": 1,       // nonzero to put legend to graph interior
    "graph_legend_inside": 1,    // exterior legend location: 0=top, 1=bottom, 2=left, 3=right
    "graph_grid_horz": 1,        // show graph horizontal grid
    "graph_grid_vert": 1,        // show graph vertical grid
    "graph_x_units": 2,          // X-axis time units: 0=us, 1=ms, 2=sec, 3=min (Time graph only)
    "graph_x_label": "Time",     // X-axis label (Time graph only)
    "graph_x_addunits": 1,       // non-zero to add time unit name to axis label (Time graph only)
    "graph_x_min": 0,            // nonzero to enable automatic X axis minimum value (XY graph only)
    "graph_x_max": 100,          // nonzero to enable automatic X axis maximum value (XY graph only)
    "graph_x_min_auto": 1,       // manual X axis minimum value (XY graph only)
    "graph_x_max_auto": 1,       // manual X axis maximum value (XY graph only)
    "auto_delete": 0,            // nonzero to set the Oscilloscope item to auto-delete mode
    // oscilloscope variables
    "var_info": [                // array of graph variables
      {
        "var_def": {             // variable definition record
          "name": "var16",       // specifying 'name' is enough to locate existing variable
        },
        "visible": 1,            // variable visible in graph
        "y_block": 0,            // assign to y-block 0..4
        "color": 3026413         // subset color
      },
      {
        "var_def": {
          "name": "var16inc",
        },
        "variable": "var16inc",
        "visible": 1,
        "y_block": 1,
        "color": 4688896
      }
    ],
    // graphs details for each Y-axis block
    "yblock_info": [             // array of Y-block definitions
      {
        "join_class": 0,         // blocks with equal value will be joined
        // left-axis settings
        "laxis_label": "16 bits", // label
        "laxis_style": 0,        // style
        "laxis_min_auto": 0,     // use automatic minimum scaling
        "laxis_max_auto": 1,     // use automatic maximum scaling
```

FMSTERUG

User guide

All information provided in this document is subject to legal disclaimers.

Rev. 4.8 — 18 September 2024

© 2024 NXP B.V. All rights reserved.

Document feedback

93 / 142

```
                "laxis_min": 0,          // minimum value (when auto=0)
                "laxis_max": 10,         // maximum value (when auto=0)
                // right-axis settings
                "raxis_label": "Increment", // label
                "raxis_style": 0,        // style
                "raxis_min": 0,          // use automatic minimum scaling
                "raxis_max": 10,         // use automatic maximum scaling
                "raxis_min_auto": 0,     // minimum value (when auto=0)
                "raxis_max_auto": 1,     // maximum value (when auto=0)
                "raxis_subsets": 1       // number of subsets to assign to right axis
            },                           // (counted from the last subset)
            {
                "join_class": 1,         // a different class means to separate this block
                "laxis_label": "lAxis",
                "laxis_style": 0,
                "laxis_min": -10,
                "laxis_max": 10,
                "laxis_min_auto": 1,
                "laxis_max_auto": 1,
                "raxis_label": "rAxis",
                "raxis_style": 0,
                "raxis_min": -10,
                "raxis_max": 10,
                "raxis_min_auto": 1,
                "raxis_max_auto": 1,
                "raxis_subsets": 0
            },
        ],
}
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| name | Name of the FreeMASTER oscilloscope item to be created or modified in the project tree. |
| defString | ActiveX: string with a JSON-formatted definition<br>JSON-RPC: definition object itself |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned when the object was created or modified. "False" is returned if an error occurred. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

**Remarks:**

This function is also available as DefineScope in the ActiveX interface for backward compatibility.

**JavaScript example:**

```
function define_my_scope()
{
  // array of oscilloscope 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";
   pcm.DefineScope("my oscilloscope", JSON.strinigy(def));
}
```

### 6.5.47  DefinePipe

**Prototype:**

```
DefineRecorder ([in] name, [in] defString, [out] retMsg)
```

**Description:**

Defines the FreeMASTER Pipe object with all its properties. The script must use a JSON notation to describe all properties of the object. The Pipe object is created or modified if it already exists. All properties that are not defined in JSON are assigned a default value when creating a new object.

The following JSON example gives a brief reference of the definition record. It can be also retrieved by calling GetXxxDefinition. Note that the same record is used as a "tree_item" object within the clipboard text format when copying a pipe item in the "Project Tree" view.

```
{
    "name": "Terminal Pipe",     // array viewer item name
    "href": "",                  // description page URL
    "pipe_port": 1,              // target pipe port number
    "pipe_refresh_period": 100,  // pipe update period
    "pipe_element_size": 1,      // character or pipe element size
    "pipe_wrap_lines": 1,        // wrap console lines
    "font_name": "Courier New",  // font used in view
    "font_size": 8,              // font size
}
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

### 6.5.48  DefineRecorder

**Prototype:**

```
DefineRecorder ([in] name, [in] defString, [out] retMsg)
```

**Description:**

Defines the FreeMASTER Recorder object with all its properties. The script must use a JSON notation to describe all properties of the object. The Recorder object is created or modified if it already exists. All properties that are not defined in JSON are assigned a default value when creating a new object.

The following JSON example gives a brief reference of the definition record. It can also be retrieved by calling GetXxxDefinition. Note that the same record is used as a "tree_item" object within the clipboard text format when copying a recorder item in the "Project Tree" view.

```
{
    "name": "var16 recorder",    // recorder item name
    "href": "",                  // description page URL
    // recorder settings
    "rec_id": 0                  // target recorder instance to use
    "rec_time_base": 0,          // recorder time base multiplier (one less)
    "rec_total_samples": 70,     // total number of recorded samples
    "trig_pretrigger": 20,       // number of pre-trigger samples
    "trig_astop_ena": 0,         // stop automatically when timeout expires
```

```
    "trig_astop_delay": 10,      // automatic stop timeout
    "trig_aload": 1,             // load data automatically when stopped
    "trig_hold_ena": 1,          // hold loaded data in graph
    "trig_hold_delay": 1,        // minimum hold time
    "trig_arun": 1,              // run automatically after hold time
    "trig_zero_x": 1,            // show trigger point as 0 on X axis
    // graph settings
    "graph_buffer": 2000,        // graph buffer size in points
    "graph_type": 0,             // graph type: 0=Time, 1=XY
    "graph_legend_loc": 0,       // nonzero if legend is visible
    "graph_legend_vis": 1,       // nonzero to put legend to graph interior
    "graph_legend_inside": 1,    // exterior legend location: 0=top, 1=bottom, 2=left, 3=right
    "graph_grid_horz": 1,        // show graph horizontal grid
    "graph_grid_vert": 1,        // show graph vertical grid
    "graph_x_units": 2,          // X-axis time units: 0=us, 1=ms, 2=sec, 3=min (Time graph only)
    "graph_x_label": "Time",     // X-axis label (Time graph only)
    "graph_x_addunits": 1,       // non-zero to add time unit name to axis label (Time graph only)
    "graph_x_min": 0,            // nonzero to enable automatic X axis minimum value (XY graph only)
    "graph_x_max": 100,          // nonzero to enable automatic X axis maximum value (XY graph only)
    "graph_x_min_auto": 1,       // manual X axis minimum value (XY graph only)
    "graph_x_max_auto": 1,       // manual X axis maximum value (XY graph only)
    "auto_delete": 0,            // nonzero to set the Oscilloscope item to auto-delete mode
    // recorder variables
    "var_info": [                // array of graph variables
      {
        "var_def": {             // variable definition record
          "name": "var16",       // specifying 'name' is enough to locate existing variable
        },
        "visible": 1,            // variable visible in graph
        "y_block": 0,            // assign to y-block 0..4
        "color": 3026413         // subset color
        "trg_mode": 1,           // trigger mode 0=off, 1=edge, 2=level
        "trg_rising": 1,         // trigger on rising edge or level high
        "trg_falling": 0,        // trigger on falling edge or level low
        "trg_threshold": 100     // trigger threshold value
      },
      {
        "var_def": {
          "name": "var16inc",
        },
        "variable": "var16inc",
        "visible": 1,
        "y_block": 1,
        "color": 4688896
        "trg_mode": 0,
      }
    ],
    // graphs details for each Y-axis block
    "yblock_info": [             // array of Y-block definitions
      {
        "join_class": 0,         // blocks with equal value will be joined
        // left-axis settings
        "laxis_label": "16 bits", // label
        "laxis_style": 0,        // style
        "laxis_min_auto": 0,     // use automatic minimum scaling
        "laxis_max_auto": 1,     // use automatic maximum scaling
        "laxis_min": 0,          // minimum value (when auto=0)
        "laxis_max": 10,         // maximum value (when auto=0)
        // right-axis settings
        "raxis_label": "Increment", // label
        "raxis_style": 0,        // style
        "raxis_min": 0,          // use automatic minimum scaling
        "raxis_max": 10,         // use automatic maximum scaling
        "raxis_min_auto": 0,     // minimum value (when auto=0)
        "raxis_max_auto": 1,     // maximum value (when auto=0)
        "raxis_subsets": 1       // number of subsets to assign to right axis
      },                         // (counted from the last subset)
      {
        "join_class": 1,         // a different class means to separate this block
        "laxis_label": "lAxis",
        "laxis_style": 0,
        "laxis_min": -10,
        "laxis_max": 10,
        "laxis_min_auto": 1,
        "laxis_max_auto": 1,
        "raxis_label": "rAxis",
        "raxis_style": 0,
        "raxis_min": -10,
        "raxis_max": 10,
        "raxis_min_auto": 1,
        "raxis_max_auto": 1,
        "raxis_subsets": 0
      },
    ],
  }
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

### 6.5.49 DefineStimulator

**Prototypes:**

```
DefineRecorder ([in] name, [in] defString, [out] retMsg);
```

**Description**

Defines the FreeMASTER Stimulator object with all its properties. The script must use a JSON notation to describe all properties of the object. The Stimulator object is created or modified if it already exists. All properties that are not defined in JSON are assigned a default value when creating a new object.

The following JSON example gives a brief reference of the definition record. It can also be retrieved by calling GetXxxDefinition.

```
{
  "name":"stimulator-var16inc",
  "step_time_ms":100,
  "run_in_loop":1,
  "variable":{"name":"var16inc"},
  "points":[
  {"time":0,"value":"10"},
  {"time":3,"value":"30"},
  {"time":6,"value":"80"},
  {"time":9,"value":"99"}
  ]
}
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|----------|-------------|
| name | Name of the FreeMASTER stimulator item to be created or modified. |
| defString | ActiveX: string with a JSON-formatted definition<br>JSON-RPC: definition object itself |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|-----------|----------------|-----------------|-------------|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned on success. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

**JavaScript example:**

```
function define_my_stimulator()
{
  // stimulator target variable
  var stim_var = {
    "name": "var16inc",
  };
```

```
    // stimulator points
    var stim_points = [
      {"time": 0, "value": "10"},
      {"time": 3, "value": "30"},
      {"time": 6, "value": "80"},
      {"time": 9, "value": "99"}
    ];

    // stimulator definition
    var def = {
      "name": name,
      "step_time_ms": 100,
      "run_in_loop": 1,
      "variable": stim_var,
      "points": stim_points
    };

    pcm.DefineStimulator("my stimulator", JSON.strinigy(def));
}
```

### 6.5.50 DefineEnum

**Prototypes:**

```
DefineEnum ([in] name, [in] defString, [out] retMsg);
```

**Description**

Defines the FreeMASTER Enum object with all its properties. The script must use a JSON notation to describe all properties of the object. The Enum object is created or modified if it already exists. All properties that are not defined in JSON are assigned a default value when creating a new object.

The following JSON example gives a brief reference of the definition record. It can also be retrieved by calling GetXxxDefinition.

```
{
  "name":"my enum",
  "items":[
    {"name":"LOW",   "value_min":"10","value_max":"20", "inclusive_min":1,"inclusive_max":1},
    {"name":"NORMAL","value_min":"20","value_max":"80", "inclusive_min":0,"inclusive_max":1},
    {"name":"HIGH",  "value_min":"80","value_max":"100","inclusive_min":0,"inclusive_max":1}
  ]
}
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| name | Name of the FreeMASTER enum item to be created or modified. |
| defString | ActiveX: string with a JSON-formatted definition<br>JSON-RPC: definition object itself |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned on success. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error |

| | | | message. Otherwise, it is empty. |
|---|---|---|---|

**JavaScript example:**

```
function define_my_enum()
{
  // enum points
  var enum_items = [
    {"name": "LOW",   "value_min": "10", "value_max": "20",  "inclusive_min": 1, "inclusive_max": 1},
    {"name": "NORMAL","value_min": "20", "value_max": "80",  "inclusive_min": 0, "inclusive_max": 1},
    {"name": "HIGH",  "value_min": "80", "value_max": "100", "inclusive_min": 0, "inclusive_max": 1}
  ];

  // enum definition
  var def = {
    "name": "my enum",
    "items": enum_items
  };

  pcm.DefineEnum("my enum", JSON.strinigy(def));
}
```

### 6.5.51 DefineWatchBlock

**JSON-RPC prototype:**

```
DefineWatchBlock ([in] name, [in] defString, [out] retMsg)
```

**Description:**

Defines the FreeMASTER Block object in the project tree with its own definition of the "Variable Watch" view. The script must use a JSON notation to describe all properties of the object. The Block object is created or modified if it already exists. All properties that are not defined in JSON are assigned a default value when creating a new object.

The following JSON example gives a brief reference of the definition record. It can be also retrieved by calling GetXxxDefinition. Note that the same record is used as a "tree_item" object within the clipboard text format when copying a block in the "Project Tree" view.

```
{
    "name": "New Block",            // block item name
    "href": "",                     // description page URL
    // Variable Watch settings
    "show_grid": 1,                 // show grid
    "ena_row_sizing": 1,            // enable row sizing
    "ena_col_sizing": 1,            // enable column sizing
    "ena_col_swapping": 1,          // enable column swapping
    "ena_inplace_values": 1         // enable variable modifications
    // watched variables
    "watch_variables": [
      {
        "var_def": {                // variable definition record
          "name": "var16",          // specifying 'name' is enough to locate existing variable
        },
        "format": {                 // format of the watched variable
          "row_height": 20          // grid row size
          "cells": [                // array of up to 5 cells of the row
            {                       //
              "font_name": "Tahoma",  // font name
              "font_size": 12,        // font size
              "font_bold": 0,         // bold font
              "font_italic": 0,       // italic font
              "font_underline": 0,    // underlined font
              "alignment": 33,        // cell alignment 33=left,34=right,36=center
              "backcolor": 16777215,  // cell background color as hex BBGGRR
              "forecolor": 255        // cell foreground color
            },                      //
          ],                        //
        }                           //
      },                            //
      {                             // next watched variable
        "var_def": {                //
```

```
        "name": "var16inc",        //
      },                           //
      "format": {                  //
        "row_height": 20           //
        "cells": []                //
      }                            //
    },                             //
    {                              // next watched variable
      "var_def": {                 //
        "name": "var16rw",         //
      },                           //
      "format": {                  //
        "row_height": 20           //
        "cells": [],               //
      }                            //
    }                              //
  ],                               //
  "column_info": [                 // array of variable watch columns
    {                              // [0] = 'name' column
      "column_visible": 1,         // column visible
      "column_width": 120,         // column size
      "column_ix": 0               // column placement (may be swapped)
    },                             //
    {                              // [1] = 'value' column
      "column_visible": 1,         //
      "column_width": 120,         //
      "column_ix": 1               //
    },                             //
    {                              // [2] = 'unit' column
      "column_visible": 1,         //
      "column_width": 120,         //
      "column_ix": 2               //
    },                             //
    {                              // [3] = 'period' column
      "column_visible": 1,         //
      "column_width": 50,          //
      "column_ix": 3               //
    },                             //
    {                              // [4] = 'comment' column
      "column_visible": 0,         //
      "column_width": 200,         //
      "column_ix": 4               //
    }                              //
  ],                               //
  }                                //
}
```

**Compatibility:**

✔ ActiveX, ✔ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|----------|-------------|
| name | Name of the FreeMASTER oscilloscope item to be created or modified in the project tree. |
| defString | ActiveX: string with a JSON-formatted definition<br>JSON-RPC: definition object itself |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|-----------|----------------|-----------------|-------------|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned when the object was created or modified. "False" is returned if an error occurred. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

### 6.5.52 DefineArrayViewer

**JSON-RPC prototype:**

```
DefineOscilloscope ([in] name, [in] defString, [out] retMsg)
```

**Description:**

Defines the FreeMASTER Array Viewer object with all its properties. The script must use a JSON notation to describe all properties of the object. The Array Viewer object is created or modified if it already exists. All properties that are not defined in JSON are assigned a default value when creating a new object.

The following JSON example gives a brief reference of the definition record. It can be also retrieved by calling GetXxxDefinition. Note that the same record is used as a "tree_item" object within the clipboard text format when copying an array viewer item in the "Project Tree" view.

```
{
    "name": "New Array Viewer",  // array viewer item name
    "href": "",                  // description page URL
    // array viewer settings
    "array_count": 10,           // number of elements read from each array
    "trig_astop_ena": 0,         // stop automatically when timeout expires
    "trig_astop_delay": 10,      // automatic stop timeout
    "trig_aload": 1,             // load data automatically when stopped
    "trig_hold_ena": 1,          // hold loaded data in graph
    "trig_hold_delay": 1,        // minimum hold time
    "trig_arun": 1,              // run automatically after hold time
    "trig_mode": 2,              // 0=off,1=variable change,2=change&auto-clear
    "trig_var_def": {            // trigger variable definition record
      "name": "var16trig",       // specifying 'name' is enough to locate existing variable
    },
    // graph settings
    "graph_buffer": 2000,        // graph buffer size in points
    "graph_type": 0,             // graph type: 0=Time, 1=XY
    "graph_legend_loc": 0,       // nonzero if legend is visible
    "graph_legend_vis": 1,       // nonzero to put legend to graph interior
    "graph_legend_inside": 1,    // exterior legend location: 0=top, 1=bottom, 2=left, 3=right
    "graph_grid_horz": 1,        // show graph horizontal grid
    "graph_grid_vert": 1,        // show graph vertical grid
    "graph_x_units": 2,          // X-axis time units: 0=us, 1=ms, 2=sec, 3=min (Time graph only)
    "graph_x_label": "Time",     // X-axis label (Time graph only)
    "graph_x_addunits": 1,       // non-zero to add time unit name to axis label (Time graph only)
    "graph_x_min": 0,            // nonzero to enable automatic X axis minimum value (XY graph only)
    "graph_x_max": 100,          // nonzero to enable automatic X axis maximum value (XY graph only)
    "graph_x_min_auto": 1,       // manual X axis minimum value (XY graph only)
    "graph_x_max_auto": 1,       // manual X axis maximum value (XY graph only)
    "auto_delete": 0,            // nonzero to set the Oscilloscope item to auto-delete mode
    // viewed arrays
    "var_info": [                //
      {
        "var_def": {             // variable definition record
          "name": "arr16[0]",    // specifying 'name' is enough to locate existing variable
        },
        "visible": 1,            // variable visible in graph
        "y_block": 0,            // assign to y-block 0..4
        "color": 3026413         // subset color
      }
    ],
    // graph details for each Y-axis block
    "yblock_info": [
      {
        "join_class": 0,         // blocks with equal value will be joined
        // left-axis settings
        "laxis_label": "lAxis",  // label
        "laxis_style": 0,        // style
        "laxis_min": -10,        // use automatic minimum scaling
        "laxis_max": 10,         // use automatic maximum scaling
        "laxis_min_auto": 1,     // minimum value (when auto=0)
        "laxis_max_auto": 1,     // maximum value (when auto=0)
        // right-axis settings
        "raxis_label": "rAxis",  // label
        "raxis_style": 0,        // style
        "raxis_min": -10,        // use automatic minimum scaling
        "raxis_max": 10,         // use automatic maximum scaling
        "raxis_min_auto": 1,     // minimum value (when auto=0)
        "raxis_max_auto": 1,     // maximum value (when auto=0)
        "raxis_subsets": 0       // number of subsets to assign to right axis
      }                          // (counted from the last subset)
    ],
```

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**101 / 142**

```
    }
  }
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| name | Name of the FreeMASTER oscilloscope item to be created or modified in the project tree. |
| defString | ActiveX: string with a JSON-formatted definition<br>JSON-RPC: definition object itself |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned when the object was created or modified. "False" is returned if an error occurred. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

### 6.5.53  EnumHrefLinks

**Prototype:**

```
EnumHrefLinks ([in] index, [out] hrefName)
```

**Description:**

Enumerates the hyperlinks referenced by the target MCU application's Active Content. The caller script is expected to call this function with an increasing "index" value until an invalid "false" value is returned.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| index | Index of hyperlink to be retrieved. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | Hyperlink URL value in a string form when a hyperlink at a given "index" exists. Otherwise, a "false" boolean value is returned. |
| hrefName | LastLinkName | response.xtra.name | Hyperlink text name - the string to be displayed for the user. |

**JavaScript example:**

See the JavaScript example of using this function in the source code of the FreeMASTER Welcome page, which is displayed when FreeMASTER starts. The Welcome page detects and displays the hyperlinks contained by the target MCU Active Content feature.

### 6.5.54 EnumProjectFiles

**Prototype:**

```
EnumProjectFiles ([in] index, [out] prjName)
```

**Description:**

Enumerates the project files referenced by the target MCU application's Active Content. The caller script is expected to call this function with an increasing "index" value until an invalid "false" value is returned.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|----------|-------------|
| index | Index of the embedded project file to be retrieved. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|-----------|----------------|-----------------|-------------|
| return value | LastResult | response.xtra.retval | Project file URL in a string form when a project at a given "index" exists. Otherwise, a "false" value is returned. |
| prjName | LastLinkName | response.xtra.name | Project name - the string to be displayed to the user. |

**JavaScript example:**

See the JavaScript example of using this function in the source code of the FreeMASTER Welcome page which is displayed when FreeMASTER starts. The Welcome page detects and displays the hyperlinks contained by the target MCU Active Content feature.

### 6.5.55 PipeOpen

**Prototype:**

```
PipeOpen ([in] port, [in, optional] txBufferSize, [in, optional] rxBufferSize, [out] retMsg)
```

**Description:**

Initializes the FreeMASTER 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.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| port | The port number that identifies the Pipe channel. Only the lower 16 bits of the value are used. |
| txBufferSize | Local transmit buffer size. This buffer is used to accumulate the data being transmitted to the target MCU by calling a PipeWrite function until the MCU is ready to accept it.<br>Optional, defaults to 0x1000. |
| rxBufferSize | 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 a PipeRead function.<br>Optional, defaults to 0x1000. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned when the Pipe was successfully initialized. "False" is returned otherwise. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

### 6.5.56  PipeClose

**Prototype:**

```
PipeClose ([in] port)
```

**Description:**

De-initializes the Pipe object.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| port | The port number that identifies the Pipe channel. Only the lower 16 bits of the value are used. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned when the Pipe was successfully closed. "False" is returned otherwise. |

### 6.5.57  PipeFlush

**Prototype:**

```
PipeFlush ([in] port, [in] timeout_ms)
```

**Description:**

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

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| port | The port number that identifies the Pipe channel. Only the lower 16 bits of the value are used. |
| timeout_ms | 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. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.data | This function returns the number of bytes that remain in the local transmit buffer after this flush attempt. |

### 6.5.58 PipeSetDefaultTxMode

**Prototype:**

```
PipeSetDefaultTxMode ([in] txAllOrNothing)
```

**Description:**

Sets the default transmission mode for the subsequent calls to any PipeWrite functions.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| txAllOrNothing | Boolean value to be used as a default in the next call to any PipeWrite functions. |

**Outputs:**

None.

### 6.5.59 PipeSetDefaultRxMode

**Prototype:**

```
PipeSetDefaultRxMode ([in] rxAllOrNothing, [in] rxTimeout_ms)
```

**Description:**

Sets the default receiving mode for the subsequent calls to any PipeRead functions.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| rxAllOrNothing | Boolean value to be used as a default in the next call to any PipeRead functions. |
| rxTimeout_ms | Default timeout (in milliseconds) to be used by the subsequent PipeRead calls. |

**Outputs:**

None.

### 6.5.60 PipeSetDefaultStringMode

**Prototype:**

```
PipeSetDefaultStringMode ([in] unicode)
```

**Description:**

Sets the default string encoding used with the PipeWriteString and PipeReadString functions.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| unicode | Boolean value to be used as a default in the next call to the PipeWriteString and PipeReadString functions. |

**Outputs:**

None.

### 6.5.61 PipeGetRxBytes

**Prototype:**

```
PipeGetRxBytes ([in] port)
```

**Description:**

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

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| port | The port number which identifies the Pipe channel. Only the lower 16 bits of the value are used. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.data | Number of bytes that are ready in the local receive buffer. |

### 6.5.62  PipeGetTxBytes

**Prototype:**

```
PipeGetTxBytes ([in] port)
```

**Description:**

Returns the number of bytes pending in the local transmit buffer.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| port | The port number which identifies the Pipe channel. Only the lower 16 bits of the value are used. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.data | The number of bytes that are pending in the local transmit buffer to be delivered to the target MCU. |

### 6.5.63  PipeGetTxFree

**Prototype:**

```
PipeGetTxFree ([in] port)
```

**Description:**

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

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| port | The port number which identifies the Pipe channel. Only the lower 16 bits of the value are used. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.data | Free space in the local transmit buffer. |

FMSTERUG

**User guide**

All information provided in this document is subject to legal disclaimers.

**Rev. 4.8 — 18 September 2024**

© 2024 NXP B.V. All rights reserved.

Document feedback

**107 / 142**

### 6.5.64  PipeGetRxBufferSize

**Prototype:**

```
PipeGetRxBufferSize ([in] port)
```

**Description:**

Returns the local receive buffer size. This is the size specified when initializing the Pipe object with the PipeOpen call.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| port | The port number that identifies the pipe channel. Only the lower 16 bits of the value are used. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.data | Local receive buffer size. |

### 6.5.65  PipeGetTxBufferSize

**Prototype:**

```
PipeGetTxBufferSize ([in] port)
```

**Description:**

Returns the local transmit buffer size. This is the size specified when initializing the Pipe object with a PipeOpen call.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| port | The port number that identifies the Pipe channel. Only the lower 16 bits of the value are used. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.data | Local transmit buffer size. |

### 6.5.66  PipeWriteString

**ActiveX prototype:**

```
PipeWriteString ([in] port, [in] string, [in] charsToWrite, [in] allOrNothing, [in] unicode)
```

**JSON-RPC prototype:**

```
PipeWriteString ([in] port, [in] string, [in] allOrNothing, [in] unicode)
```

**Description:**

Writes the characters from an input string into the selected Pipe.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite - APIs not fully compatible

**Inputs:**

| Argument | Description |
|---|---|
| port | The port number that identifies the pipe channel. Only the lower 16 bits of the value are used. |
| string | The string to be written. |
| charsToWrite (ActiveX only) | The number of characters to be written to a pipe. This is an optional parameter. When omitted, the whole string is written. |
| allOrNothing | When non-zero, the PipeWrite function attempts to write all data to the local transmit buffer at once. When the data do not fit into the buffer, no data are transmitted.<br>This is an optional parameter, the default value is determined by the PipeSetDefaultTxMode function. |
| unicode | When non-zero, the individual characters are written in Unicode encoding, two bytes per character. The Unicode characters are always written atomically, there is no risk of sending a partial value.<br>This is an optional parameter; the default value is determined by the PipeSetDefaultStringMode function. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.data | This function returns the number of characters written. |

### 6.5.67  PipeWriteXxxArray

**ActiveX prototypes:**

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

```
PipeWriteIntArray ([in] port, [in] elemSize, [in] data, [in, optional] count, [in, optional] allOrNothing)
```

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

```
PipeWriteUIntArray ([in] port, [in] elemSize, [in] data, [in, optional] count, [in, optional] allOrNothing)
```

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

```
PipeWriteFloatArray ([in] port, [in] data, [in, optional] count, [in, optional] allOrNothing)
```

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

```
PipeWriteDoubleArray ([in] port, [in] data, [in, optional] count, [in, optional] allOrNothing)
```

**JSON-RPC prototypes:**

JSON-RPC functions do not have the "count" argument, the array size is inferred from the data automatically.

```
PipeWriteIntArray ([in] port, [in] elemSize, [in] data, [in, optional] allOrNothing)
PipeWriteUIntArray ([in] port, [in] elemSize, [in] data, [in, optional] allOrNothing)
PipeWriteFloatArray ([in] port, [in] data, [in, optional] allOrNothing)
PipeWriteDoubleArray ([in] port, [in] data, [in, optional] allOrNothing)
```

**Description:**

Translates an array of numbers passed by the caller to a block of bytes and writes 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.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite - APIs not fully compatible

**Inputs:**

| Argument | Description |
| --- | --- |
| port | The port number which identifies the pipe channel. Only the lower 16 bits of the value are used. |
| elemSize | The size of an array element in bytes. The total number of bytes to be written to the target can be calculated as *count * elemSize*. |
| data | The array of values to be written. |
| count (ActiveX only) | The number of elements to be written to the pipe. This is an optional parameter. When omitted, the whole array is written. |
| allOrNothing | When non-zero, the PipeWrite function attempts to write all data to the local transmit buffer at once. When the data do not fit to the buffer, no data are transmitted.<br>This is an optional parameter, the default value is determined by the PipeSetDefaultTxMode function. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
| --- | --- | --- | --- |
| return value | LastResult | response.data | This function returns the number of array elements successfully written. |

### 6.5.68 PipeReadString

**Prototype:**

```
PipeReadString ([in] port, [in] rxTimeout_ms, [in] charsToRead, [in, optional] allOrNothing, [in, optional] unicode, [out] retString)
```

**Description:**

Reads the characters from a selected pipe.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
| --- | --- |
| port | The port number that identifies the pipe channel. Only the lower 16 bits of the value are used. |

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**110 / 142**

| Argument | Description |
|----------|-------------|
| rxTimeout_ms | Time (in milliseconds) to wait for the requested number of characters to be received.<br>This is an optional parameter, the default value is determined by PipeSetDefaultRxMode. |
| charsToRead | 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. |
| allOrNothing | When non-zero and "charsToRead" 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.<br>This is an optional parameter. The default value is determined by the PipeSetDefaultRxMode function. |
| unicode | 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.<br>This is an optional parameter and the default value is determined by the PipeSetDefaultString Mode function. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|-----------|----------------|-----------------|-------------|
| return value | LastResult | response.xtra.retval | This function returns the number of characters read. |
| retString | LastPipe_data | response.data | This parameter variable receives the string. |

### 6.5.69  PipeReadXxxArray

**Prototypes:**

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

```
PipeReadIntArray ([in] port, [in] elemSize, [in] rxTimeout_ms, [in] count, [in] allOrNothing, [out] data)
```

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

```
PipeReadUIntArray ([in] port, [in] elemSize, [in] rxTimeout_ms, [in] count, [in] allOrNothing, [out] data)
```

To access a 4-byte floating-point variable:

```
PipeReadFloatArray ([in] port, [in] rxTimeout_ms, [in] count, [in] allOrNothing, [out] data)
```

To access an 8-byte floating-point variable:

```
PipeReadDoubleArray ([in] port, [in] rxTimeout_ms, [in] count, [in] allOrNothing, [out] data)
```

**Description:**

Read a requested count of integer or floating-point numbers from a pipe and return it to the caller as an array of integer or floating-point numbers.

For each call, there is an optional function which can be used in different scripting environments:

- PipeReadXxxArray returns data compatible with any scripting environment.
  - ActiveX: data are returned as a safe array of variants, which can be used in scripting languages, such as VB script, and in compiled languages, such as Visual Basic. Because the VB script engine does not handle variants encapsulating 2 and 4-byte unsigned integer types, this method converts such values to a floating-

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**111 / 142**

point format before returning. For JavaScript: the safearray must be converted to a JavaScript array using the toArray() method.

– JSON-RPC: data are returned as a native JSON array.

For ActiveX only, there are also other optional functions that can be used in different scripting environments:

- PipeReadXxxArray_v is the same as PipeReadXxxArray, except that it does not perform the VB script 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.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| port | The port number which identifies the pipe channel. Only the lower 16 bits of the value are used. |
| elemSize | The size of an array element in bytes. The total number of bytes to be read from a pipe can be calculated as *count * elemSize*. |
| rxTimeout_ms | Time (in milliseconds) to wait for the requested number of characters to be received. This is an optional parameter and the default value is determined by PipeSetDefaultRxMode. |
| count | 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. |
| allOrNothing | When true and the "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. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | This function returns the number of array elements read. |
| data | LastPipe_data | response.data | This parameter variable receives the resulting array. |

### 6.5.70 EnumVariables

**Prototype:**

```
EnumVariables ([in] index, [out] variableName)
```

**Description:**

Enumerate the FreeMASTER variables in the current project. The caller script is expected to call this function with an increasing "index" value until an invalid "false" value is returned.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| index | Index of the variable to be retrieved. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value when a valid value is being returned. "False" when the "index" does not map to any valid variable object and the enumeration should be stopped. |
| variableName | LastEnum_name | response.data | Variable name. |

### 6.5.71 EnumSymbols

**Prototype:**

```
EnumSymbols ([in] index, [out] symbolName)
```

**Description:**

Enumerate the target application symbols, as loaded in the current project. The caller script is expected to call this function with an increasing "index" value until an invalid "false" value is returned.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| index | Index of the symbol to be retrieved. |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value when a valid value is being returned. "False" when the "index" does not map to any valid symbol object and the enumeration should be stopped. |
| symbolName | LastEnum_name | response.data | Symbol name. |

### 6.5.72 GetDetectedBoardInfo

**Prototype:**

```
GetDetectedBoardInfo ()
```

**Description:**

Retrieves the information obtained during the initial FreeMASTER protocol handshake between the Host PC and the target MCU application. With the introduction of FreeMASTER communication protocol version 4, this function and the returned parameters got obsolete. Protocol V4 enables to define multiple Oscilloscope, Recorder, and Pipe objects which cannot be described by this legacy data structure.

Use one of the GetConfigParamXxx functions to retrieve the configuration parameters of protocol V4.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

None

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value when the board information was retrieved. "False" otherwise. |
| | LastBoardInfo _protVer | response.data. protVer | Protocol version byte. When value 4 (or higher) is returned, use GetConfigParamXxx to retrieve additional configuration values. |
| | LastBoardInfo _data BusWidth | response.data. data BusWdt | Data bus width, typically 1 on most MCU platforms. May be 2 on some DSC platforms. |
| | LastBoardInfo _version Word | response.data. globVer | FreeMASTER driver version word. |
| | LastBoardInfo _cmd BuffSize | response.data. cmd BuffSize | Size of the FreeMASTER serial communication buffer. |
| | LastBoardInfo _recBuff Size | response.data. recBuff Size | Size of the data buffer for Recorder use (protocol v4: Recorder #0 information only). |
| | LastBoardInfo _rec TimeBase | response.data. rec TimeBase | Base Recorder sample rate encoded in units described in the Serial Driver documentation for protocol v3. |
| | LastBoardInfo _ description | response.data. descr | Board or FreeMASTER driver description string. |

### 6.5.73 GetConfigParamXxx

**Prototypes:**

```
GetConfigParamU8 ([in] name, [out] paramValue) GetConfigParamULEB ([in] name, [out] paramValue) GetConfigParamString ([in] name, [out]
    paramValue)
```

**Description:**

Reads the configuration parameter from the target MCU application. With the introduction of FreeMASTER protocol version 4, this set of functions replaces the GetDetectedBoardInfo function.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite

**Inputs:**

| Argument | Description |
|---|---|
| name | String containing configuration parameter name. The list of default parameter names follows, an application may define custom parameters too.<br>U8 parameters:<br>• **F1** - Flags<br>• **RC** - Number of recorders implemented on target |

| Argument | Description |
|---|---|
| | • **SC** - Number of oscilloscopes implemented on target<br>• **PC** - Number of pipes implemented on target<br>ULEB parameters (ULEB is a general unsigned numeric value):<br>• **MTU** - Size of an internal communication buffer for handling command and response frames<br>• **BA** - Base address used by optimized memory read/write commands<br>String parameters:<br>• **VS** - Version string<br>• **NM** - Application name string<br>• **DS** - Description string<br>• **BD** - Build date/time string |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value if the configuration value was obtained successfully. "False" otherwise. |
| paramValue | LastConfigParam_ value | response.data | The actual parameter value in numeric or string formats. |

### 6.5.74 GetCommPortInfo

**Prototype:**

```
GetCommPortInfo ([in, optional] name)
```

**Description**

This function can be used to retrieve information about the communication port settings. FreeMASTER Lite clients may use this method to also determine information about the configured ELF file.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✓ Lite - APIs not fully compatible

**Inputs:**

| Argument | Description |
|---|---|
| name | Name identifying the connection. This parameter is optional for desktop FreeMASTER as there is only one valid connection configured at the moment. This built-in connection may also be identified by a special name "preset". |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | True when the board is detected and information retrieved. False otherwise. |
| | | response.data.name | Name of the connection. |
| | LastCommPortInfo _ description | response.data. description | Connection description text. |
| | LastCommPortInfo _ connectString | response.data. connect String | Full connection string used by FreeMASTER to open the port. |

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**115 / 142**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| | | response.data.elf | ELF file name used to load symbolic information. |
| | LastCommPortInfo _is Plugin | response.xtra.isPlugin | True when a communication plug-in is used. |
| | LastCommPortInfo _is RS232 | response.xtra.isRS232 | True when the native serial port or the USB-to-serial port is used. |
| | LastCommPortInfo _speed | response.xtra.speed | Serial baud rate used. Valid only if the isRS232 member is true. |
| | LastCommPortInfo _portNum | response.xtra.portHint | COM port number. Valid only if the isRS232 member is true. |
| | LastCommPortInfo _portHint | response.xtra.portNum | Hint word used to look up 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. |

### 6.5.75 EnableEvents

**Prototype:**

```
EnableEvents([in] enable)
```

**Description:**

This function must be called in the JSON-RPC interface of the FreeMASTER desktop application to activate the JSON-RPC events firing. Call this function to enable the handling of OnRecorderDone, OnCommPortStateChanged, OnBoardDetected, and OnVariableChanged events.

**Compatibility:**

✗ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| enable | True to enable the event firing by the JSON-RPC server. False to disable. |

**Outputs:**

Promise object resolves, but carries no value.

### 6.5.76 SetProjectOptions

**Prototype:**

```
SetProjectOptions(defString, refreshViews);
```

**Description:**

Sets project options. The script must use a JSON notation to describe all project options. The following JSON example gives a brief reference of the project options. It can also be retrieved by calling GetProjectOptions.

```
{
    "comm":{
        "rs232_conn":"RS232; port=COM3;speed=115200;",
```

```
        "port":2,                          // optional when rs232_conn is not specified
        "port_hint":"COM3",                // optional when rs232_conn is not specified
        "speed":115200,                    // optional when rs232_conn is not specified
        "plugin_use":0,
        "plugin_conn":"",                  // use plug-in connect string if plugin_use is 1
        "plugin_id":""                     // use plug-in CLSID in {} brackets
    },
    "html":{
        "control_page":"test.htm",         // control page HTML source
        "rendering_engine":2,              // 0=Internet Explorer, 1=Chromium, 2=Edge
        "auth_xobjects":[                  // list of authorized ActiveX objects for the Edge mode
            "Msxml2.DOMDocument.3.0"
        ]
    },
    "demomode":{
        "enable":0,                        // Enable demo mode after project is loaded next time
        "protect":0,                       // Protect demo mode with password
        "password":""                      // The password (must be provided also when un-protecting)
    }
}
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| defString | Stringified JSON object defining the project properties to be set. |
| refreshViews | Boolean value indicating if all active views must be updated after changing the project options. Use the "true" value for example when changing the control page or HTML engine selection option (note that the "calling" control page script is then reloaded immediately). |

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A boolean "true" value is returned on success. |
| retMsg | LastRetMsg | response.error.msg | When an error occurs, this value contains an error message. Otherwise, it is empty. |

## 6.5.77 GetProjectOptions

**Prototypes:**

```
GetProjectOptions();
```

## Description

Get project options as a stringified JSON record.

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

None

**Outputs:**

| Parameter | ActiveX access | JSON-RPC access | Description |
|---|---|---|---|
| return value | LastResult | response.xtra.retval | A string with a JSON record representing one or more project options. See the Set ProjectOptions() description for more information about the JSON items. |

### 6.5.78 SetMultiDimArrayRowMajorOrder

**Prototype**:

```
SetMultiDimArrayRowMajorOrder([in] rowOrder)
```

**Description:**

Sets the row-major order (the column-major order is used by default) used when accessing a multi-dimensional array passed as an argument in methods like WriteXXXArray() or PipeWriteXXXArray().

Different programming languages or libraries may use a different storage order for multi-dimensional arrays. The setting can be retrieved by GetMultiDimArrayRowMajorOrder().

**Compatibility**:

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

| Argument | Description |
|---|---|
| rowOrder | Boolean value "true" if the row-major order should be used when accessing a multi-dimensional array. "False" for the column-major order. |

**Outputs:**

None.

### 6.5.79 GetMultiDimArrayRowMajorOrder

**Prototype:**

```
GetMultiDimArrayRowMajorOrder([out] rowOrder)
```

**Description:**

Gets the storage order used when accessing a multi-dimensional array set by SetMultiDimArrayRowMajorOrder().

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Inputs:**

None.

**Outputs**:

| Argument | Description |
|----------|-------------|
| rowOrder | Boolean value "true" if the row-major order should be used when accessing a multi-dimensional array. "False" for the column-major order. |

## 6.6 ActiveX properties

The FreeMASTER ActiveX methods contain input and output parameters. Passing the input parameters is straightforward, but handling the output values may present an issue in languages like JavaScript. Such languages do not support the reception of output values, so you must omit them when calling the method. After the call is made, all output values can be fetched using different ActiveX properties, as described in the below table.

For each property in the FreeMASTER ActiveX interface, there is also the "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 the ActiveX object properties is not possible. For example, to read the "LastRetMsg" property, use the GetLastRetMsg() function in languages that are only able to call methods.

*Note:* *These properties are meaningless when using the JSON-RPC interface. The JSON-RPC server returns all output data in the response object, which makes it very easy for the client to process them.*

**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 the "retMsg" output parameter). |
| LastVariable_vValue | The value of the "numeric value" output parameter returned by the last *ReadVariable* method called. |
| LastVariable_tValue | The value of the "text value" output parameter returned by the last *ReadVariable* method called. |
| LastMemory_data | The array of values returned in the "arrData" output parameter by the last call to one of the *Read Memory* or *ReadXxxArray* methods. |
| LastRecorder_data | The array of values returned in the "arrData" output parameter by the last call to the *GetCurrent RecorderData* or *GetCurrentRecorderSeries* methods. |
| LastRecorder_serie Names | The array of values returned in the "arrSerieNames" output parameter by the last call to the *Get CurrentRecorderData* method. |
| LastRecorder_time BaseSec | The value of the "timeBaseSec" output parameter returned by the last call to the *GetCurrent RecorderData* method. |
| LastRecorder_state | The value of the "state" output parameter returned by the last call to the *GetCurrentRecorderState* method. |
| LastLocalFile_string | The "retString" text buffer returned by the last call to the *LocalFileReadString* method. |
| LastSymbolInfo_size | The "symSize" value returned by the last call to the *GetSymbolInfo* method. |
| LastSymbolInfo_addr | The "symAddr" value returned by the last call to the *GetSymbolInfo* method. |
| LastSymbolInfo_type | The "type" value returned by the last call to the *GetSymbolInfo* method. |
| LastMemberInfo_size | The "membSize" value returned by the last call to the *GetStructMember* method. |
| LastMemberInfo_offset | The "membOff" value returned by the last call to the *GetStructMember* method. |
| LastAddressInfo_name | The "symbolName" value returned by the last call to the *GetAddressInfo* method. |
| LastAddressInfo_type | The "symbolType" value returned by the last call to the *GetAddressInfo* method. |
| LastAddressInfo_exact | The "isExactMatch" value returned by the last call to the *GetAddressInfo* method. |

FMSTERUG
All information provided in this document is subject to legal disclaimers.
© 2024 NXP B.V. All rights reserved.

User guide
**Rev. 4.8 — 18 September 2024**
Document feedback

119 / 142

**Table 1. FreeMASTER ActiveX object properties**...*continued*

| Property name | Description |
|---|---|
| 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. |
| LastMultipleVariables_ json | JSON string returned by *ReadMultipleVariables* or *WriteMultipleVariables*. |

## 6.7 ActiveX and JSON-RPC events

Events are notification messages that FreeMASTER uses to inform the client about asynchronous changes of states or other events. Handling events is quite straightforward for the ActiveX and JSON-RPC interfaces.

For Internet Explorer and ActiveX, the most robust and simplest way to install an event handler is to use a VBScript subroutine which forwards the execution further to the VBScript or JavaScript functions. For example, to handle an "OnBoardDetected" event of a FreeMASTER ActiveX object named "pcm", use this subroutine:

```
<script language="VBScript">
Sub pcm_OnBoardDetected()
    ' call JavaScript or VBScript handler function
    my_handler_code()
End Sub
</script>
```

When using the FreeMASTER ActiveX object in the Microsoft Edge view, the FreeMASTER executes the same event handlers, but inside the JavaScript environment. The function prefix should match the FreeMASTER object id from the OBJECT element in the HTML code. The updated code would be:

```
<script>
function pcm_OnBoardDetected() {
 // my handler code
}
</script>
```

In JSON-RPC, the events are sent as notification messages from the server to the client. Different JSON-RPC client implementations may differ in event handling. The PCM wrapper object distributed with the FreeMASTER installation uses the "simple-jsonrpc-js" implementation, which makes it very easy to handle any server events. The PCM code simplifies the event handler installation even more. The user code assigns the handler function to the PCM object property named after the event itself. Keep in mind that for JSON-RPC events to be received, the EnableEvents method must be called first. For the OnBoardDetected event, the code may look like this:

```
function my_handler_code()
{
    console.log("event received");
}
// enable events to be received
pcm.EnableEvents().then(() => {
    // install the handler
    pcm.OnBoardDetected = my_handler_code;
});
```

The following sections describe the events generated by the FreeMASTER desktop application.

### 6.7.1 OnRecorderDone

**Prototype:**

```
OnRecorderDone()
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Description:**

This event is called when the active recorder or array finishes downloading new data. The data may then be retrieved using the GetCurrentRecorderData or GetCurrentRecorderSeries methods.

**Important:** The EnableEvents method must be called first to receive the events on the JSON-RPC interface.

### 6.7.2 OnCommPortStateChanged

**Prototype:**

```
OnCommPortStateChanged(portOpen)
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Description**

This event is called when the communication port is open (**portOpen** is *true*) or closed (**portOpen** is *false*).

**Important:** The EnableEvents method must be called first to receive the events on the JSON-RPC interface.

### 6.7.3 OnBoardDetected

**Prototype:**

```
OnBoardDetected()
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Description**

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

**Important:** The EnableEvents method must be called first to receive the events on the JSON-RPC interface.

### 6.7.4 OnVariableChanged

**Prototype:**

```
OnVariableChanged(varName, subscriptionId)
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Description**

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

**Important:** The EnableEvents method must be called first to receive the events on the JSON-RPC interface.

### 6.7.5 OnCustomEvent

**Prototype:**

```
OnCustomEvent(arg)
```

**Compatibility:**

✓ ActiveX, ✓ JSON-RPC, ✗ Lite

**Description**

This event is raised after any client connected to the same FreeMASTER ActiveX or JSON-RPC servers calls the FireCustomEvent method. The event provides the argument value specified by the caller.

**Important:** The EnableEvents method must be called first to receive the events on the JSON-RPC interface.

## 6.8 Scripting examples

The example applications shown in this section demonstrate how to initialize and use the ActiveX or JSON-RPC 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.

Refer to FreeMASTER Lite package for more JS examples on JSON-RPC usage, including Jupyter Notebooks containing NodeJS and Python examples.

### 6.8.1 VisualBasic script embedded in HTML page

The VisualBasic (VB) script used to be widely used scripting language used in web pages when Internet Explorer was a standard browser. This technology is not as popular today, because it is not supported by other web browsers, but it is still used by many users to provide active code behind locally rendered pages. The VB script is one of few scripting languages that natively supports by-reference [output] parameters returned from the ActiveX methods. Using a FreeMASTER object with the VB script is very straightforward.

```
<html>
<head>
    <script language="VBScript">
    Function onError(err)
        'Erors are reported in the status field.
        document.getElementById("status").innerHTML = err
    End Function
    Function read_variable(name, span_id)
        'ReadVariable uses FreeMASTER variable object from current project. Use
        'ReadUIntVariable to access the memory directly using a symbol name.
        bSucc = pcm.ReadVariable(name, vValue, tValue, bsRetMsg)
        If bSucc then
            document.getElementById(span_id).innerHTML = tValue
        else
            onError("Error when reading variable " & name & ". " + bsRetMsg)
        End If
    End Function
    Function read_array(name, elemSize, length, span_id)
        'Arrays are accessed in memory directly, using a symbol name and element size.
        bSucc = pcm.ReadUIntArray(name, length, elemSize, arr, bsRetMsg)
        If bSucc then
            document.getElementById(span_id).innerHTML = ""
            For i = 0 to uBound(arr)
                document.getElementById(span_id).innerHTML =
                    document.getElementById(span_id).innerHTML & arr(i) & ", "
            Next
        else
            onError("Error when reading variable " & name & ". " + bsRetMsg)
```

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**122 / 142**

```
                 End If
         End Function
         Function write_variable(name, input_id)
                 val = document.getElementById(input_id).value
                 'WriteVariable uses FreeMASTER variable object from current project. Use
                 'WriteUIntVariable to access the memory directly using a symbol name.
                 bSucc = pcm.WriteVariable(name, val, bsRetMsg)
                 If bSucc then
                     document.getElementById("status").innerHTML = "Write of the " & name & " succeeded."
                 else
                     onError("Error when reading variable " & name & ". " + bsRetMsg)
                 End If
         End Function
         </script>
</head>
<body>
        <!-- The main FreeMASTER ActiveX communication object -->
        <object id="pcm" height="0" width="0" classid="clsid:48A185F1-FFDB-11D3-80E3-00C04F176153">
        </object>
        <!-- User form -->
        Read var16 = <span id="var16_read">N/A</span> <input type="button" value="Read var16"
            onclick="call read_variable('var16', 'var16_read')" /> <br />
        Read var16inc = <span id="var16inc_read">N/A</span> <input type="button" value="Read var16inc"
            onclick="call read_variable('var16inc', 'var16inc_read')" /> <br />
        Read arr16 = <span id="arr16_read">N/A</span> <input type="button" value="Read arr16"
            onclick="call read_array('arr16', 2, 10, 'arr16_read')" /> <br />
        Write var16inc: <input type="text" id="var16inc_val" value="10" />
        <input type="button" value="Write var16inc"
            onclick="call write_variable('var16inc', 'var16inc_val')" /> <br />
        Status: <span id="status">No errors.</span> <br />
</body>
</html>
```

## 6.8.2 JavaScript and JSON-RPC embedded in HTML page

This example shows the use of the JSON-RPC communication in JavaScript, as introduced in FreeMASTER 3.0. This example works in the Chromium view embedded in the FreeMASTER main window, as well as in the Chrome web browser running externally.

```
<html>
<head>
    <!-- Get the  information object named "FreeMASTER" (needs version 3.1.3 and above) -->
    <script type="text/javascript" src="fmstr://localapp/info.js"></script>
    <!-- load JSON-RPC and FreeMASTER wrapper object -->
    <script type="text/javascript" src="./simple-jsonrpc-js.js"></script>
    <script type="text/javascript" src="./freemaster-client.js"></script>
    <script type="text/javascript">
    var pcm;  // the main FreeMASTER communication object
    function main()
    {
        /* Desktop FreeMASTER listens on port 41000 by default, unless this is
         * overridden on command line using /rpcs option. FreeMASTER Lite
         is configurable. */
        // default address
        var rpcs_addr = "localhost:41000";
        // freemaster 3.1.3 and above provides the info about itself:
        if(typeof(FreeMASTER) != "undefined")
        rpcs_addr = FreeMASTER.rpcs_addr;
        pcm = new PCM(rpcs_addr, on_connected, on_error, on_error);
        pcm.OnServerError = on_error;
        pcm.OnSocketError = on_error;
    }
    function on_connected()
    {
    }
    function on_error(err)
    {
        /* Erors are reported in the status field. */
        document.getElementById("status").innerHTML = err;
    }
    function read_variable(name, span_id)
    {
        /* ReadVariable uses FreeMASTER variable object from current project. Use
         * ReadUIntVariable to access the memory directly using a symbol name. */
        return pcm.ReadVariable(name)
            .then((value) => {
                document.getElementById(span_id).innerHTML = value.data;
            })
            .catch((err) => {
                on_error(err.msg);
            });
```

```
    }
    function read_array(name, elemSize, length, span_id)
    {
        /* Arrays are accessed in memory directly, using a symbol name and element size.  */
        pcm.ReadUIntArray(name, length, elemSize)
            .then((value) => {
                document.getElementById(span_id).innerHTML = "";
                for(i=0; i<value.data.length; i++)
                    document.getElementById(span_id).innerHTML += value.data[i] + ", ";
            })
            .catch((err) => {
                on_error(err.msg);
            });
    }
    function write_variable(name, input_id)
    {
        var val = document.getElementById(input_id).value;
        /* WriteVariable uses FreeMASTER variable object from current project. Use
         * WriteUIntVariable to access the memory directly using a symbol name. */
        pcm.WriteVariable(name, val)
            .then(() => {
                document.getElementById("status").innerHTML = "Write of the " + name + " succeeded.";
            })
            .catch((err) => {
                on_error(err.msg);
            });
    }
    </script>
</head>
<body onload="main()">
    <!-- User form -->
    Read var16 = <span id="var16_read">N/A</span> <input type="button" value="Read"
        onclick="read_variable('var16', 'var16_read')" /> <br />
    Read var16inc = <span id="var16inc_read">N/A</span> <input type="button" value="Read"
        onclick="read_variable('var16inc', 'var16inc_read')" /> <br />
    Read arr16 = <span id="arr16_read">N/A</span> <input type="button" value="Read"
        onclick="read_array('arr16', 2, 10, 'arr16_read')" /> <br />
    Write var16inc: <input type="text" id="var16inc_val" value="10" />
    <input type="button" value="Write"
        onclick="write_variable('var16inc', 'var16inc_val')" /> <br />
    Status: <span id="status">No errors.</span> <br />
</body>
</html>
```

## 6.8.3  JavaScript with ActiveX embedded in HTML page

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

• JavaScript does not natively support the by-reference [output] parameters. The parameters should be avoided and the output values should be retrieved from the LastResult and other LastXxx property members. See Section 6.6 for more details.

• JavaScript 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:

```
<html>
<head>
    <script type="text/javascript">
    function on_error(err)
    {
        /* Erors are reported in the status field. */
        document.getElementById("status").innerHTML = err;
    }
    function read_variable(name, span_id)
    {
        /* ReadVariable uses FreeMASTER variable object from current project. Use
         * ReadUIntVariable to access the memory directly using a symbol name. */
        if(pcm.ReadVariable(name))
            document.getElementById(span_id).innerHTML = pcm.LastVariable_vValue;
        else
            on_error("Error when reading variable " + name + ". " + pcm.LastRetMsg);
    }
    function read_array(name, elemSize, length, span_id)
    {
        /* Arrays are accessed in memory directly, using a symbol name and element size.  */
        if(pcm.ReadUIntArray(name, length, elemSize))
        {
            var rarr = pcm.LastResult ? pcm.LastMemory_data.toArray() : new Array();
            document.getElementById(span_id).innerHTML = "";
```

Document feedback

```
            for(i=0; i<rarr.length; i++)
                document.getElementById(span_id).innerHTML += rarr[i] + ", ";
        }
        else
        {
            on_error("Error when reading array " + name + ". " + pcm.LastRetMsg);
        }
    }
    function write_variable(name, input_id)
    {
        var val = document.getElementById(input_id).value;
        /* WriteVariable uses FreeMASTER variable object from current project. Use
         * WriteUIntVariable to access the memory directly using a symbol name. */
        if(pcm.WriteVariable(name, val))
            document.getElementById("status").innerHTML = "Write of the " + name + " succeeded.";
        else
            on_error("Error when writting variable " + name + ". " + pcm.LastRetMsg);
    }
    </script>
</head>
<body>
    <!-- The main FreeMASTER ActiveX communication object -->
    <object id="pcm" height="0" width="0" classid="clsid:48A185F1-FFDB-11D3-80E3-00C04F176153">
    </object>
    <!-- User form -->
    Read var16 = <span id="var16_read">N/A</span> <input type="button" value="Read var16"
        onclick="read_variable('var16', 'var16_read')" /> <br />
    Read var16inc = <span id="var16inc_read">N/A</span> <input type="button" value="Read var16inc"
        onclick="read_variable('var16inc', 'var16inc_read')" /> <br />
    Read arr16 = <span id="arr16_read">N/A</span> <input type="button" value="Read arr16"
        onclick="read_array('arr16', 2, 10, 'arr16_read')" /> <br />
    Write var16inc: <input type="text" id="var16inc_val" value="10" /> <input type="button"
        value="Write var16inc" onclick="write_variable('var16inc', 'var16inc_val')" /> <br />
    Status: <span id="status">No errors.</span> <br />
</body>
</html>
```

## 6.8.4 VisualBasic for Applications in Excel

The FreeMASTER object can also be used in the Visual Basic for Applications (VBA) project in Excel. Before writing the code, register a reference to the "FreeMASTER ActiveX Type Library" in the "Tools / References..." dialog, which is available in the VisualBasic development environment.
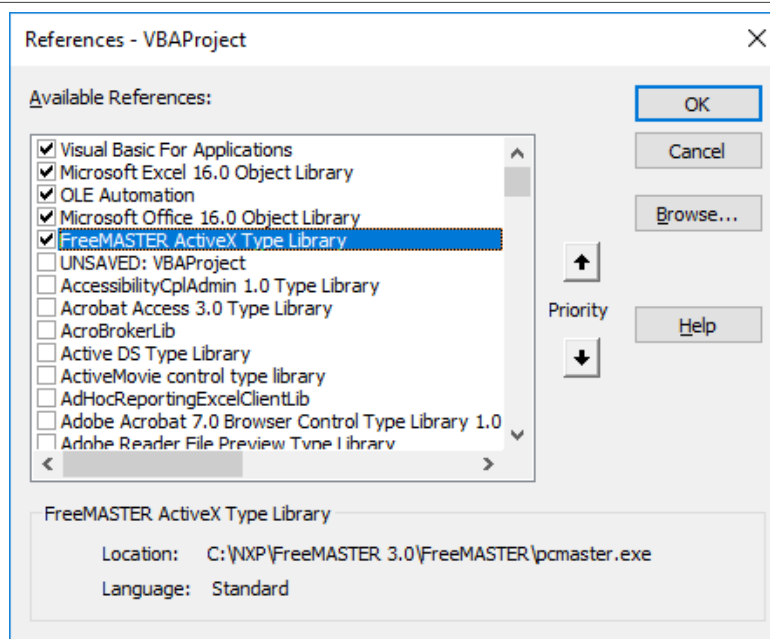


**Figure 52. Registering FreeMASTER Type Library in VBA**

The name of the FreeMASTER ActiveX object that should be declared in the VB code is `McbPcm`. If the type library is registered properly, the editor should automatically detect the FreeMASTER object and enable intelligent object method look-up while typing the code.
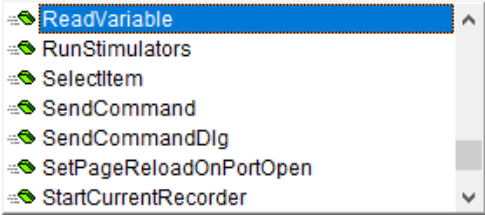


**Figure 53. Editing code in VBA environment**

```
Dim pcm As McbPcm
Dim sht
Private Sub Form_Load()
  'initialize FreeMASTER object
  Set pcm = New McbPcm
  Set sht = Worksheets("sheet1")
End Sub
Sub OnError(err)
    'Show error in status cell
    sht.Range("B7").Value = "ERROR: " & err
End Sub
Sub ShowStatus(text)
    'Show status in status cell
    sht.Range("B7").Value = text
End Sub
Sub ReadVar16()
    'Macro read var16
    Call ReadVariable("var16", "B1")
End Sub
Sub ReadVar16inc()
    'Macro read var16inc
    Call ReadVariable("var16inc", "B2")
End Sub
Sub ReadArr16()
    'Macro read arr16
    Call ReadArray("arr16", 2, 10, "B3:K3")
End Sub
Sub WriteVar16inc()
    'Macro write var16
    Call WriteVariable("var16inc", "B5")
End Sub
Sub ClearArray()
    'Macro clear cells for array
    sht.Range("B3:K3").Value = ""
    ShowStatus ("Array was cleared")
End Sub
Sub ReadVariable(name, cell)
    Dim bSucc As Boolean
    If pcm Is Nothing Then
        Call Form_Load
    End If
    'ReadVariable uses FreeMASTER variable object from current project. Use
    'ReadUIntVariable to access the memory directly using a symbol name.
    bSucc = pcm.ReadVariable(name, vValue, tValue, bsRetMsg)
    If bSucc Then
        sht.Range(cell).Value = tValue
        ShowStatus ("Readvariable OK")
    Else ' something failed
        OnError (bsRetMsg)
    End If
End Sub
Sub ReadArray(name, elemSize, length, cell)
    Dim bSucc As Boolean
    If pcm Is Nothing Then
```

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**126 / 142**

```
        Call Form_Load
    End If
    'Arrays are accessed in memory directly, using a symbol name and element size.
    bSucc = pcm.ReadUIntArray(name, length, elemSize, arr, bsRetMsg)
    If bSucc Then
        sht.Range(cell).Value = arr
        ShowStatus ("ReadUIntArray OK")
    Else ' something failed
        OnError (bsRetMsg)
    End If
End Sub
Sub WriteVariable(name, cell)
    Dim bSucc As Boolean
    Dim val As String
    If pcm Is Nothing Then
        Call Form_Load
    End If
    val = sht.Range(cell).Value
    'WriteVariable uses FreeMASTER variable object from current project. Use
    'WriteUIntVariable to access the memory directly using a symbol name.
    bSucc = pcm.WriteVariable(name, val, bsRetMsg)
    If bSucc Then
        ShowStatus ("Write of " & name & " was successful.")
    Else ' something failed
        OnError (bsRetMsg)
    End If
End Sub
```



**Figure 54. Excel form**

### 6.8.5 VBA and Excel cell functions

It is not always good to invoke Excel macros using active buttons inserted in the spreadsheet, as demonstrated in the previous section. This section demonstrates the use of Excel functions used inside the cells. The ready-to-use example is available in the "example_cell_functions.xlsm" file.

The variable names and values to be read or written are put to spreadsheet cells along with custom functions, named PCMVAR for reading, PCMWR for writing, and PCMFIRE for custom event processing.

Use the spreadsheet with FreeMASTER running the demo project attached to a target application. The "var8", "var16", "var32", "var16inc", and "var32inc" variables should be defined in the project. Exercise the function by modifying the values in the green cells. Also, pressing "F9" ("Formulas/Calculate Now" command) refreshes the PCMVAR functions and reads the immediate variable values from the target.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**127 / 142**
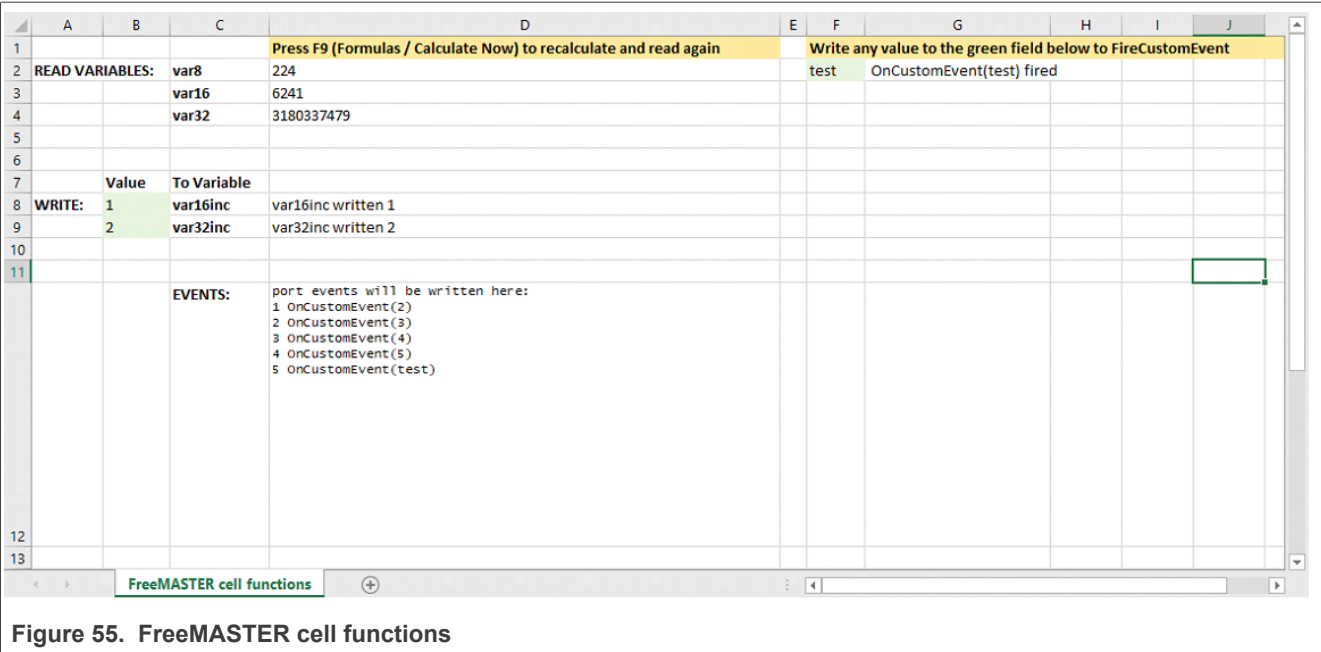
**Figure 55.  FreeMASTER cell functions**

The code in "ThisWorkbook" section provides a safe access to the FreeMASTER "pcm" object and installs the event handlers according to VBA standards:

```
Public WithEvents pcm As McbPcm
Private evCounter As Integer
Public Function GetPCM(reset As Boolean)
    If reset Then
        Set ThisWorkbook.pcm = Nothing
    End If
    If pcm Is Nothing Then
        Set ThisWorkbook.pcm = New McbPcm
    End If
    Set GetPCM = ThisWorkbook.pcm
End Function
Private Sub pcm_OnCustomEvent(ByVal vArg As Variant)
    evCounter = evCounter + 1
    Set r = Sheet1.Range("events")
    r.value = r.value & CStr(evCounter) & " OnCustomEvent(" & CStr(vArg) & ")" & vbNewLine
End Sub
Private Sub pcm_OnCommPortStateChanged(ByVal vPortOpen As Variant)
    evCounter = evCounter + 1
    Set r = Sheet1.Range("events")
    r.value = r.value & CStr(evCounter) & " OnCommPortStateChanged(" & CStr(vPortOpen) & ")" & vbNewLine
End Sub
Private Sub pcm_OnBoardDetected()
    evCounter = evCounter + 1
    Set r = Sheet1.Range("events")
    r.value = r.value & CStr(evCounter) & " OnBoardDetected" & vbNewLine
End Sub
```

The code in Module1 of the spreadsheet defines the cell functions:

```
Function PCMRESET()
    ThisWorkbook.GetPCM (True)
End Function
' call ReadVariable
Function PCMVAR(name)
    Dim pcm As McbPcm
    Set pcm = ThisWorkbook.GetPCM(False)
    Application.Volatile
    ok = pcm.ReadVariable(name, vval, tval, msg)
    If (ok) Then
        PCMVAR = tval
    Else
        PCMVAR = msg
    End If
End Function
' call WriteVariable
Function PCMWR(name, value)
```

```
    Dim pcm As McbPcm
    Set pcm = ThisWorkbook.GetPCM(False)
    ok = pcm.WriteVariable(name, value, msg)
    If (ok) Then
        PCMWR = name + " written " + CStr(value)
    Else
        PCMWR = msg
    End If
End Function
' call FireCustomEvent with an argument
Function PCMFIRE(arg)
    Dim pcm As McbPcm
    Set pcm = ThisWorkbook.GetPCM(False)
    ok = pcm.FireCustomEvent(arg)
    If (ok) Then
        PCMFIRE = " OnCustomEvent(" & CStr(arg) & ") fired"
    Else
        PCMFIRE = "Error when firing OnCustomEvent"
    End If
End Function
```

### 6.8.6 Matlab m-script

Test this code in the Matlab console:

```
% create FreeMASTER ActiveX object
pcm = actxserver ('MCB.PCM');
% write 10 value into "var16" variable
bSucc = pcm.WriteVariable('var16inc', 10);
var16 = 0;
% read the "var16" variable as defined in FreeMASTER project
bSucc = pcm.ReadVariable('var16');
if bSucc
    var16 = pcm.LastVariable_vValue;
end
% show the value
var16
% configure matlab to accept safe array as single dimension
feature('COM_SafeArraySingleDim', 1) ;
% write 4 bytes to 'arr8' array. WriteMemory function expects SafeArray as input data
bSucc = pcm.WriteMemory('arr8', 4, {11;22;33;44})
% reads 4 bytes from memory at address of var32inc variable
bSucc = pcm.ReadMemory('var32inc', 4);
if bSucc
    % reads data of last call the ReadMemory()function.
    readMemResult =  pcm.LastMemory_data
end
```

## 6.9 Script accessing multiple running FreeMASTER instances

Complex scripts or control pages can access multiple running FreeMASTER instances. This may be useful when more target boards are attached to a computer and a script must collect data from all of them.

For COM+/ActiveX interface:

- A special ActiveX object named "Multi-FreeMASTER Class" (identifier is MCB.MULT) enables you to run, stop, and control multiple instances of FreeMASTER application.
- The object can be instantiated in an Internet Explorer control page "<OBJECT id="mult" name="mult" height="1" width="1" classid="clsid:A8E26DF5-85A1-4552-AD0E-6C8AA10641EA"></OBJECT>".
- The object has just one method ("GetAppObject(instanceName, runMode, openProject)") which returns a common FreeMASTER object instance to be controlled using the API described earlier in this section.
- Parameters:
  - "instanceName" - a string identifier of the running instance. You can name instances using the "/sharex NAME" command-line parameter. When a required instance is not found, then a new FreeMASTER instance is started.
  - "runMode" - a number: 1 = normal, 2 = minimized, which can be added to: 0x10 = port closed, 0x20 = port open, 0x30 = port open verbose.
  - "openProject" - the project to open if an instance is not found and it is being started.

- As a result, a single "MCB.MULT" object can be used to run or attach to multiple named FreeMASTER instances. All instances can then be controlled from a single script.
- FreeMASTER version 3.1.3 extends the MCB.MULT object support to 64-bit applications, so even if FreeMASTER itself remains a 32-bit executable, any 64-bit client can fully use its functions.

For JSON-RPC interface:

- Run each FreeMASTER with a different "/rpcs PORT" command-line option to start the JSON-RPC server at a given port. The default port is 41000.
- When the "/rpcs" option is not used, other FreeMASTER instances pick a server port from a range of 41001..41016. If a script is running in an HTML code hosted in Chromium view inside FreeMASTER, the port number may be detected from a virtual JScript code declared as follows:

```
<script type="text/javascript" src="fmstr://localapp/info.js"></script>
```

For more information, see Section 6.3 and the example code in Section 6.8.2.
- In the script acting as a JSON-RPC client, you can connect to as many server instances as needed at proper ports and use the connection interface to control each of them.

## 6.10 FreeMASTER Internet Explorer migration notes

### 6.10.1 History

The Internet Explorer (IE) technology and the WebView control was one of the key building blocks of FreeMASTER since its very first version 1.0 released in year 2000. It enables HTML content to be displayed directly inside the FreeMASTER main window and to host JScript-powered active content inside it. This HTML view can be used to host interactive dashboards and other kinds of control pages with an access to the target MCU application runtime variables and other resources.

The technology behind the IE component hosting, rendering dynamic HTML, and JScript access to services provided by FreeMASTER is called Microsoft Component Object Model COM+ (sometimes also called ActiveX). The HTML code and scripts running inside IE component (or even in the standalone IE browser) create a special COM+/ActiveX object interface and access the FreeMASTER services by calling the object methods and by reading the object properties. The same services can also be consumed by other clients compatible with COM+, for example Microsoft Excel VBA script or Matlab scripts. Although it is not very precise technically, the FreeMASTER User Guide and other documentation often refer to this object as "FreeMASTER ActiveX Object", so the same term is used in the text below.

### 6.10.2 Why to change?

As the modern web browsers evolve towards standardization, the Internet Explorer and its ability to access COM+ and ActiveX components will soon become obsolete. Although FreeMASTER ActiveX object will still be supported and it will be possible to access it from clients like Matlab or Microsoft Excel, the Internet Explorer itself will no longer be usable on Microsoft Windows systems.

FreeMASTER started to address the upcoming change in the V3 version line with the first version 3.0 released in 2019. Version 3.0 adds a new browser component based on the Chromium engine. The new technology (which enables to host the Chrome-like browser inside the FreeMASTER main window) is known as Chromium Embedded Framework (CEF). New dashboards and control pages can now be rendered in the new CEF view, but the script code can no longer create instances of the FreeMASTER ActiveX object.

As a replacement for ActiveX, the FreeMASTER Version 3 has added a new JSON-RPC interface as an additional option to access its services. JSON-RPC is a text-based protocol running over HTTP, WebSocket, or a direct TCP connection and it is very well suitable to be used by scripts running inside the CEF, in any modern

web browser running standalone and also by other popular scripting frameworks (such as Node.JS or Python). The same protocol can be also used to access the cross-platform FreeMASTER Lite service.

### 6.10.3 Benefits and pitfalls of JSON-RPC

The JSON-RPC protocol is lightweight and easy to understand. FreeMASTER 3.x installation package contains a *freemaster-client.js* implementation of all details related to the WebSocket protocol transport and encapsulates the JSON-RPC service in a JScript module with an API very similar to the "legacy" FreeMASTER ActiveX object. The User Guide describes all methods and their compatibility with the ActiveX and JSON-RPC interfaces. The new JScript module can be used in HTML pages hosted in a CEF view running directly inside the FreeMASTER window as well as in any modern web browser running standalone on the same computer or even remotely over a network. A significant benefit of the JSON-RPC is how easy it is to set up a remote access comparing it to the traditional COM+/ActiveX technology.

The benefits of the JSON-RPC make it the preferred and recommended option for new designs, but it is not always easy to use it as a COM+/ActiveX replacement in existing FreeMASTER projects. Unlike the COM+, the JSON-RPC method execution is *asynchronous*, where the caller is not blocked during the object's method execution. The method results is processed just in time after it finishes, which enhances the script performance and responsiveness, but it may complicate the code design. The best way to handle this asynchronous nature of the new object API is by using programming techniques like JavaScript Promises, but it often requires a significant code rewrite when migrating from projects using the COM+/ActiveX approach.

The traditional COM+ and the FreeMASTER ActiveX object use *synchronous* method execution, which blocks the caller script execution until a called method finishes. With this approach, the code design is more straightforward, but it has also negative performance aspects, sometimes leading even to momentary freezing of the UI when a script keeps the caller's page blocked.

### 6.10.4 Microsoft Edge WebView2

The new Microsoft Edge WebView2 option introduced in FreeMASTER 3.2 brings together the best of the two existing options. The WebView2 component enables to host the Microsoft Edge view directly inside the FreeMASTER window, just like the IE or CEF. Microsoft Edge is based on Chromium engine so it is fully compatible with the FreeMASTER's JSON-RPC client interface. Additionally, Microsoft Edge also partially supports the COM+ host object embedding, which reenables the client script to access the FreeMASTER ActiveX object interface. The support of COM+ access is limited when compared to IE, but it still offers significantly easier way when migrating old IE-based projects to a modern browser framework.

### 6.10.5 Migrating FreeMASTER dashboards and scripts to WebView2

The Microsoft Edge WebView2 does not provide all COM+/ActiveX container capabilities known from IE. The following section describe the most important changes.

#### 6.10.5.1 FreeMASTER ActiveX object instantiation

With IE, the FreeMASTER object was typically instantiated using an OBJECT element put in the page HTML code:

```
<OBJECT> id="pcmObject" height="0" width="0" classid="clsid:48A185F1-FFDB-11D3-80E3-00C04F176153">
</OBJECT>
```

The object named *pcmObject* was created by the IE host application automatically as soon as the page was loaded. The magic class identifier is registered in the system to be the FreeMASTER interface object. Object methods can now be called as follows:

```
if(pcmObject.ReadVariable("var16"))
```

```
        alert("Variable value is " + pcmObject.LastVariable_vValue);
```

Note that by using a different class identifier, the OBJECT element can be used to create any other COM+ or ActiveX component registered in the Windows system and embed it in the page.

The WebView2 does not support the OBJECT element. The element tag can still be present in the page source but it is ignored by the Microsoft Edge engine. FreeMASTER uses a special method AddHostObjectToScript to inject a FreeMASTER ActiveX object to each HTML page it displays. The object is officially accessible as

```
window.chrome.webview.hostObjects.sync.pcm
```

On top of this, FreeMASTER uses the so-called 'prefix' script to be executed in each HTML page rendered by the WebView2 component. The script is accessible in the FreeMASTER installation in the *edge-prefix.js* file. The JScript code of this file is executed before any other script on any page is displayed in the FreeMASTER view. It scans the DOM model of the loaded page and tests the existence of legacy FreeMASTER OBJECT elements. When found, the object ID is retrieved from the element tag and a JScript alias variable is created referring to the artificially injected object.

For example, the OBJECT element defined above in the example of an IE page results in defining the following global JScript variable:

```
pcmObject = window.chrome.webview.hostObjects.sync.pcm;
```

The variable name on the left-hand side of the assignment is extracted from the OBJECT element ID, so the same syntax of accessing the object as in the original IE example is possible.

### 6.10.5.2  IE code changes needed when migrating to Edge WebView2

In combination, the FreeMASTER object injection into `window.chrome.webview.hostObject.sync` and the prefix script functionality nicely emulate a behavior of the legacy OBJECT element tag for the FreeMASTER object. When using old IE projects in the new Microsoft Edge WebView2 mode, it is very likely that the COM+ object interface will work as expected *without any change*.

Exceptions:

- Only the FreeMASTER ActiveX object is supported to be created using the OBJECT tag. No other COM+ or ActiveX objects are possible to be created this way.
- The object alias variable (e.g. pcmObject) exists only after the complete DOM model loads and the prefix script executes. It is not available in the top-level script being executed while the page loads. In this case, the full name of the object must be used as `window.chrome.webview.hostObjects.sync.pcm`.
- The alias variable is a common JScript variable and it may behave differently than the object identifier in the legacy IE pages. For example, it may be overwritten and assigned a different value.

### 6.10.6  FreeMASTER object events

The FreeMASTER ActiveX object also generates events that can be processed by the host script. For example, any change of a communication port state made manually or programmatically in the FreeMASTER application results in firing an `OnCommPortStateChanged` event in the script. More advanced example could be a situation where a target MCU application variable is "subscribed" using a method named `SubscribeVariable` so that any future change of the variable causes an `OnVariableChanged` event to be fired in the script.

### 6.10.6.1  Event handling in the IE

There were several ways how the legacy IE pages and their scripts installed the event handlers.

FMSTERUG

**User guide**

All information provided in this document is subject to legal disclaimers.

**Rev. 4.8 — 18 September 2024**

© 2024 NXP B.V. All rights reserved.

Document feedback

**132 / 142**

As the IE did support a VBscript syntax along with JScript, the event handlers were possible to be implemented as small VBscript subroutines calling further JScript code. In VBscript, any subroutine with a name defined as `objectId_eventName()` was registered as an event handler. This way was also recommended by old FreeMASTER documentation:

```
<script language="VBScript">
Sub pcmObject_OnVariableChanged(varName, nSubscriptionId)
' call JScript function
event_OnVariableChanged varName, nSubscriptionId
End Sub
Sub pcmObject_OnCommPortStateChanged(bOpen)
' call JScript function
event_OnCommPortStateChanged bOpen
End Sub
</script>
```

- Recent versions of IE did also support special syntax of the JScript function definition in the form of `objectId::eventName()`. Similarly to VBScript, the function was then automatically registered as an event handler. This is still recommended to be used with IE in the current documentation:

```
<script>
function pcmObject::OnCommPortStateChanged(bOpen) {
// event processing...
}
function pcmObject::OnVariableChanged(varName, nSubscriptionId) {
// event processing...
}
</script>
```

### 6.10.6.2 Event handling in Edge WebView2

Unfortunately, Microsoft Edge does NOT support event handling of COM+ object even for objects injected to the page with the `AddHostObjectToScript` call. Luckily, with the WebView2 component hosted inside the FreeMASTER main window, the event handler execution can be orchestrated directly from the FreeMASTER application code using the <u>ExecuteScript</u> call.

Thanks to the prefix JScript code described in the previous section, FreeMASTER is able to detect any FreeMASTER OBJECT tag and create an appropriate object alias. Later, upon any event fired for such 'injected' ActiveX object, FreeMASTER invokes the following JScript code in the page: `objectId_eventName();` - which causes the function to be invoked (if defined).

When such method exists in the page code, it gets executed so the behavior exactly matches the one in the legacy IE, except for the event handler naming. The `::` syntax is not allowed anymore in the Edge JScript engine, so the event handler uses the underscore character instead (the same naming is used in the legacy VBscript handlers, but now purely in the JScript environment).

### 6.10.6.3 IE code changes needed when migrating to Edge WebView2

Event handlers installed for FreeMASTER objects created with an OBJECT tag which were coded in VBscript or in JScript 'dual colon' `::` syntax must be renamed to use a single underscore separator, as follows:

```
<script>
function pcmObject_OnCommPortStateChanged(bOpen) {
// event processing...
}
function pcmObject_OnVariableChanged(varName, nSubscriptionId) {
// event processing...
}
</script>
```

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 4.8 — 18 September 2024

Document feedback

133 / 142

### 6.10.7  Creating other kinds of COM+/ActiveX objects

Some dashboard and other control page applications coded for the IE view in FreeMASTER often used also other 3rd party COM+ or ActiveX objects for various reasons:

- Invisible objects are only available for calculations or data processing (e.g. XML parsing, advanced mathematical calculations, S-record creation, etc.)
- True visible ActiveX UI controls implement advanced visualization and interactive interfaces like angular gauges, color buttons, sliders, graphs, etc.

There were two ways of how the objects could be added to the IE page:

- Using the OBJECT element tag with a proper class ID representing the object class. The class type needed to be installed and registered in the system before it could be used. There was an option for on-demand automated download and installation of the object but it was typically disabled due to security reasons.
- Dynamically in the JScript code using `new ActiveXObject(class_name)` statement. Also in this case, the object class must have been installed and registered in the system.

Modern browsers including Chrome and Edge do **not** support the ActiveX technology. Edge does support the object injection as described earlier, but neither of the browsers supports visual object embedding, OBJECT tag syntax nor the ActiveXObject constructor.

FreeMASTER 3.2 attempts to enable at least the minimum IE functionality in the hosted Edge WebView2 view:

- The OBJECT tag is artificially scanned by the injected prefix code and used to create FreeMASTER object aliases. This method is only usable with FreeMASTER objects.

Dynamic creation of the (invisible) object is possible via the FreeMASTER object. A new method named `CreateAuthorizedXObject(class_name)` has been added to the FreeMASTER object API and it enables you to create 3rd party objects registered in the system. For security reasons, the FreeMASTER project maintains a list of class names that are allowed to be created dynamically.

The FreeMASTER prefix code for Microsoft Edge view contains the following function:

```
function ActiveXObject(name) {
    console.log("ActiveXObject attempts to create new ActiveX object: " + name);
    return window.chrome.webview.hostObjects.sync.pcm.CreateAuthorizedXObject(name);
}
```

- Thanks to proper naming and JScript object constructor behavior, this function enables the user script to use the statement known from the IE code:

```
var myObject = new ActiveXObject("class_name");
```

### 6.10.7.1  IE code changes needed when migrating to Edge WebView2

Changes are only needed in case the IE pages used other 3rd party ActiveX objects.

- If objects were created dynamically using the `new ActiveXObject()` statement, make sure the object class name is added to the list of "authorized" object classes in the FreeMASTER project options.
- If objects were created using the OBJECT element tag, change the behavior to create the same objects dynamically using the `new ActiveXObject()` statement. This method can be used also in top-level script which is executed while the page is loaded.
- Be aware that ActiveX object creation is only possible in the Edge WebView2 view running inside the FreeMASTER main window (this is needed for all extended WebView2 features described in this document).

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**134 / 142**

### 6.10.8  Conclusion

FreeMASTER version 3.2 introduces the new Microsoft Edge WebView2 view as the 3rd option to render HTML pages and run JScript code. The existing Internet Explorer option is expected to be phased out in future versions and it may not work correctly on Microsoft Windows systems beyond June 2022.

Thanks to partial COM+ support in the Edge WebView2 and to extensions done by the container FreeMASTER application, most of the legacy IE functionality is usable in the Edge view with just minor changes required in the page script code.

All IE migration steps described in this document assume that the Edge WebView2 view runs embedded inside the FreeMASTER main application window. If there is a need to run the page in a standalone browser locally or remotely over a network, the JSON-RPC interface must be used. The use of the JSON-RPC interface is still a preferred way when creating new FreeMASTER projects. This document may be useful when migrating legacy projects (which use the IE view) into the new Edge WebView2 technology.

## 7  FreeMASTER low level

FreeMASTER enables users to monitor and control the target microcontroller application. The monitoring is done using a text-based Variable Watch grid or using the Oscilloscope and Recorder charts graphically. The basic control features are achieved using the Application Commands or by modifying the variables in the Watch grid. A bidirectional communication using FreeMASTER pipes is also available.

On top of this basic functionality, custom Control Pages can be written in HTML and JavaScript and use additional graphical objects (like buttons, gauges, sliders, and so on). Using the ActiveX or JSON-RPC interfaces, the Control Page has access to the FreeMASTER object and its programming interface. From the software layering perspective, the Control Page implements an upper layer built on top of the FreeMASTER foundation framework.

The Control Pages are not the only way of using the FreeMASTER technology for custom development. The FreeMASTER software itself is internally split to several layers which may be interfaced directly. Such a customization is definitely not trivial and may require some support from NXP. Reach out to the FreeMASTER community forum for more information.

### 7.1  Communication library

The communication library is a low layer of the FreeMASTER desktop application, which handles the entire communication with the target MCU application. It has a form of dynamically loaded library which may also be used by custom applications that use the FreeMASTER communication protocol. In Windows, the library is named `mcbcXX.dll`.

For FreeMASTER versions between 1.2 and 2.0, `mcbc12.dll` is used. FreeMASTER 2.5 and 3.0 introduce `mcbc30.dll`. The FreeMASTER Lite uses similar communication library updated for cross-platform operation with the same API.

The DLL can be interfaced from almost any programming languages that generate a native code for the Windows operating system and even several scripting languages. An interface header file and a *.lib* file for compile-time linking of the library is available in the FreeMASTER installation in the *userdev* directory.

- *mcbcom.h* - main communication library header file
- *mcberr.h* - error code definition header file

**Note:**  Obey the NXP licensing terms and conditions when designing custom applications with the FreeMASTER protocol or FreeMASTER communication library. The license only permits using the FreeMASTER tool with applications and systems that utilize NXP Semiconductors processors.

## 7.2 Communication plugins

Communication plugins are dynamically loaded libraries that may be used by FreeMASTER (namely by the FreeMASTER Communication Library) to implement alternate ways of physical communication with the target. The native serial communication protocol is implemented by the FreeMASTER communication library directly, using a low-level RS-232 or USB communication interface available in the operating system. The general routines to send protocol frames and receive response frames are provided in the library. Any other means of physical communication may be provided in the form of plugins which supply alternate send-and-receive implementations suitable for the selected physical media.

FreeMASTER comes with several communication plugins installed automatically along with the main application:

- The "CAN communication plug-in" uses CAN messages to send and receive serial protocol frames. The FreeMASTER driver must be configured for the CAN interface also in the target MCU application.
- The "BDM/JTAG communication plug-in" emulates the serial protocol, catches any variable-access protocol frames, and emulates the same behavior using a direct memory access over the JTAG interface. With this plugin, no protocol driver needs to be used in the target MCU application. All memory accesses are made on a background without any MCU CPU intervention. Advanced features (like Recorder or Pipes) do not work with this plugin, because there is no "cooperating" driver code in the MCU application.
- The "Packet-Driven BDM communication plug-in" uses the JTAG interface as a communication interface instead of a direct access to the target memory. In this case, the FreeMASTER driver must be again present in the target MCU application and must be configured for the PD-BDM communication.
- The "Network communication plug-in" supports direct TCP or UDP connectivity to the target embedded application. The application must use a TCP/IP stack (such as lwIP) and support the FreeMASTER protocol. See the "fmstr_net" and "fmstr_wifi" demo applications available in MCUXpresso SDK v2.10 (and later).
- The network plug-in also supports a communication over SEGGER J-Link debugger probes using an RTT protocol running locally or via a remote computer. The RTT communication may be used together with an active debugger session, sharing the same J-Link device.

The FreeMASTER desktop application and FreeMASTER Lite both support communication plugins, but the technology differs:

- FreeMASTER uses Microsoft COM+ technology to implement plug-ins and invoke their methods. The interface files needed to design custom communication plugins are available in the FreeMASTER installation in the *userdev* directory.
  - *mcbcom_i.idl* - the main COM+ interface definition file
  - *mcbcom_i.c and mcbcom_i.h* - defines the interface and related GUID identifiers
- FreeMASTER Lite uses general localhost RPC over a TCP protocol.

The development of custom plug-in modules may be a challenging task. Reach out to the NXP FreeMASTER community forum to seek support or to discuss your development plans.

***Note:*** *Obey the NXP licensing terms and conditions when designing custom applications with the FreeMASTER protocol or the FreeMASTER communication library. The license only permits using the FreeMASTER tool with applications and systems that utilize processors from NXP Semiconductors.*

## 8   References

- *FreeMASTER Usage Serial Driver Implementation* (document AN4752)
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document AN4771)
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document AN4860)
- FreeMASTER tool home: www.nxp.com/freemaster

- FreeMASTER community area: community.nxp.com/community/freemaster
- MCUXpresso SDK home: www.nxp.com/mcuxpresso
- MCUXpresso SDK builder: mcuxpresso.nxp.com/en

# 9 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# 10 Revision history

Table 2 summarizes the changes done to this document since the initial release.

**Table 2. Revision history**

| Revision | Date | Description |
|---|---|---|
| 4.8 | 18 September 2024 | Added Section 6.5.78 and Section 6.5.79. Updated Section 6.5.76. |
| 4.7 | 26 June 2023 | New and updated ActiveX/JSON-RPC API documented in (Section 6.5.34/Section 6.5.36). Added Figure 38 in Section 4.5.5. |
| 4.6 | 12/2022 | Accompanying the FreeMASTER version 3.2.1. New and updated ActiveX/JSON-RPC API documented in Section 6. |
| 4.5 | 06/2022 | Accompanying the FreeMASTER version 3.2.0. New Section 6.10. New ActiveX/JSON-RPC API documented in Section 6. |
| 4.4 | 03/2022 | Accompanying the FreeMASTER version 3.1.4. New ActiveX/JSON-RPC API documented in Section 6. |
| 4.3 | 10/2021 | Added information about RTT communication, MCB.MULT object, and CEF "info.js" virtual file. Accompanying the FreeMASTER 3.1.3 release. |
| 4.2 | 09/2020 | Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description. |

**Table 2. Revision history**...*continued*

| Revision | Date | Description |
|---|---|---|
| 4.1 | 04/2020 | Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8. |
| 4.0 | 04/2019 | Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms. |
| 3.0 | 08/2016 | Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. Added MCAN interface. Folders structure at the installation destination was rearranged. |
| 2.9 | 03/2015 | Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support |
| 2.8 | 06/2014 | Removed obsolete license text, see the software package content for up-to-date license. |
| 2.7 | 12/2013 | Added FLEXCAN32 interface, byte access and isr callback configuration option. |
| 2.5 | 08/2011 | Added Packet Driven BDM interface. |
| 2.4 | 06/2011 | Serial driver update, adds support for USB CDC interface. |
| 2.3 | 04/2011 | Added support for Kxx Kinetis platform and MQX operating system. |
| 2.2 | 04/2010 | Added support for MPC56xx platform, Added new API for use CAN interface. |
| 2.1 | 12/2007 | Added description of the new Fast Recorder feature and its API. |
| 2.0 | 09/2007 | Updated for FreeMASTER version. New Freescale document template used. |
| 1.0 | 03/2006 | Limited initial release |

# Legal information

## Definitions

**Draft** — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

## Disclaimers

**Limited warranty and liability** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

**Right to make changes** — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Terms and conditions of commercial sale** — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

**Export control** — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

**Suitability for use in non-automotive qualified products** — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

**HTML publications** — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

**Translations** — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

**Security** — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

**NXP B.V.** — NXP B.V. is not an operating company and it does not distribute or sell products.

## Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

**NXP** — wordmark and logo are trademarks of NXP B.V.

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**139 / 142**

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

**User guide**

**Rev. 4.8 — 18 September 2024**

Document feedback

**140 / 142**

# Contents

FMSTERUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 4.8 — 18 September 2024

Document feedback

141 / 142