

# **μC/Probe™** *Graphical Live Watch™*

## **User's Manual** **V3.3**

**Micrium**  
 **Press**

Weston, FL 33326

Micrium  
1290 Weston Road, Suite 306  
Weston, FL 33326  
USA

[www.micrium.com](http://www.micrium.com)

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where Micrium Press is aware of a trademark claim, the product name appears in initial capital letters, in all capital letters, or in accordance with the vendor's capitalization preference. Readers should contact the appropriate companies for more complete information on trademarks and trademark registrations. All trademarks and registered trademarks in this manual are the property of their respective holders.

Copyright © 2014 by Micrium except where noted otherwise. All rights reserved. Printed in the United States of America. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

$\mu$ C/Probe and the accompanying files are sold "as is". Micrium makes and customer receives from Micrium no express or implied warranties of any kind with respect to the software product, documentation, maintenance services, third party software, or other services. Micrium specifically disclaims and excludes any and all implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Due to the variety of user expertise, hardware and software environments into which  $\mu$ C/Probe may be subjected, the user assumes all risk of using  $\mu$ C/Probe. The maximum liability of Micrium will be limited exclusively to the purchase price.

# Table of Contents

<b>Chapter 1</b>	Introduction .....	6
<b>Chapter 2</b>	<b>μC/Probe System Overview</b> .....	9
<b>2-1</b>	<b>μC/Probe Data Client</b> .....	11
<b>Chapter 3</b>	<b>μC/Probe Symbol Browser</b> .....	15
<b>3-1</b>	<b>ELF File</b> .....	16
<b>3-1-1</b>	<b>Loading an ELF file</b> .....	16
<b>3-1-2</b>	<b>Browsing the ELF file</b> .....	16
<b>3-2</b>	<b>CDF File</b> .....	17
<b>3-2-1</b>	<b>Loading a CDF file</b> .....	18
<b>3-3</b>	<b>CSF File</b> .....	19
<b>3-3-1</b>	<b>Creating a CSF file</b> .....	19
<b>3-4</b>	<b>MQTT Configuration File</b> .....	23
<b>3-4-1</b>	<b>Creating an MQTT Configuration File</b> .....	24
<b>Chapter 4</b>	<b>μC/Probe Settings</b> .....	29
<b>4-1</b>	<b>General Settings</b> .....	30
<b>4-2</b>	<b>Communication Settings Overview</b> .....	31
<b>4-2-1</b>	<b>Debugger-based Interfaces</b> .....	31
<b>4-2-2</b>	<b>Peripheral-based Interfaces</b> .....	33
<b>4-2-3</b>	<b>Third Party Plugins</b> .....	35
<b>4-2-4</b>	<b>MQTT Interface</b> .....	36
<b>4-3</b>	<b>Communication Settings Window</b> .....	37
<b>4-3-1</b>	<b>Segger J-Link</b> .....	38
<b>4-3-2</b>	<b>CMSIS-DAP</b> .....	40
<b>4-3-3</b>	<b>Cypress PSoC Prog</b> .....	41
<b>4-3-4</b>	<b>USB</b> .....	42

---

<b>4-3-5</b>	TCP/IP .....	43
<b>4-3-6</b>	RS-232 .....	44
<b>Chapter 5</b>	<b>μC/Probe Workspace Explorer .....</b>	<b>45</b>
<b>Chapter 6</b>	<b>μC/Probe Toolbox .....</b>	<b>47</b>
<b>6-1</b>	Writable Controls .....	48
<b>6-2</b>	Linear Gauges .....	49
<b>6-3</b>	Horizontal Linear Gauges .....	49
<b>6-4</b>	Quadrant Gauges .....	50
<b>6-5</b>	Semicircle Gauges .....	50
<b>6-6</b>	Circular Gauges .....	51
<b>6-7</b>	Half Donuts .....	51
<b>6-8</b>	Cylinders .....	52
<b>6-9</b>	Charts .....	52
<b>6-10</b>	Numeric Indicators .....	53
<b>6-11</b>	LEDs .....	53
<b>6-12</b>	Advanced .....	54
<b>Chapter 7</b>	<b>μC/Probe Layout Design Tools .....</b>	<b>55</b>
<b>7-1</b>	μC/Probe Example .....	57
<b>Chapter 8</b>	<b>Associating Symbols to Virtual Controls and Indicators .....</b>	<b>58</b>
<b>Chapter 9</b>	<b>Run-Time Mode .....</b>	<b>61</b>
<b>9-1</b>	Run-Time Checklist .....	61
<b>9-2</b>	Running μC/Probe and your Debugging Software at the same time	62
<b>9-3</b>	IAR Systems C-SPY Plugin for μC/Probe .....	64
<b>9-3-1</b>	Configuring the TCP/IP Bridge between IAR C-SPY and μC/Probe	. 65
<b>Appendix A</b>	<b>Configuring Virtual Controls and Indicators .....</b>	<b>67</b>
<b>A-1</b>	Virtual Indicators .....	68
<b>A-2</b>	Virtual Controls .....	74
<b>A-3</b>	Charts .....	87

---

<b>Appendix B</b>	Kernel Awareness Screen .....	96
<b>Appendix C</b>	Terminal Window Control .....	99
<b>C-1</b>	Terminal Window Control Configuration .....	101
<b>C-2</b>	Properties Editor .....	102
<b>Appendix D</b>	µC/Trace Triggers Control .....	103
<b>Appendix E</b>	Spreadsheet Control .....	106
<b>E-1</b>	Adding an instance of the Spreadsheet Control .....	107
<b>E-2</b>	Configuring the Spreadsheet .....	108
<b>E-3</b>	Other Features .....	109
<b>E-4</b>	Application Example .....	110
<b>Appendix F</b>	Scripting Control .....	111
<b>F-1</b>	Writing a Script .....	111
<b>F-2</b>	Adding an Instance of the Scripting Control .....	115
<b>F-3</b>	Configuring the Scripting Control .....	116
<b>F-4</b>	Executing the Script .....	117
<b>Appendix G</b>	Data Logging Control .....	119
<b>Appendix H</b>	Licensing .....	123
<b>H-1</b>	Ordering .....	124
<b>H-2</b>	Activating .....	126
<b>Appendix I</b>	Bibliography .....	128
	Index .....	129

Introduction

$\mu$ C/Probe is a Windows application designed to read and write the memory of any embedded target processor during run-time. Memory locations are mapped to a set of virtual controls and indicators placed on a dashboard. Figure 1-1 shows an overview of the system and data flow.

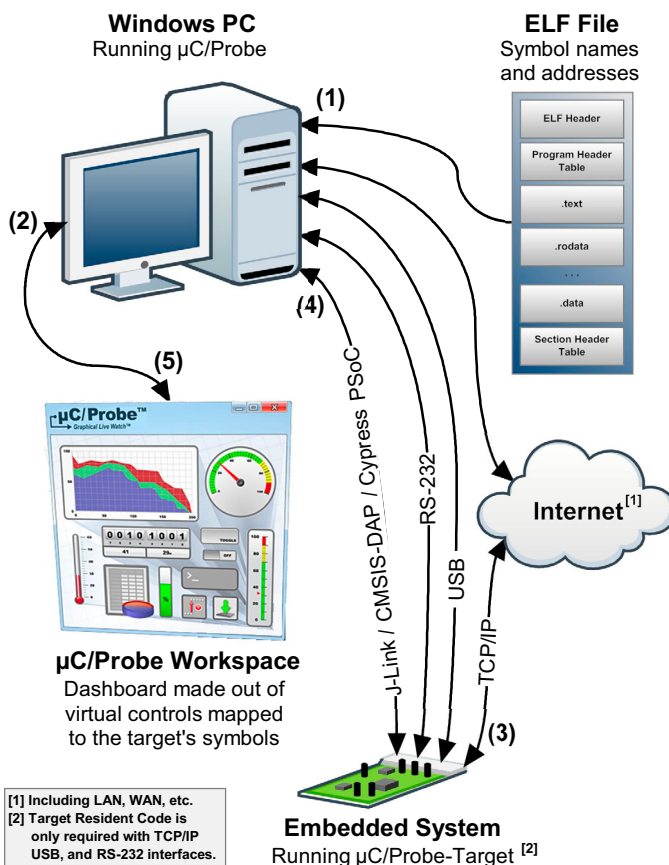


Figure 1-1  $\mu$ C/Probe Data Flow Diagram

---

F1-1(1) You have to provide  $\mu$ C/Probe with an ELF file with DWARF-2, -3 or -4 debugging information. The ELF file is generated by your toolchain's linker.  $\mu$ C/Probe parses the ELF file and reads the addresses of each of the embedded target's symbols (i.e. global variables) and creates a catalog known as *symbol browser*, which will be used by you during design-time to select the symbols you want to display on your dashboard. Refer to the document  $\mu$ C/Probe Target Manual for more information on installing the  $\mu$ C/Probe Target C files and building the ELF file.

Alternatively, you can also provide a chip definition file that contains the chip's peripheral register addresses or provide your own custom XML based symbol file for those cases when your toolchain cannot generate one of the supported ELF formats.

F1-1(2) During design-time, you create a  $\mu$ C/Probe workspace using a Windows PC and  $\mu$ C/Probe. You design your own dashboard by dragging and dropping virtual controls and indicators onto a *data screen*. Each virtual control and indicator needs to be mapped to an embedded target's symbol by selecting it from the symbol browser. This document aims at providing more information on creating your own dashboard with  $\mu$ C/Probe.

F1-1(3) Before proceeding to the run-time stage,  $\mu$ C/Probe needs to be configured to use one of the following communication interfaces: J-Link, CMSIS-DAP, Cypress PSoC Prog, USB, RS-232 or TCP/IP. In order to start the run-time stage, you click the *Run* button and  $\mu$ C/Probe starts making requests to read the value of all the memory locations associated with each virtual control and indicator (i.e. buttons and gauges respectively). At the same time,  $\mu$ C/Probe sends commands to write the memory locations associated with each virtual control (i.e. buttons on a click event).

F1-1(4) In the case of a reading request, the embedded target responds with the latest value. In the case of a write command, the embedded target responds with an acknowledgement. Refer to the document  $\mu$ C/Probe Target Manual for more information on all you need in regards to the firmware that implements the communication interface that runs on the embedded target.

F1-1(5)  $\mu$ C/Probe parses the responses from the embedded target and updates the virtual controls and indicators.

---

In case the communication of your choice is USB, RS-232 or TCP/IP, refer to the document  $\mu$ C/Probe Target Manual for more information about the firmware that resides on the Embedded System.

This document only provides information about the Windows PC side of the system.



# Chapter 2

## μC/Probe System Overview

This section provides an overview of the μC/Probe Windows Application.

Whenever you start μC/Probe in your Windows PC, three different modules are started: μC/Probe Automatic Updates and Licensing System, μC/Probe Data Client and μC/Probe Generic Target Communications Module as illustrated in Figure 2-1:

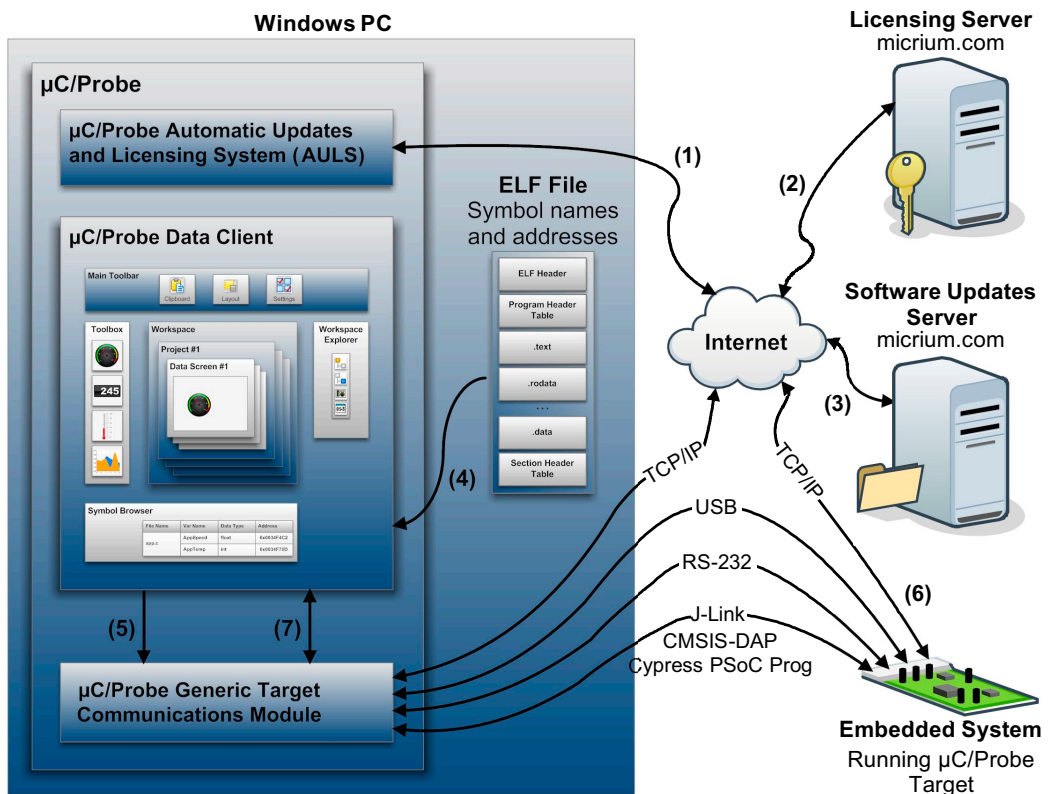


Figure 2-1 μC/Probe System Overview

---

F2-1(1) The Automatic Updates and Licensing System (AULS) is the part of the application that allows you to install and keep your  $\mu$ C/Probe application up to date.

F2-1(2) The Educational Edition of  $\mu$ C/Probe is deployed with some of the Basic and Professional Editions' features and do not require internet access to activate the software application. The Basic and Professional Edition of  $\mu$ C/Probe require internet access to validate the license key provided by your Micrium's sales representative.

For more information on  $\mu$ C/Probe Licensing see Appendix I, "Licensing" on page 123.

F2-1(3) All Editions of  $\mu$ C/Probe are self-updating and every time you start the application, if internet access is available, the  $\mu$ C/Probe AULS module checks for newer versions of  $\mu$ C/Probe from the Micrium website and as they become available, the  $\mu$ C/Probe AULS module, automatically replaces any updated files.

F2-1(4) The  $\mu$ C/Probe Data Client is the part of the application that allows you to design your dashboard (design-time mode) and run it (run-time mode).

The next section in this document provides more information in regards to using the  $\mu$ C/Probe Data Client during design-time and run-time.

F2-1(5) The  $\mu$ C/Probe Generic Target Communications Module is the part of the application that connects directly with the Embedded Target and responds to the requests from the Data Client.

When the  $\mu$ C/Probe run-time mode gets started, the Data Client sends requests to the Generic Target Communications Module. The requests contain not only the embedded target communication settings but also all the symbol's memory address required by your dashboard design.

F2-1(6) The Generic Target Communications Module takes the request from the  $\mu$ C/Probe Data Client and initiates a communication with the embedded target through the configured communication interface.

F2-1(7) The μC/Probe Data Client exchanges requests to read and write the memory locations required by the current view of your dashboard's design with the embedded target through the Generic Target Communications Module.

## 2-1 μC/PROBE DATA CLIENT

The μC/Probe Data Client is illustrated in more detail in Figure 2-2:

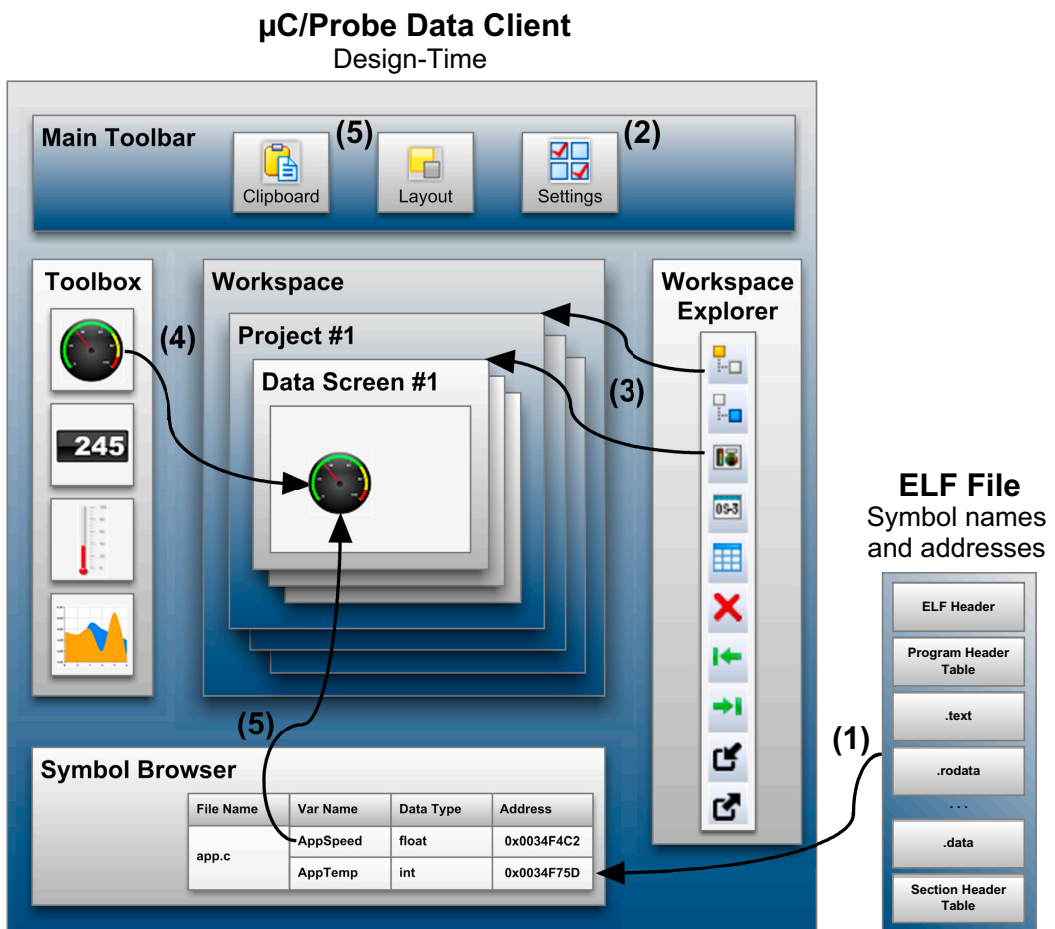


Figure 2-2 μC/Probe Data Client: Design Time

F2-2(1) The μC/Probe Data Client is the part of the application that during design-time takes the ELF file with DWARF-2, -3 or -4 debugging information. As previously discussed, the ELF file is generated by your toolchain's linker. The μC/Probe Data Client parses the ELF file and reads the addresses of each of the embedded target's symbols (i.e. global variables) and creates a catalog known as symbol browser, which will be used by you during design-time to select the symbols you want to display on your dashboard. Refer to the document μC/Probe Target Manual for more information on installing the μC/Probe Target C files and building the ELF file.

For more information, see Chapter 3, "μC/Probe Symbol Browser" on page 15.

F2-2(2) During design-time it is necessary to adjust the communication and other general settings. See Chapter 4, "μC/Probe Settings" on page 29 for more information on configuring μC/Probe.

F2-2(3) The Workspace Explorer in the μC/Probe Data Client allows you to add or delete Projects and Data Screens among other things.

For more information, see Chapter 5, "μC/Probe Workspace Explorer" on page 45.

F2-2(4) The μC/Probe Toolbox displays icons for the virtual controls and indicators that you can add to your Data Screens. Each toolbox icon can be dragged and dropped onto the Data Screen to build your own dashboard.

For more information, see Chapter 6, "μC/Probe Toolbox" on page 47.

F2-2(5) The μC/Probe Layout Design Tools help you arrange the virtual controls and indicators on your data screen by speeding up the creation of your dashboard and making it look great.

For more information, see Chapter 7, "μC/Probe Layout Design Tools" on page 55.

F2-2(6) The last step during design-time is to map each virtual control and indicator in your Data Screen with an Embedded Target's memory location. The symbol browser allows you to quickly find the variable you want to display and then all you have to do is drag the variable from the symbol browser and drop it onto the virtual control or indicator of your choice.

See Chapter 8, “Associating Symbols to Virtual Controls and Indicators” on page 58 for more information on using the symbol browser to map virtual controls and indicators to the embedded target's memory locations.

The actual μC/Probe windows application is shown in Figure 2-3:

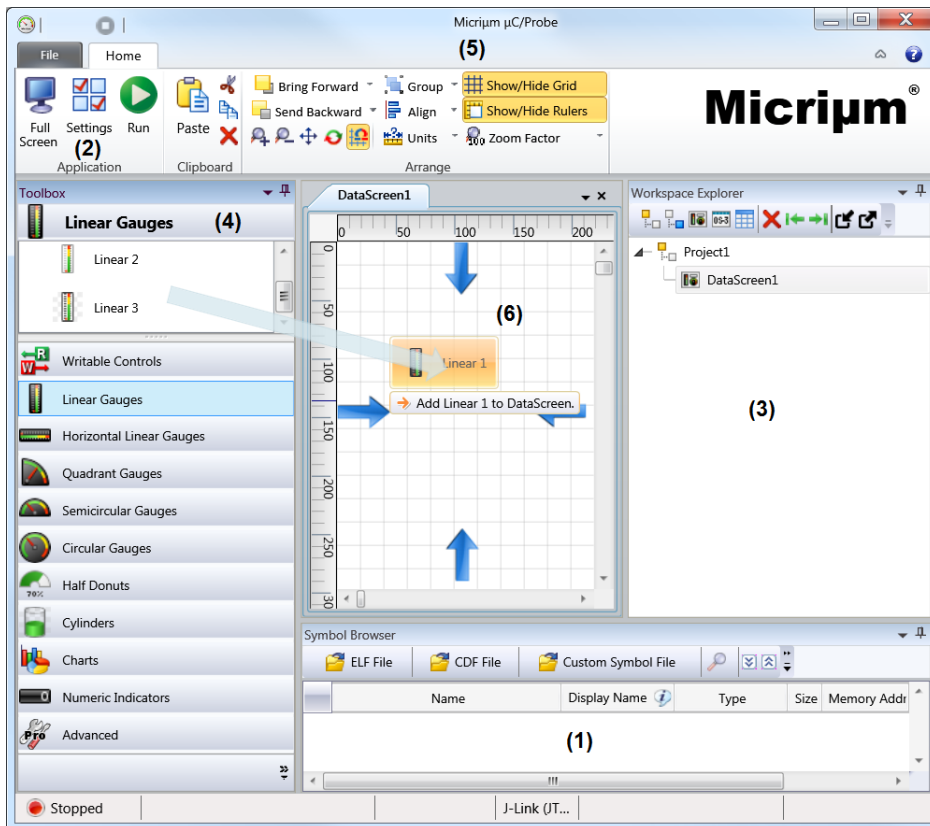


Figure 2-3 μC/Probe Windows Application

F2-3(1) Chapter 3, “μC/Probe Symbol Browser” on page 15.

- F2-3(2) Chapter 4, “μC/Probe Settings” on page 29.
- F2-3(3) Chapter 5, “μC/Probe Workspace Explorer” on page 45.
- F2-3(4) Chapter 6, “μC/Probe Toolbox” on page 47.
- F2-3(5) Chapter 7, “μC/Probe Layout Design Tools” on page 55.
- F2-3(6) Chapter 8, “Associating Symbols to Virtual Controls and Indicators” on page 58.

## µC/Probe Symbol Browser

The µC/Probe's Symbol Browser is a list of your embedded target's symbols that helps you quickly find the symbol you want to use in your data screen. The symbol browser is available during design-mode and it is located at the bottom of the application window.

There are four types of symbol files supported by µC/Probe's parser:

- ELF (Executable and Linkable Format).
- CDF (Chip Definition File).
- CSF (Custom Symbol File).
- MQTT Configuration File (Message Queueing Telemetry Transport).

To load any of these files, you start by clicking one or more of the buttons indicated in Figure 3-1:

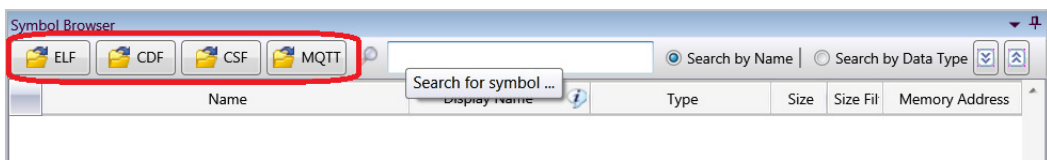


Figure 3-1 µC/Probe Symbol Browser: Loading a symbols file

The following sections will describe each of the four types of symbol files supported by µC/Probe. Keep in mind that you only require to load at least one of the following files.

## 3-1 ELF FILE

This file is the output from your compiling and linking process. It contains the name, data type and address of all your global variables.

### 3-1-1 LOADING AN ELF FILE

In order to provide  $\mu$ C/Probe with the path of the embedded target's output file (ELF file), locate and click the button labeled *ELF File* indicated in Figure 3-1.

$\mu$ C/Probe will prompt for the ELF file path by using a standard open file dialog. Locate the output file in your PC. The path is usually configured from your toolchain's linker configuration.

The ELF file needs to have symbolic information for debugging purposes in the DWARF-2, -3 or -4 format.

### 3-1-2 BROWSING THE ELF FILE

$\mu$ C/Probe parses the ELF file and creates a catalog that you can browse to search for the symbol you need. The symbol browser is a five-column tree list of symbols grouped by the C file name where the variable was declared as shown in Figure 3-2:

(7)	(2) Name	(3) Display Name	(4) Type	(5) Size	(6) Memory Address
>	LiquidLevelController.out	N/A	N/A	0	N/A
	app.c		N/A	0	N/A
	AppInflow	Liquid Inflow	unsigned char	1	0x20006110
	AppLevelActual	Actual Liquid Level	unsigned short	2	0x200060EE
	AppLevelDesired	AppLevelDesired	unsigned short	2	0x200060EC
	AppLevelMax	AppLevelMax	unsigned short	2	0x200060E8
	AppLevelMin	AppLevelMin	unsigned short	2	0x200060EA
	AppLevelUnits	AppLevelUnits	unsigned char	1	0x2000610F
	AppOutflow	AppOutflow	unsigned char	1	0x20006111
	AppValveInPct	AppValveInPct	unsigned char	1	0x20006112

Figure 3-2  $\mu$ C/Probe Symbol Browser: Symbols grouped by C file



- 
- F3-2(1) The symbol browser allows you to quickly find the symbol you want. Click on the symbol browser headers row to sort the list by the column you want. You can also expand and collapse tree nodes to focus on a particular C file, or you can use the search box and search by symbol name or data type.
- F3-2(2) The **Name** column shows the name of the symbol as declared in your C file.
- F3-2(3) The **Display Name** column by default displays the name of the symbol as declared in your C file, but also allows you to create an alias for the symbol. Double-click over the Display Name cell to create an alias.
- F3-2(4) The **Type** column displays the symbol's C data type.
- F3-2(5) The **Size** column displays the size in bytes of the symbol.
- F3-2(6) The **Memory Address** column displays the symbol's location in the embedded target's memory.
- F3-2(7) Click on the red **X** next to the name of the ELF file, to remove a symbol file from the symbol browser.
- F3-2(8) The *expand all* and *collapse all* buttons allow you to browse more efficiently throughout the symbol browser tree.

Be aware that the symbol browser in  $\mu$ C/Probe will detect if the ELF file has been re-compiled and will refresh all addresses. However, in case you move the ELF file to a different location in your file system,  $\mu$ C/Probe cannot update the addresses automatically. Instead, you can update the symbol browser with the new path by first removing the ELF file (red X next to the filename) and then opening the new ELF file.

## 3-2 CDF FILE

The CDF (Chip Definition File) contains the name, data type and address of all your device's I/O registers.  $\mu$ C/Probe installs in your PC with a very large catalog of chip definition files that includes chips from semiconductors such as Analog Devices, Atmel, Cypress, Freescale, Infineon, Renesas, ST, Texas Instruments and Xilinx among others.

### 3-2-1 LOADING A CDF FILE

You can browse the CDF catalog and select your platform's chip definition file by clicking the button *CDF File* and using the CDF browser shown in Figure 3-3:

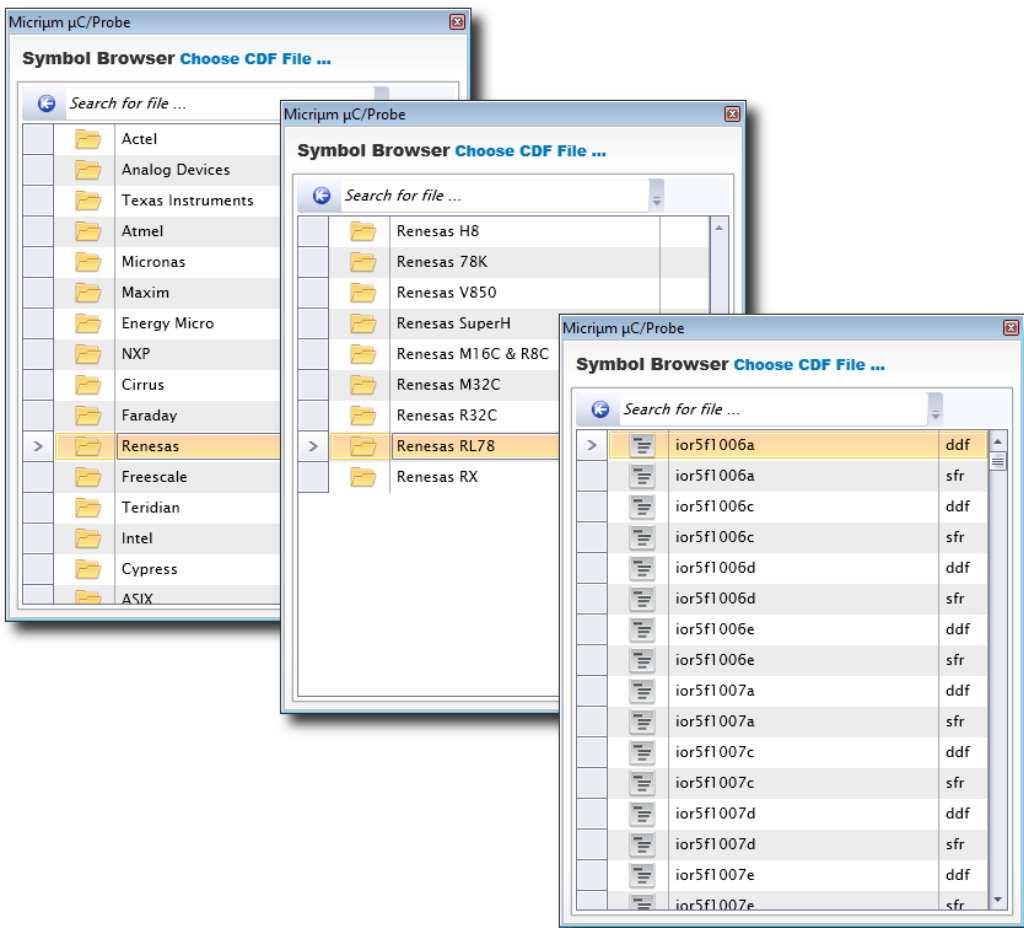


Figure 3-3 Chip Definition Files Browser

The CDF files provide the chip's peripheral I/O register names and addresses which you can use to associate with controls such as the *Bit Control* described in section A-2-8 "Bit Control Properties Editor" on page 82.

## 3-3 CSF FILE

$\mu$ C/Probe is capable of parsing XML-based Custom Symbol Files (CSF), which is very useful for those cases where your toolchain is incapable of generating one of the ELF file formats supported by  $\mu$ C/Probe.

### 3-3-1 CREATING A CSF FILE

The best way to create a CSF file is by modifying the template located in your  $\mu$ C/Probe installation directory at:

```
$\Micrium\uC-Probe\Templates\uC-Probe-CSF-Template.csf
```

The template is associated with an XSD document that defines the XML schema for CSF files supported by  $\mu$ C/Probe. We recommend using an XML editor capable of providing *IntelliSense* features such as Visual Studio, shown in Figure 3-4:

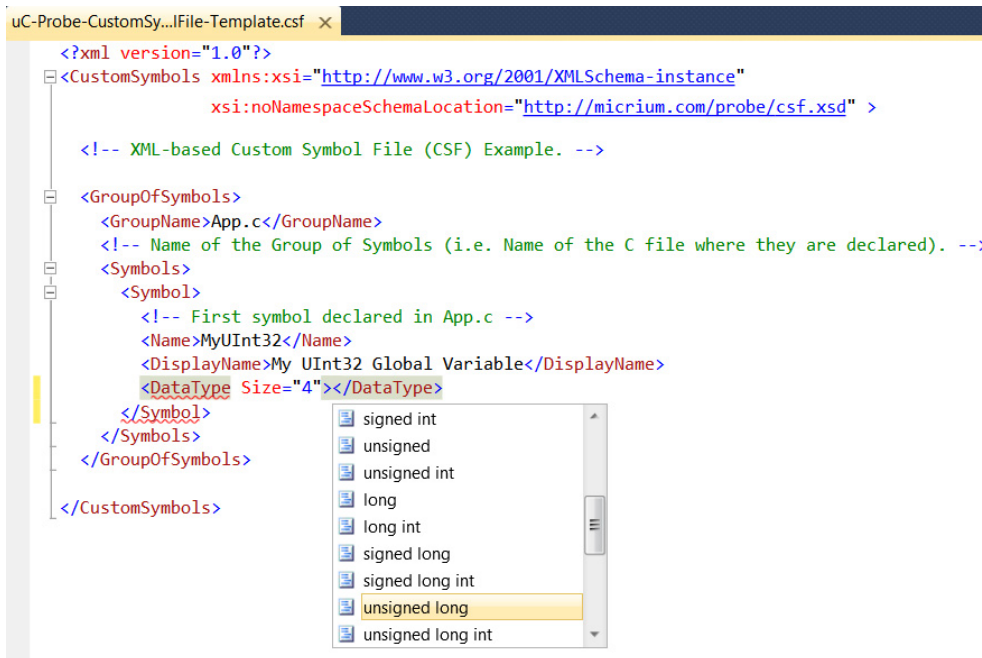


Figure 3-4 Creating a CSF in Visual Studio: Drop-Down Lists

Visual Studio makes editing your CSF file easier by filling required XML syntax for you. For example, after a schema is associated with your CSF, you get a drop-down list of expected elements any time you type "<".

When you type *SPACE* from inside a start tag, you also get a drop-down list showing all attributes that can be added to the current element.

Likewise, when you type "=" for an attribute value, or the opening quote for the value, you also get a list of possible values for that attribute.

Moreover, *ToolTips* appear on these IntelliSense lists giving you a description of each element as illustrated in Figure 3-5:

```

uC-Probe-CustomSy...|File-Template.csf ×
<?xml version="1.0"?>
<CustomSymbols xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://micrium.com/probe/csf.xsd"
  <!-- XML-based Custom Symbol File (CSF) Example. -->
  <GroupOfSymbols>
    <GroupName>App.c</GroupName>
    <!-- Name of the Group of Symbols (i.e. Name of the C file where they are declared). -->
    <Symbols>
      <Symbol>
        <!-- First symbol declared in App.c -->
        <Name>MyUInt32</Name>
        <DisplayName>My UInt32 Global Variable</DisplayName>
        <DataType Size="4">unsigned long</DataType>
        <MemoryAddress>0x10F4</MemoryAddress>
      </Sym
      Memory address in either decimal or hexadecimal format (0x1234).
    </Sym
  </GroupOfSymbols>
</CustomSymbols>

```

Figure 3-5 **Creating a CSF in Visual Studio: Tool Tips**

Custom Symbol Files need to have the extension `.csf` for  $\mu$ C/Probe to recognize them as such. Listing 3-1 shows an example of a CSF file that declares one integer, one array and one data structure.

```

<?xml version="1.0"?>
<CustomSymbols xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://micrium.com/probe/csf.xsd" >      (1)
  <GroupOfSymbols>                                                       (2)
    <GroupName>App.c</GroupName>                                         (3)
    <Symbols>
      <Symbol>
        <Name>MyUInt32</Name>                                             (4)
        <DisplayName>My UInt32 Global Variable</DisplayName>           (5)
        <DataType Size="4">unsigned long</DataType>                     (6)
        <MemoryAddress>0x10F4</MemoryAddress>                           (7)
      </Symbol>
      <Symbol>
        <Name>MyStruct</Name>
        <DataType Size="50">struct</DataType>
        <MemoryAddress>0x50AC</MemoryAddress>
        <DataMembers>                                                    (8)
          <Symbol>
            <Name>MyArray</Name>
            <DataType Size="16" IsArray="true" ArrayLength="4">int</DataType> (9)
            <MemoryAddress>0x50AC</MemoryAddress>
          </Symbol>
          <Symbol>
            <Name>MyCharPointer</Name>
            <DataType Size="4" IsPointer="true">char</DataType>         (10)
            <MemoryAddress>1025</MemoryAddress>
          </Symbol>
        </DataMembers>
      </Symbol>
    </Symbols>
  </GroupOfSymbols>
</CustomSymbols>

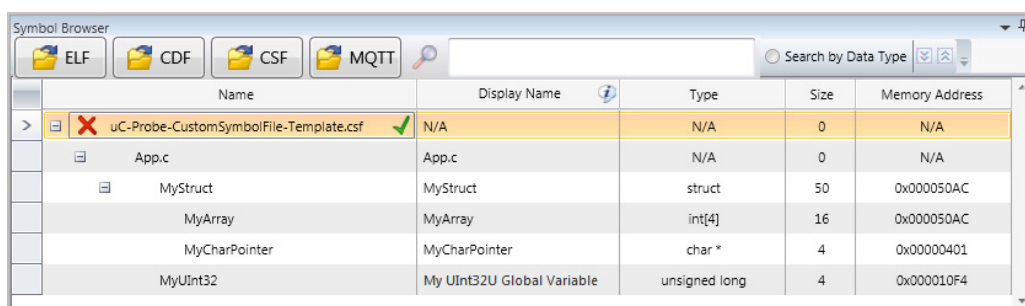
```

Listing 3-1 XML-based Custom Symbol File Example

- L3-1(1) The root element is called `<CustomSymbols>`. It includes the XSD schema reference to help you editing in an XML editor such as Visual Studio.
- L3-1(2) Each symbol or group of symbols are within the element tag `<GroupOfSymbols>`. In this case it is used with the purpose of creating a group of symbols declared in one single C file.
- L3-1(3) The name of the group of symbols is `App.c`.
- L3-1(4) The name of the first symbol in the group as declared in C.

- L3-1(5) You can also specify an alias for the group name for display purposes.
- L3-1(6) The element tag `<DataType>` is the ANSI C data type of the variable including the size in bytes as an attribute.
- L3-1(7) The element tag `<MemoryAddress>` is the variable's memory address in either decimal or hexadecimal format (0x1234).
- L3-1(8) For more complex symbols such as data structures, there is a tag called `<DataMembers>` that allows you to specify a group of symbols that make part of a data structure.
- L3-1(9) In order to declare an array, you need to specify three attributes: A boolean flag that indicates that the symbol is an array, the total number of bytes and the number of elements in the array.
- L3-1(10) Finally, any data type can be declared as a pointer by using the data type boolean attribute `IsPointer`. In this case, it is the intention to specify a symbol declared as `char *`.

In order to verify your CSF file, you can use the Symbol Browser from within  $\mu$ C/Probe as shown in Figure 3-6. Notice the relationship between the XML tags and the tree nodes in the Symbol Browser.:



Name	Display Name	Type	Size	Memory Address
uC-Probe-CustomSymbolFile-Template.csf	N/A	N/A	0	N/A
App.c	App.c	N/A	0	N/A
MyStruct	MyStruct	struct	50	0x000050AC
MyArray	MyArray	int[4]	16	0x000050AC
MyCharPointer	MyCharPointer	char *	4	0x00000401
MyUInt32	My UInt32U Global Variable	unsigned long	4	0x000010F4

Figure 3-6 XML-based Custom Symbol File Example as seen from  $\mu$ C/Probe's Symbol Browser

### 3-4 MQTT CONFIGURATION FILE

The MQTT (Message Queuing Telemetry Transport) protocol is becoming the de facto standard for IoT (Internet of Things) applications.

$\mu$ C/Probe can be used to build an MQTT Client. If you have an embedded system that is MQTT-ready, you can monitor and control your embedded system remotely by using any of the virtual controls and indicators in  $\mu$ C/Probe's toolbox.

Let's take for example an embedded systems-based *Weather Station* that remotely publishes via MQTT its readings to an MQTT broker as illustrated in Figure 3-7.

The MQTT broker stores the readings and then you can use  $\mu$ C/Probe to create a GUI that displays the readings using some of the virtual indicators (e.g. thermometer to display temperature) and configures the Weather Station with some of the virtual controls (e.g. slider control to configure the sampling rate).

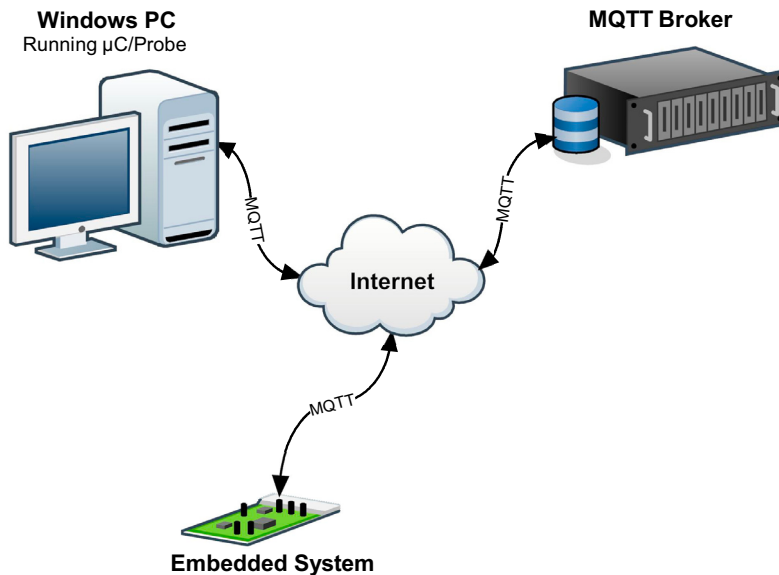


Figure 3-7  $\mu$ C/Probe as an MQTT gateway

The details on how to setup an MQTT broker and how to make an embedded systems MQTT-ready are beyond the scope of this document.

However, a good place to start is looking at 2lemetry as an MQTT broker and Micrium's  $\mu$ C/TCP-IP and  $\mu$ C/MQTTc as the firmware you need to make your embedded systems MQTT-ready.

The next section will explain how to create an MQTT configuration file assuming that you already have an account and configuration setup at the MQTT broker of your choice.

### **3-4-1 CREATING AN MQTT CONFIGURATION FILE**

Similar to the Custom Symbol File (CSF) in section 3-3 "CSF File" on page 19, the best way to create an MQTT configuration file is by modifying the template located in your  $\mu$ C/Probe installation directory at:

```
$(Micrium\uC-Probe\Templates\uC-Probe-MQTT-CfgFile-Template.mqtt
```



The template is associated with an XSD document that defines the XML schema for MQTT configuration files supported by  $\mu$ C/Probe. We recommend using an XML editor capable of providing *IntelliSense* features such as Visual Studio, shown in Figure 3-4:

The screenshot shows the Visual Studio editor with the file `uC-Probe-MQTT-CfgFile-Template.mqtt*` open. The XML content is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<!--
  XML-based Message Queue Telemetry Transport (MQTT) Configuration File Example.
-->
<Brokers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLoca

<!-- User 1 Account at 2lemetry.com -->
<Broker
  HostName="q.m2m.io"
  Port="1883"
  ClientID="any_unique_string_1"
  UseSSL="false"
  KeepAlive="30"
  Username="user_1@domain.com"
  Password="password_1"
  SendPasswordWithoutHashing="false"
  >

<!-- Topic -->
<Topic WriteTo="com.micrium/WeatherMeterCfg/33076" ReadFrom="com.micrium/WeatherMet

  <!-- JSON Format -->
  <Payload>
    <!-- Read/Write variables -->
    <Variable Name="SamplingRate"
    <!-- Read-only variables -->
    <Variable Name="RelativeHumidity"
    <Variable Name="Temperature"
    <Variable Name="Pressure"
    <Variable Name="WindSpeed"
    <Variable Name="WindDirection"
    <Variable Name="Rainfall"
    <Variable Name="UVIndex"

  </Payload>
</Topic>
```

IntelliSense is active, showing drop-down lists for several elements:

- For `<Variable Name="SamplingRate"`, the `DataType` list includes: char, signed char, unsigned char, short, unsigned short, **int** (selected), unsigned int, long, unsigned long. The description for `int` is "Samples per second."
- For `<Variable Name="RelativeHumidity"`, the `DataType` list includes: char, signed char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long. The description for `int` is "r's relative h
- For `<Variable Name="Temperature"`, the `DataType` list includes: char, signed char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long. The description for `int` is "emperature in F
- For `<Variable Name="Pressure"`, the `DataType` list includes: char, signed char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long. The description for `int` is "mospheric pres
- For `<Variable Name="WindSpeed"`, the `DataType` list includes: char, signed char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long. The description for `int` is "nd speed in mi
- For `<Variable Name="WindDirection"`, the `DataType` list includes: char, signed char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long. The description for `int` is "nd direction i
- For `<Variable Name="Rainfall"`, the `DataType` list includes: char, signed char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long. The description for `int` is "infall in incl
- For `<Variable Name="UVIndex"`, the `DataType` list includes: char, signed char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long. The description for `int` is "length of ultr

Figure 3-8 Creating an MQTT Configuration File in Visual Studio: Drop-Down Lists

Visual Studio makes editing your MQTT configuration file easier by filling required XML syntax for you. For example, after a schema is associated with your MQTT configuration, you get a drop-down list of expected elements any time you type `<`.

When you type `SPACE` from inside a start tag, you also get a drop-down list showing all attributes that can be added to the current element.

Likewise, when you type "=" for an attribute value, or the opening quote for the value, you also get a list of possible values for that attribute.

Moreover, *ToolTips* appear on these IntelliSense lists giving you a description of each element as illustrated in Figure 3-5 when the user hovers the mouse over the Topic's **WriteTo** property:

The screenshot shows a Visual Studio editor window titled "uC-Probe-MQTT-CfgFile-Template.mqtt". The code is XML-like and defines a topic with various variables. A tooltip is displayed over the "WriteTo" attribute value, containing the text: "Topic that contains the variables that have read/write access as provisioned in the MQTT server." The code is as follows:

```

<!-- Topic -->
<Topic WriteTo="com.micrium/WeatherMeterCfg/33076" ReadFrom="com.micrium/WeatherMeter/33076" >
  <!--
  Topic that contains the variables that have read/write access as provisioned in the MQTT server.
  -->
  <Payload>
    <!-- Read/Write variables -->
    <Variable Name="SamplingRate"      DataType="int"      Description="Samples per second." />
    <!-- Read-only variables -->
    <Variable Name="RelativeHumidity"  DataType="int"      Description="Air's relative humidity as a
    <Variable Name="Temperature"      DataType="int"      Description="Temperature in Farenheit degr
    <Variable Name="Pressure"          DataType="float"    Description="Atmospheric pressure in inch
    <Variable Name="WindSpeed"        DataType="int"      Description="Wind speed in miles per hour
    <Variable Name="WindDirection"    DataType="int"      Description="Wind direction in degrees." /
    <Variable Name="Rainfall"         DataType="int"      Description="Rainfall in inches (in)." />
    <Variable Name="UVIndex"          DataType="int"      Description="Strength of ultraviolet radi
  </Payload>
</Topic>

```

Figure 3-9 Creating an MQTT Configuration File in Visual Studio: Tool Tips

MQTT Configuration Files need to have the extension `.mqtt` for  $\mu$ C/Probe to recognize them as such. Listing 3-1 shows an example of an MQTT Configuration File that declares a few variables.

```

<?xml version="1.0" encoding="utf-8" ?>
<!--
XML-based Message Queue Telemetry Transport (MQTT) Configuration File Example.
-->
<Brokers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="http://micrium.com/probe/mqtt.xsd"           (1)
<!-- User 1 Account at 2lemetry.com -->
<Broker
  HostName="q.m2m.io"
  Port="1883"
  ClientID="any_unique_string_1"
  UseSSL="false"
  KeepAlive="30"
  Username="user_1@domain.com"
  Password="password_1"
  SendPasswordWithoutHashing="false"                                           (2)
  <!-- Topic -->
  <Topic WriteTo="com.micrium/WeatherMeterCfg/33076"
        ReadFrom="com.micrium/WeatherMeter/33076" >                             (3)
    <!-- JSON Format -->
    <Payload>                                                                     (4)
      <!-- Read/Write variables -->
      <Variable Name="SamplingRate"
        DataType="int"
        Description="Samples per second." />                                       (5)
      <!-- Read-only variables -->
      <Variable Name="RelativeHumidity"
        DataType="int"
        Description="Air's relative humidity as a percentage (%)." />
      <Variable Name="Temperature"
        DataType="int"
        Description="Temperature in Fahrenheit degrees (F)." />
    </Payload>
  </Topic>
</Broker>
</Brokers>

```

Listing 3-2 XML-based MQTT Configuration File Example

- L3-2(1) The root element is called **<Brokers>**. Inside this tag, you can have more than one MQTT broker in case your data comes from various sources. The tag includes the XSD schema reference to help you editing in an XML editor such as Visual Studio.
- L3-2(2) Each MQTT broker needs to be configured with the access credentials and other settings specific to the account you have with your MQTT broker.

- L3-2(3) MQTT brokers are usually provisioned with two topics per device (thing). In this example, the weather meter at zip code 33326 is provisioned with two topics; One that contains the Read/Write access variables and the other one that contains the Read-only access variables. And if you want added security you can even create a separate account for Read/Write access.
- L3-2(4) The data to be sent needs to be in JSON format. The tag <Payload> specifies the JSON format expected by both the MQTT broker and µC/Probe.
- L3-2(5) The tag <Variable> allows you to specify the name, data type and an optional description for each variable within a topic.

In order to verify your MQTT configuration file, you can use the Symbol Browser from within µC/Probe as shown in Figure 3-10. Notice the relationship between the XML tags and the tree nodes in the Symbol Browser.:

Name	Display Name	Type	Size	Size Filter	Memory Address
MQTT Example Template.mqtt	MQTT Example Template.mqtt		64	64	0x0
q.m2m.io	q.m2m.io		32	32	0x0
com.micrium/WeatherMeter/33076 - com.micrium/\	com.micrium/WeatherMeter/33076		32	32	0x0
SamplingRate	SamplingRate	int	4	4	0x0
RelativeHumidity	RelativeHumidity	int	4	4	0x0
Temperature	Temperature	int	4	4	0x0
Pressure	Pressure	float	4	4	0x0
WindSpeed	WindSpeed	int	4	4	0x0
WindDirection	WindDirection	int	4	4	0x0
Rainfall	Rainfall	int	4	4	0x0
UVIndex	UVIndex	int	4	4	0x0
q.m2m.io	q.m2m.io		32	32	0x0

Figure 3-10 XML-based MQTT Configuration File Example as seen from µC/Probe's Symbol Browser

## µC/Probe Settings

The µC/Probe application's tool bar is located at the top of the application window.

The µC/Probe Settings window is opened by making click on the **Settings** button in the application's tool bar as indicated in Figure 4-1:

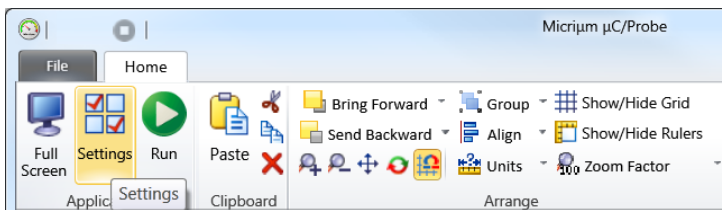


Figure 4-1 µC/Probe Toolbar: Settings

The settings window is divided in the following categories:

- General Settings:

The general settings include the application debug logging, automatic updates and data collection settings.

- Communication Settings:

The communication settings window includes the endianness type and the communication interface settings.

## 4-1 GENERAL SETTINGS

Figure 4-2 shows the  $\mu$ C/Probe General Settings window:

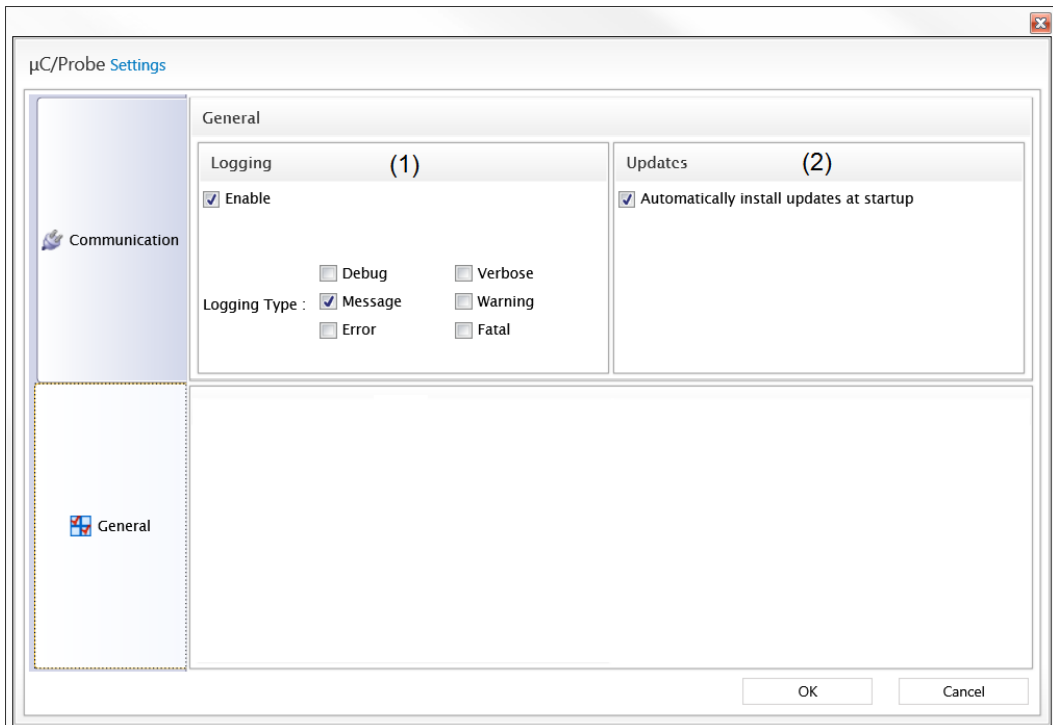


Figure 4-2  $\mu$ C/Probe General Settings

F4-2(1)  $\mu$ C/Probe can be configured to log a specific level of verbosity for technical support purposes. If you ever have to contact Micrium's technical support for any issues with your  $\mu$ C/Probe application, you can select a logging type that better describes your failure scenario.

F4-2(2)  $\mu$ C/Probe can be configured to automatically check and install updates as they become available from the Micrium website.  $\mu$ C/Probe will check for software updates at startup if internet access is available.

## **4-2 COMMUNICATION SETTINGS OVERVIEW**

µC/Probe supports a variety of communication interfaces that can be classified in four groups:

- Debugger-based Interfaces
- Peripheral-based Interfaces
- 3rd. Party Plugins
- MQTT Interface

The following sections will describe each of the groups.

### **4-2-1 DEBUGGER-BASED INTERFACES**

µC/Probe can interface with your embedded target via standard debugging tools such as:

- J-Link
- CMSIS-DAP
- Cypress PSoC Prog
- OpenSDA

When using one of these interfaces, µC/Probe reads and writes your global variables in a non-intrusive way and without the need for any target resident code. These are the ideal interface options if you require to maintain the real-time behavior of your application.

Follow the decisions tree diagram in Figure 4-3 to see if you can use one of these debugger-based interfaces with your platform.

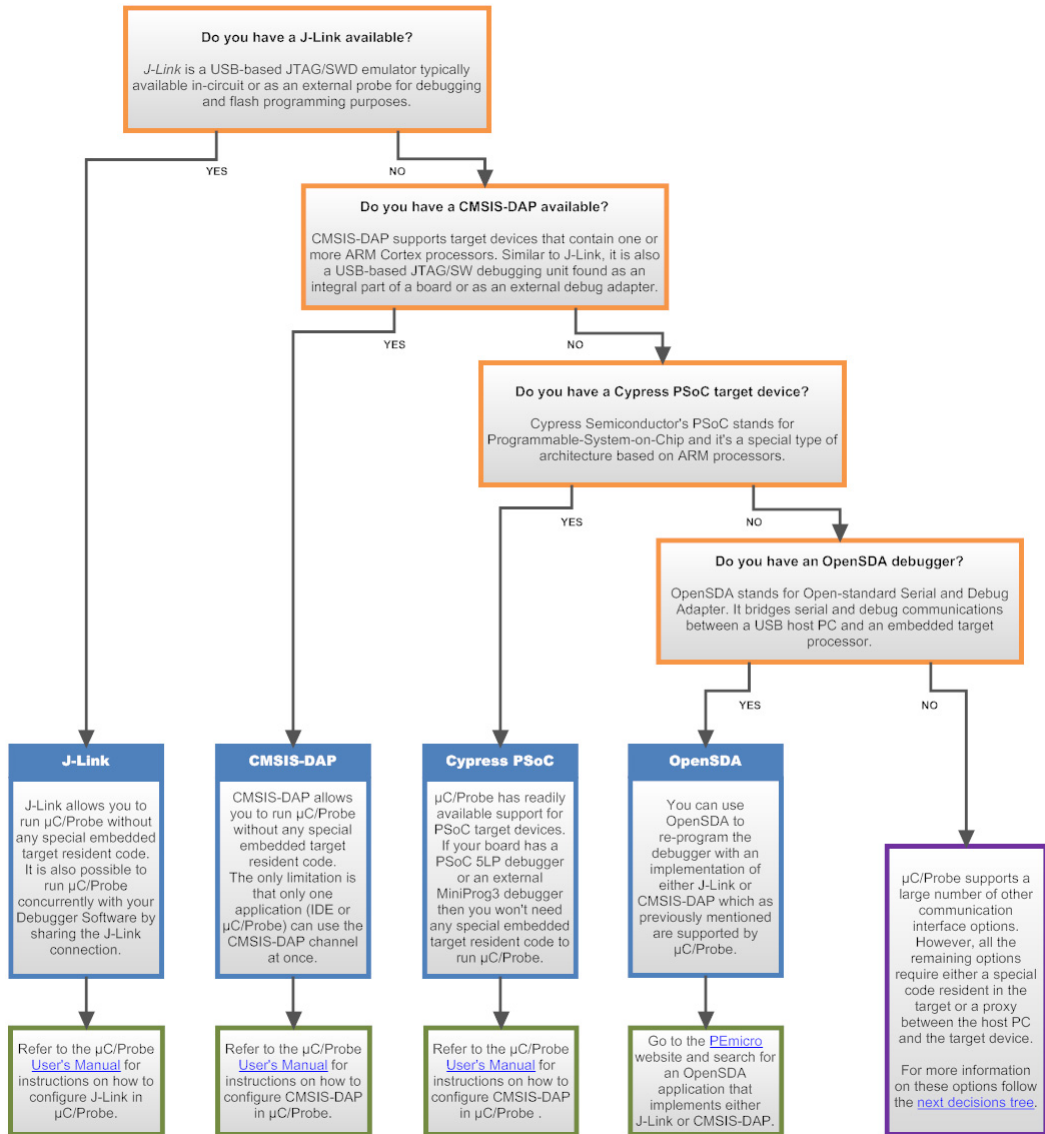


Figure 4-3 Debugger-based Interfaces



## **4-2-2 PERIPHERAL-BASED INTERFACES**

$\mu$ C/Probe can interface via the following communication modules:

- USB
- TCP/IP
- RS-232

These interfaces however, require a special target resident code that implements the  $\mu$ C/Probe protocol.

Micrium has RS-232 drivers for various platforms available for free and if you are a  $\mu$ C/USB or  $\mu$ C/TCP-IP licensee then  $\mu$ C/Probe is ready to run on these stacks.

Follow the decisions tree diagram in Figure 4-4 to see if you can use one of these interfaces with your platform.

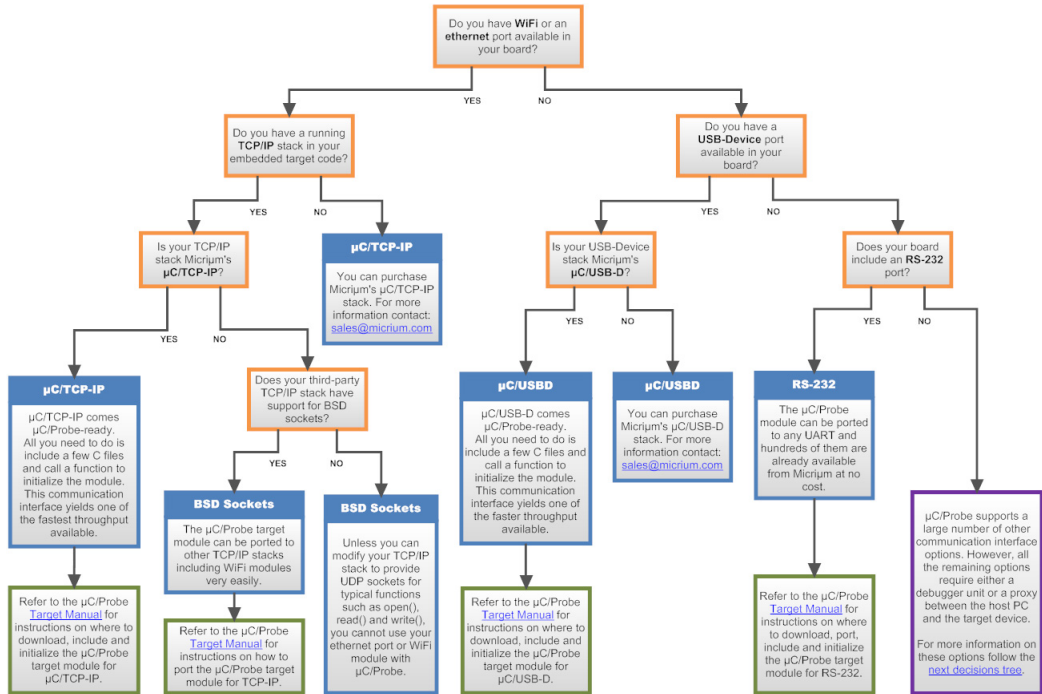


Figure 4-4 Peripheral-based Interfaces

### 4-2-3 THIRD PARTY PLUGINS

µC/Probe can interface with the embedded target via your IDE thanks to a public API that enables third party software applications to establish a TCP/IP bridge between the IDE and µC/Probe.

The TCP/IP bridge is created in the form of a plugin and gives µC/Probe access to as many platforms as the IDE supports.

Examples of these plugins are:

- IAR Systems C-SPY plugin for µC/Probe
- Eclipse plugin for µC/Probe

Follow the decisions tree diagram in Figure 4-5 to see if you can use one of these plugins with your platform.

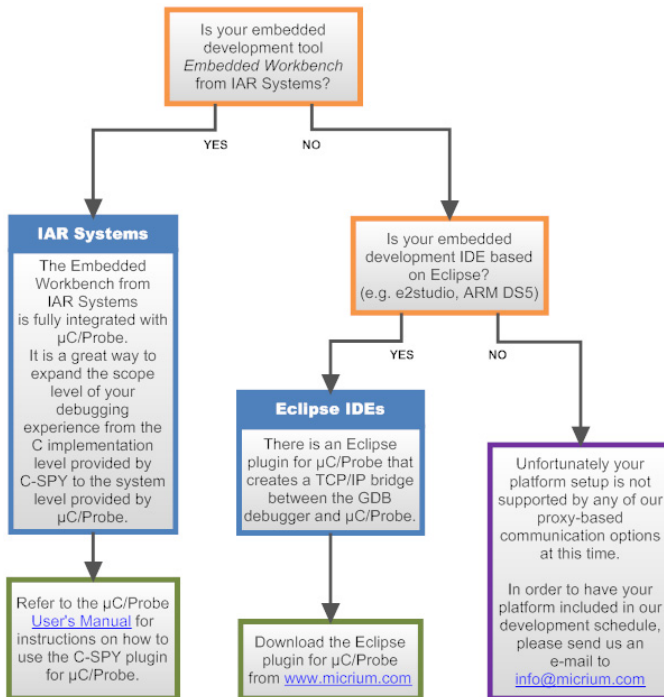


Figure 4-5 Third Party Plugins

#### **4-2-4 MQTT INTERFACE**

$\mu$ C/Probe can also interface remotely to your embedded target via MQTT.

MQTT is becoming the de facto protocol for IoT applications and  $\mu$ C/Probe can be configured to be an MQTT client for those embedded systems that do not have neither TCP/IP connectivity nor intelligence that implements the MQTT protocol.

### 4-3 COMMUNICATION SETTINGS WINDOW

Once you have chosen the appropriate communication interface for your platform, you can configure  $\mu$ C/Probe from the communication settings window shown in Figure 4-6:

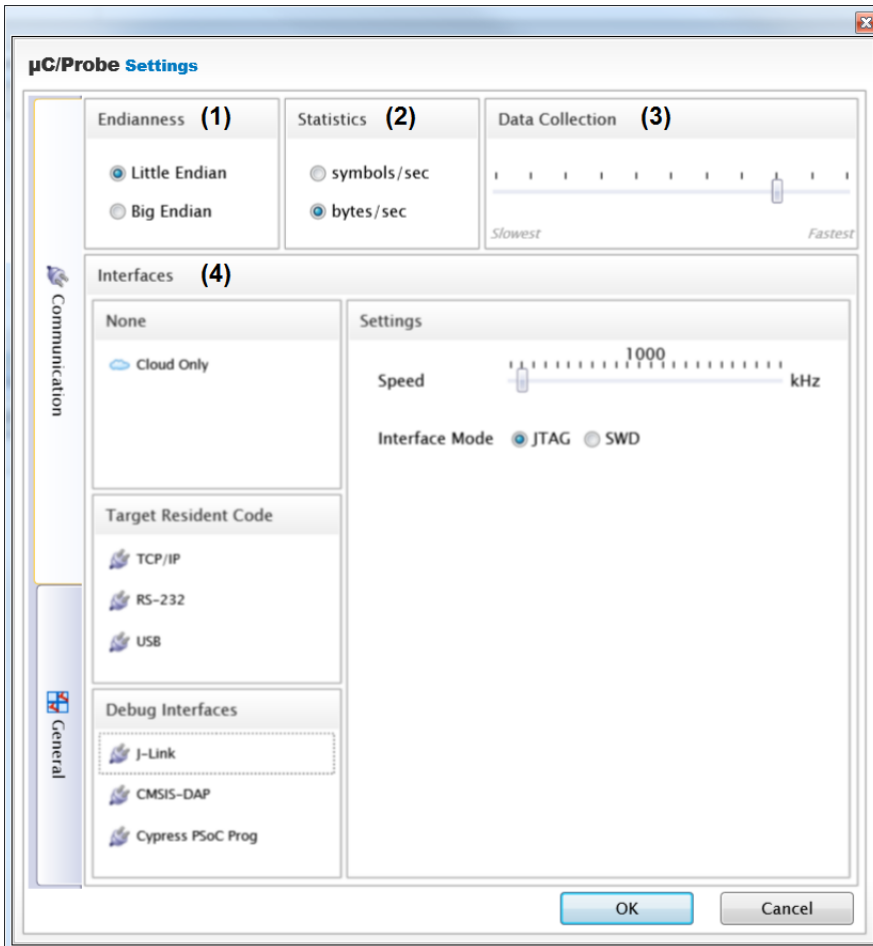


Figure 4-6  $\mu$ C/Probe Communication Settings

- F4-6(1)  $\mu$ C/Probe can be configured to interpret the byte order in either little or big endian, select the option that corresponds to your target's endianness type.
- F4-6(2) During run-time,  $\mu$ C/Probe calculates the speed of your connection and displays the value in either symbols per second or bytes per second.

- F4-6(3) You can adjust how fast you want  $\mu$ C/Probe to make requests to the target depending on your application's bandwidth. The fastest collection rate takes a toll in both the embedded target and your Windows PC resources.
- F4-6(4)  $\mu$ C/Probe supports MQTT, J-Link, CMSIS-DAP, Cypress PSoC Prog, USB, TCP/IP and RS-232. Select the interface that your target supports, and configure the settings corresponding to the interface.

The following sections describe how to configure each communication interface. Refer to the document [μC/Probe-Target Manual](#) for more information on the communication interface supported by the embedded target.

### 4-3-1 SEGGER J-LINK

J-Link is a USB powered JTAG emulator designed by Segger. In order to install the windows drivers for J-Link (J-Link DLL) go to Segger's website at [www.segger.com](http://www.segger.com) and download the J-Link software pack for Windows.

J-Link is the most popular emulator for ARM cores and it does not require any special code resident in the embedded target to connect with  $\mu$ C/Probe.

If using J-Link, you can interface  $\mu$ C/Probe even with a bare-metal application running no kernel at all, as shown in Figure 4-7:

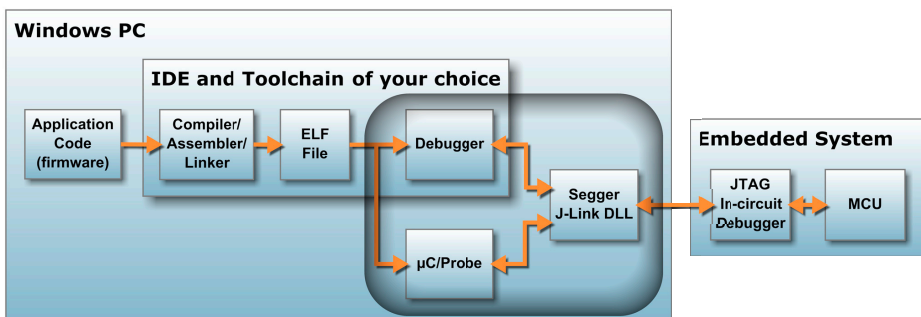


Figure 4-7  $\mu$ C/Probe via J-Link

$\mu$ C/Probe supports two types of J-link interface modes; JTAG and SWD. Select the interface mode from the radio buttons and configure the J-Link speed from the horizontal slider shown in Figure 4-8.  $\mu$ C/Probe will negotiate the speed you configured and if your device does not support it, then it will select the maximum possible:

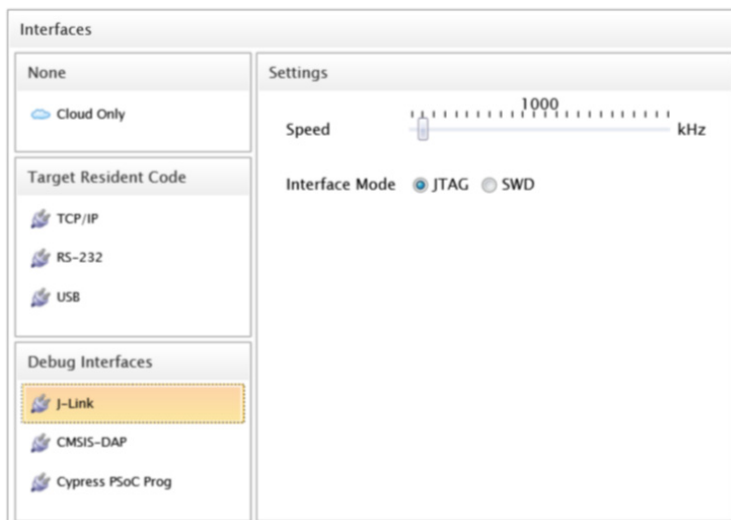


Figure 4-8  $\mu$ C/Probe Communication Settings: J-Link

### 4-3-2 CMSIS-DAP

CMSIS-DAP is the interface firmware for an ARM Cortex processor's Debug Unit that connects the Debug Port to USB.  $\mu$ C/Probe which executes on a host computer, connects via USB to the Debug Unit and to the device that runs the application software. The Debug Unit connects via JTAG or SW to the target device.

The CMSIS-DAP interface does not require you to install any drivers, simply connect a USB cable between your board debug port and your Windows PC and then select the CMSIS-DAP interface from the Settings window as shown in Figure 4-9:

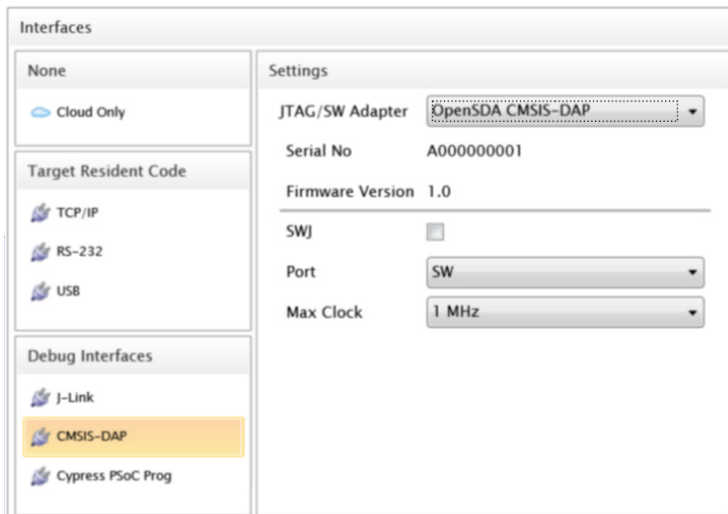


Figure 4-9  $\mu$ C/Probe Communication Settings: CMSIS-DAP

Keep in mind that the CMSIS-DAP interface supports the communication of one single client at a time. In other words, you cannot run  $\mu$ C/Probe and your debugger software at the same time.



### 4-3-3 CYPRESS PSoC PROG

Cypress have their own debugging interface for their PSoC devices called PSoC Programmer. The interface allows you to not only program and configure the PSoC device but also to debug it.  $\mu$ C/Probe is tightly integrated with Cypress PSoC 5LP devices through this interface.

To use this interface with your Cypress PSoC 5LP device you need to download and install PSoC Programmer from the Cypress website at: <http://www.cypress.com>

Then you simply connect the board and select the PSoC Prog interface from the Settings menu as shown in Figure 4-10:

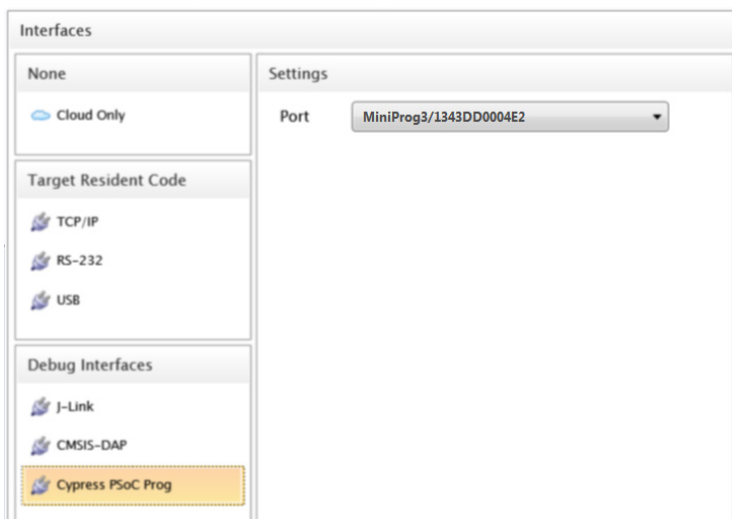


Figure 4-10  $\mu$ C/Probe Communication Settings: Cypress PSoC Prog

Keep in mind that the Cypress PSoC Prog interface supports the communication of one single client at a time. In other words, you cannot run  $\mu$ C/Probe and your debugger software at the same time.

### 4-3-4 USB

$\mu$ C/Probe supports a USB interface over the  $\mu$ C/USB Device stack by Micrium. This USB interface requires  $\mu$ C/Probe-Target code resident in your embedded system and because of the nature of USB, it also requires a kernel. Micrium supports many cores and most likely this code is available from Micrium. Contact Micrium to find out if resident code for your particular setup is available.

Once your embedded system is running  $\mu$ C/Probe-Target as described in the document [μC/Probe-Target Manual](#), the device should be ready to connect after plugging in. The Windows computer will enumerate the device and will display it as one of the available devices in the communication settings window as shown in Figure 4-11:

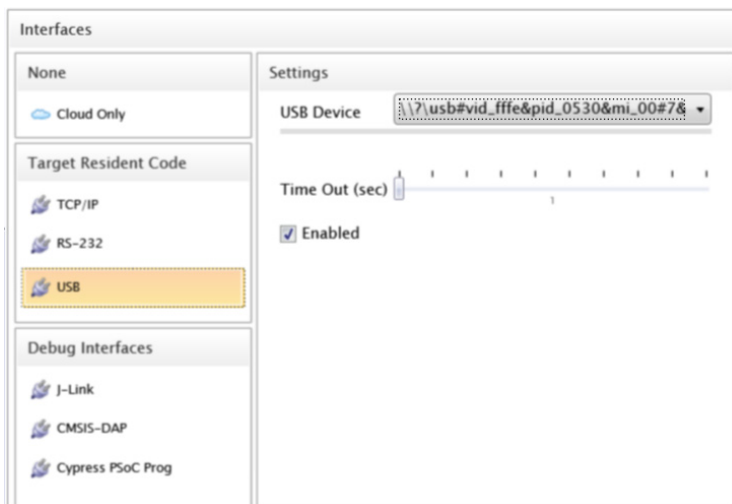


Figure 4-11  $\mu$ C/Probe Communication Settings: USB

You can specify an optional timeout in seconds, which is the time  $\mu$ C/Probe is willing to wait for the target to respond before presenting an error message.

### 4-3-5 TCP/IP

$\mu$ C/Probe supports a TCP/IP interface over the UDP protocol. The target requires a TCP/IP stack that provides a BSD sockets interface. Regardless of the TCP/IP stack being used, this interface requires  $\mu$ C/Probe-Target code resident in your embedded system and because of the nature of TCP/IP, it also requires a kernel. Micrium supports many cores and most likely this code is available from Micrium. Contact Micrium to find out if resident code for your particular setup is available.

Assuming your embedded system is running  $\mu$ C/Probe-Target as described in the document [μC/Probe-Target Manual](#), enter the IP address and port number of your embedded system in the text boxes shown in Figure 4-12:

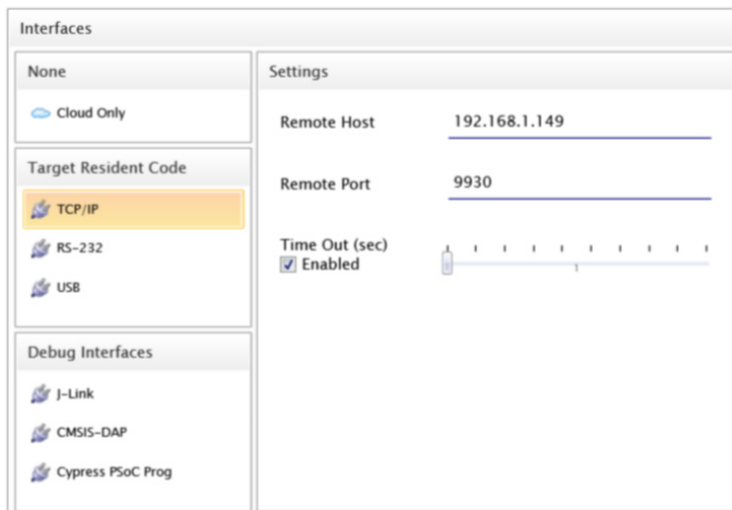


Figure 4-12  $\mu$ C/Probe Communication Settings: TCP/IP

You can specify an optional timeout in seconds, which is the time  $\mu$ C/Probe is willing to wait for the target to respond before presenting an error message.

The TCP/IP interface is also used to interface through third-party plugin proxies such as the IAR Systems C-SPY plugin for  $\mu$ C/Probe as described in section 9-3 “IAR Systems C-SPY Plugin for  $\mu$ C/Probe” on page 64.

### 4-3-6 RS-232

$\mu$ C/Probe supports a Serial RS-232 interface. This serial interface requires  $\mu$ C/Probe-Target code resident in your embedded system. Micrium supports many UARTs and most likely this code is available from Micrium. Contact Micrium to find out if resident code for your particular setup is available.

Assuming your embedded system is running  $\mu$ C/Probe-Target as described in the document [μC/Probe-Target Manual](#), enter the serial COM port number that your embedded target is attached to and select the baud rate from the drop downs shown in Figure 4-13:

The screenshot shows the 'Interfaces' settings window. It is divided into three main sections: 'None', 'Target Resident Code', and 'Debug Interfaces'. The 'None' section has a 'Cloud Only' option. The 'Target Resident Code' section has three options: 'TCP/IP', 'RS-232' (which is highlighted with a yellow dashed border), and 'USB'. The 'Debug Interfaces' section has three options: 'J-Link', 'CMSIS-DAP', and 'Cypress PSoC Prog'. To the right of these sections is a 'Settings' panel with two dropdown menus: 'COM Port' set to 'COM1' and 'Baud Rate' set to '38400'.

Figure 4-13  $\mu$ C/Probe Communication Settings: RS-232

## μC/Probe Workspace Explorer

The μC/Probe Workspace Explorer is located on the right side of the application window and it is shown in Figure 5-1:

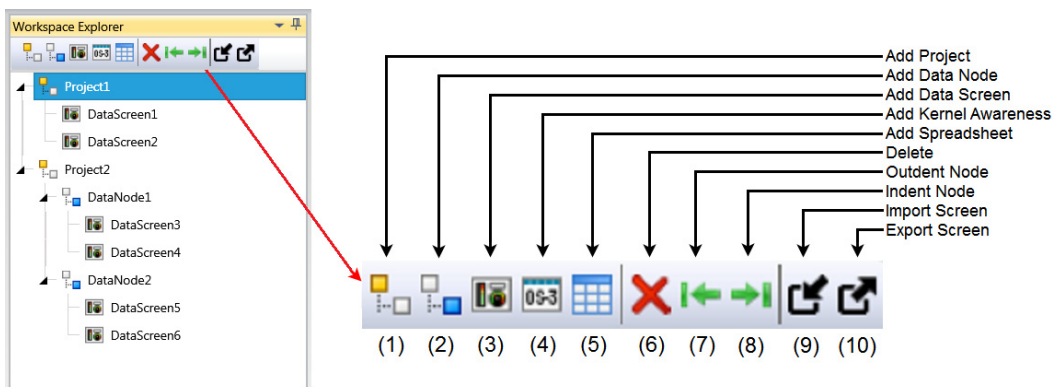


Figure 5-1 μC/Probe Workspace Explorer

- F5-1(1) μC/Probe allows you to create a dashboard or user interface in a matter of minutes. The data screen is where you drag and drop the virtual controls and indicators. Your data screen's appearance and layout are very important. You can use Projects as a means to separate complex dashboards into multiple regions. Each project can contain multiple data screens and the workspace explorer allows you to navigate through this hierarchy.
- F5-1(2) Similar to Projects, Data Nodes are just another level of hierarchy that allows you to group sets of virtual controls and indicators together into categories you define.
- F5-1(3) Data Screens are the screens where you drag and drop the virtual controls and indicators. You can add as many data screens as you want.

- 
- F5-1(4) The Kernel Awareness Screen is a pre-configured Data Screen with all the symbols related to  $\mu$ C/OS-III. See Appendix B, “Kernel Awareness Screen” on page 96 for more information about this.
- F5-1(5) Use this button to create a bridge between  $\mu$ C/Probe and Microsoft Excel.
- F5-1(6) Use the Delete button to delete an item from the workspace explorer, including Projects, Data Nodes, Data Screens and Kernel Awareness Screens.
- F5-1(7) Use the indent button to push in an item in the Workspace Explorer tree. The items you can adjust the level of indentation include Projects, Data Nodes, Data Screens and Kernel Awareness Screens.
- F5-1(8) Use the outdent button to push out an item in the Workspace Explorer tree. The items you can outdent include Projects, Data Nodes, Data Screens and Kernel Awareness Screens.
- F5-1(9) Use the Import Screen button to import a previously exported screen.
- F5-1(10) Use the Export Screen button to export the screen currently in focus to a file.

In order to organize your workspace tree you can also use your mouse to drag and drop items and rename items by invoking the context menu with a right-click.

Figure 5-2 shows an example of using projects and data nodes to better present a control panel for a liquid level control system:

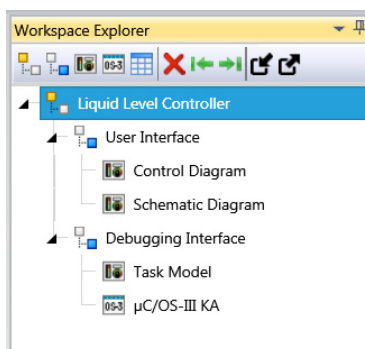


Figure 5-2 **Workspace Explorer for a Liquid Level Control System**

$\mu$ C/Probe Toolbox

The  $\mu$ C/Probe Toolbox is located on the left side of the application window and it is shown in Figure 6-1:



Figure 6-1  $\mu$ C/Probe Toolbox

Depending on the  $\mu$ C/Probe Edition you purchased, the available tools will vary. This document describes all the features found in the Professional Edition of  $\mu$ C/Probe. For more information on which features you have, see Appendix I, “ $\mu$ C/Probe Editions Comparison Table” on page 125.

The items in the toolbox are contained in an accordion type of panel. You click on each button to display the items that belong to a category.

If you are running the Basic or Professional Edition of  $\mu$ C/Probe and have the automatic updates enabled, each category in this toolbox will expand with more virtual controls and indicators as software updates become available.

The following sections present a brief introduction to each of the toolbox categories. For more information on configuring each type of virtual control or indicator, see Appendix A, “Configuring Virtual Controls and Indicators” on page 67.

## 6-1 WRITABLE CONTROLS

The writable controls shown in Figure 6-2 include buttons, check boxes, sliders, a bit control, an RGB LED color picker, a numeric up/down and a textbox. Use these controls to read and modify the value of symbols from the embedded target. For more information configuring the properties of writable controls see Appendix A, “Virtual Controls” on page 74.

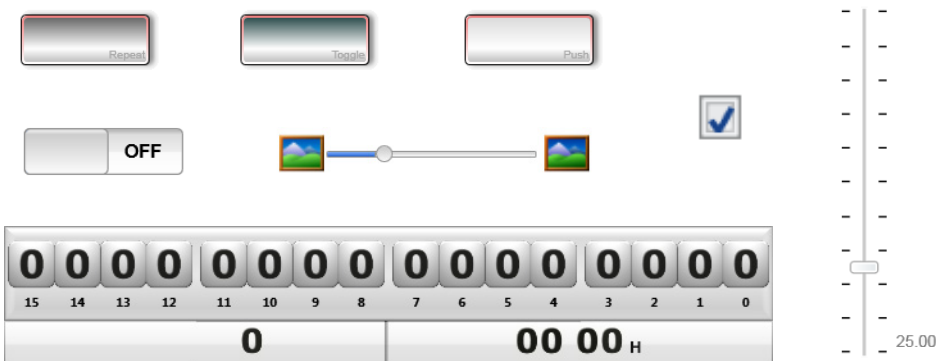


Figure 6-2  $\mu$ C/Probe Toolbox: Writable Controls



## 6-2 LINEAR GAUGES

Use the linear gauges shown in Figure 6-3 to display numeric data in a tri-color vertical scale. For more information configuring the properties of linear gauges see Appendix A, “Virtual Indicators” on page 68.

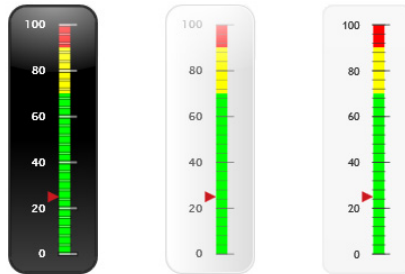


Figure 6-3  $\mu$ C/Probe Toolbox: Linear Gauges

## 6-3 HORIZONTAL LINEAR GAUGES

Use the horizontal linear gauges shown in Figure 6-4 to display numeric data in a tri-color horizontal scale. For more information configuring the properties of horizontal linear gauges see Appendix A, “Virtual Indicators” on page 68.

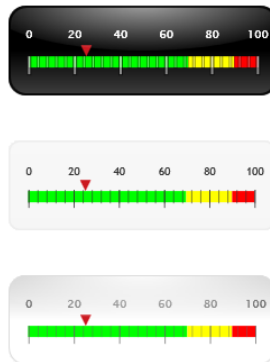


Figure 6-4  $\mu$ C/Probe Toolbox: Horizontal Linear Gauges

## 6-4 QUADRANT GAUGES

Use the quadrant gauges shown in Figure 6-5 to display numeric data in a tri-color quadrant scale. For more information configuring the properties of quadrant gauges see Appendix A, “Virtual Indicators” on page 68.

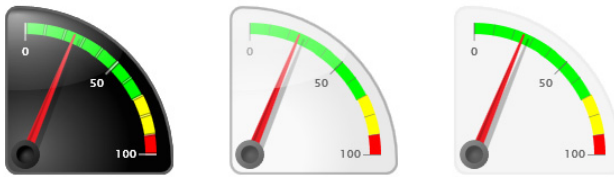


Figure 6-5  $\mu\text{C}/\text{Probe}$  Toolbox: Quadrant Gauges

## 6-5 SEMICIRCLE GAUGES

Use the semicircle gauges shown in Figure 6-6 to display numeric data in a tri-color semicircular scale. For more information configuring the properties of semicircle gauges see Appendix A, “Virtual Indicators” on page 68.

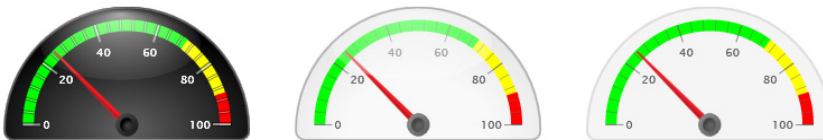


Figure 6-6  $\mu\text{C}/\text{Probe}$  Toolbox: Semicircle Gauges

### 6-6 CIRCULAR GAUGES

Use the circular gauges shown in Figure 6-7 to display numeric data in a tri-color circular scale. For more information configuring the properties of circular gauges see Appendix A, “Virtual Indicators” on page 68.



Figure 6-7  $\mu$ C/Probe Toolbox: Circular Gauges

### 6-7 HALF DONUTS

Use the half donut indicators shown in Figure 6-8 to display numeric data in a bi-color semicircular scale. For more information configuring the properties of half donuts see Appendix A, “Virtual Indicators” on page 68.

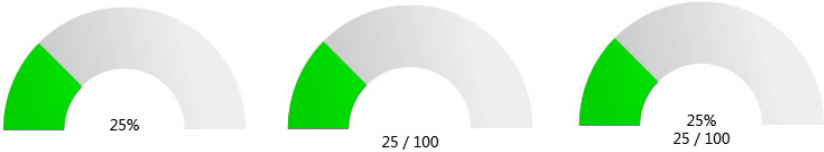


Figure 6-8  $\mu$ C/Probe Toolbox: Half Donuts

## 6-8 CYLINDERS

Use the cylinder or tank level indicator shown in Figure 6-9 to display numeric data in a solid or gradient color. For more information configuring the properties of cylinders see Appendix A, “Virtual Indicators” on page 68.



Figure 6-9  $\mu$ C/Probe Toolbox: Cylinders

## 6-9 CHARTS

Use the charts shown in Figure 6-10 to display numeric data including arrays in a marker, line, area or scatter x-y chart. For more information configuring the properties of charts show Appendix A, “Timeline Charts” on page 87.



Figure 6-10  $\mu$ C/Probe Toolbox: Charts

## 6-10 NUMERIC INDICATORS

Use the numeric indicators shown in Figure 6-11 to display numeric data in text. For more information configuring the properties of numeric indicators see Appendix A, “Formatting Properties Editor” on page 68 and Appendix A, “Numeric Indicator Properties Editor” on page 70.



Figure 6-11  $\mu$ C/Probe Toolbox: Numeric Indicators

## 6-11 LEDS

Use the LEDs shown in Figure 6-12 to display not only boolean data types but also any numeric variable that represents a color code in ARGB format. For more information configuring the properties of LEDs see Appendix A, “Formatting Properties Editor” on page 68 and Appendix A, “LED Properties Editor” on page 71.



Figure 6-12  $\mu$ C/Probe Toolbox: LEDS

## 6-12 ADVANCED

The advanced category of the toolbox includes other miscellaneous indicators such as a text box, terminal window, scripting control, spreadsheet control,  $\mu$ C/Trace trigger control, data logger, HID control and an image container capable of displaying an indexed array of images. For more information configuring the properties of these advanced controls see Appendix A, “Virtual Indicators” on page 68, Appendix E, “Spreadsheet Control” on page 106 and Appendix F, “Scripting Control” on page 111.

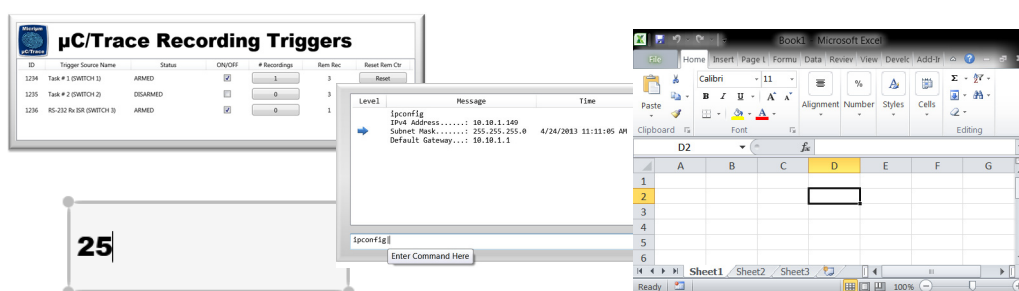


Figure 6-13  $\mu$ C/Probe Toolbox: Advanced

## μC/Probe Layout Design Tools

The Layout Design Tools are located on the Main Toolbar at the top of the application's window. They include tools to arrange the virtual controls and indicators on your data screen as shown in Figure 7-1:

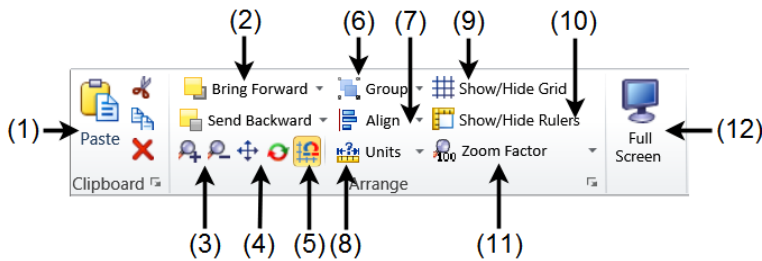


Figure 7-1 μC/Probe Layout Design Tools

- F7-1(1) μC/Probe allows you to copy, cut and paste a single or multiple virtual controls or indicators. Everything you select will be stored temporarily into μC/Probe's clipboard during your session.
- F7-1(2) μC/Probe allows you to drag and drop virtual controls and indicators onto the data screen and have them overlap one another. In some cases may be necessary to move some of them to the front of the data screen. Select the virtual control and indicator you want to move and click the **Bring Forward** or **Send Backward** button to move it to the front or to the back respectively.
- F7-1(3) μC/Probe includes accessibility features that make the software more user friendly. You can control the zoom level of your μC/Probe Data Screen during both design-time and run-time. Click the buttons with the magnifier glass to zoom in and out, or use the keyboard shortcuts **Ctrl+** to zoom-in and **Ctrl-** to zoom-out.

---

F7-1(4) The panning tool in  $\mu$ C/Probe makes it easy to move a data screen around while you are zoomed in. The **Pan** button is a toggle button, clicking the button toggles the panning mode on and off.

F7-1(5) When you drag and drop a virtual control or indicator onto the data screen, the next thing you usually do is resize or move the object around the data screen. Turn the Snap-to-Grid mode on in order to align the virtual control or indicator to the nearest intersection of grid lines. The **Snap to Grid** button is a toggle button, clicking the button toggles the snap-to-grid mode on and off.

F7-1(6) You can combine multiple virtual controls and indicators so you can work with them as though they were a single object. You can resize, move, copy and paste all virtual controls and indicators in a group as a single unit.

After you have grouped virtual controls and indicators, you can still select any single object within the group without ungrouping by first selecting the group, and then clicking on the object you want to select.

F7-1(7)  $\mu$ C/Probe allows you to easily align virtual controls and indicators by first selecting the group of objects you want to align and then clicking on one of the following alignment options:

- Left or Right Edges
- Top or Bottom Edges
- Horizontal or Vertical Centers

All the objects are aligned with respect to the first selected item.

F7-1(8) Use the **Units** button to select the grid and ruler's metric system.

F7-1(9) Use the **Show/Hide Grid** button to show and hide the grid lines on the data screen. The snap-to-grid mode still works even if the grid is not visible.

F7-1(10) Use the **Show/Hide Rulers** button to show or hide the ruler. The Show/Hide Rulers button is a toggle button, clicking the button turns the rulers on and off.



F7-1(11) Every time you click the magnifier glass buttons to zoom in and out, μC/Probe zooms in and out by certain zooming factor. Click the **Zoom Factor** button to select a different zooming factor.

F7-1(12) Click the **Full Screen** mode button to hide all the tools except the data screen. The Full Screen button is a toggle button, clicking the button turns the full screen mode on and off.

## 7-1 μC/PROBE EXAMPLE

In order to demonstrate the previous layout design tools, Figure 7-2 shows an example of a power plant's diagram used as a background to create a control panel with μC/Probe:

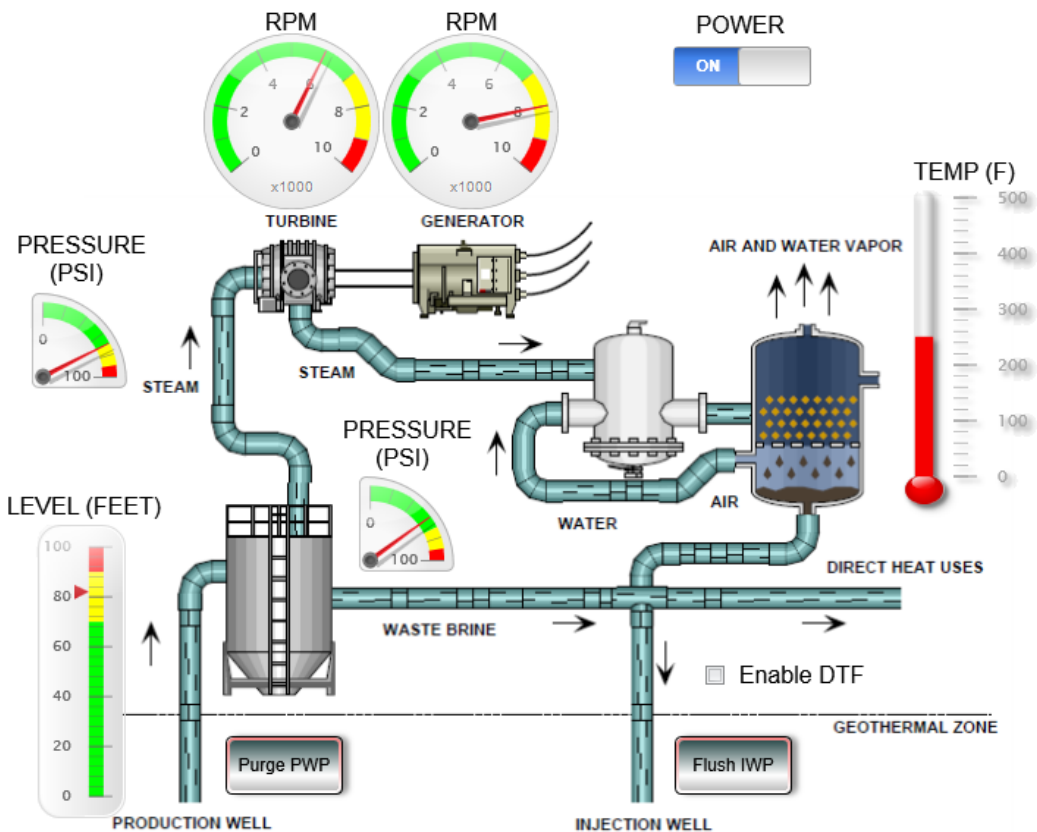


Figure 7-2 μC/Probe Example of a Power Plant

# Chapter 8

## Associating Symbols to Virtual Controls and Indicators

During design-time, use the  $\mu$ C/Probe symbol browser discussed in Chapter 3, “ $\mu$ C/Probe Symbol Browser” on page 15, to search and select the embedded target variables you want to associate to each of the virtual controls and indicators you placed on your data screen.

Once you find the symbol you want to associate, drag and drop the symbol over the virtual control or indicator you want, as shown in Figure 8-1:

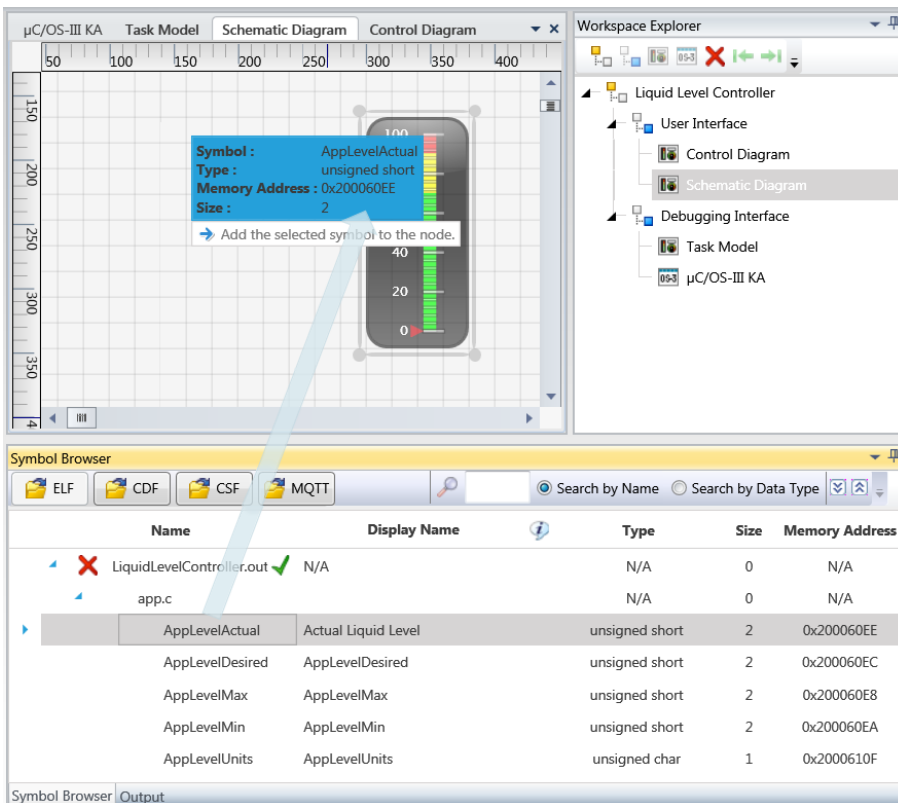


Figure 8-1 Associating Symbols to Virtual Controls and Indicators

Repeat the same process for each of the virtual controls and indicators placed on your data screen and  $\mu$ C/Probe will be ready to go into Run Mode unless you want to further configure other optional settings.

In order to configure other optional settings you can use the *symbols manager* by hovering your mouse pointer over the virtual control or indicator and making click on the icon shown in Figure 8-2:

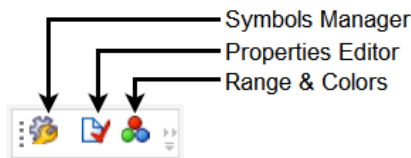


Figure 8-2 Invoking the Symbols Manager

Figure 8-3 shows the Symbols Manager:

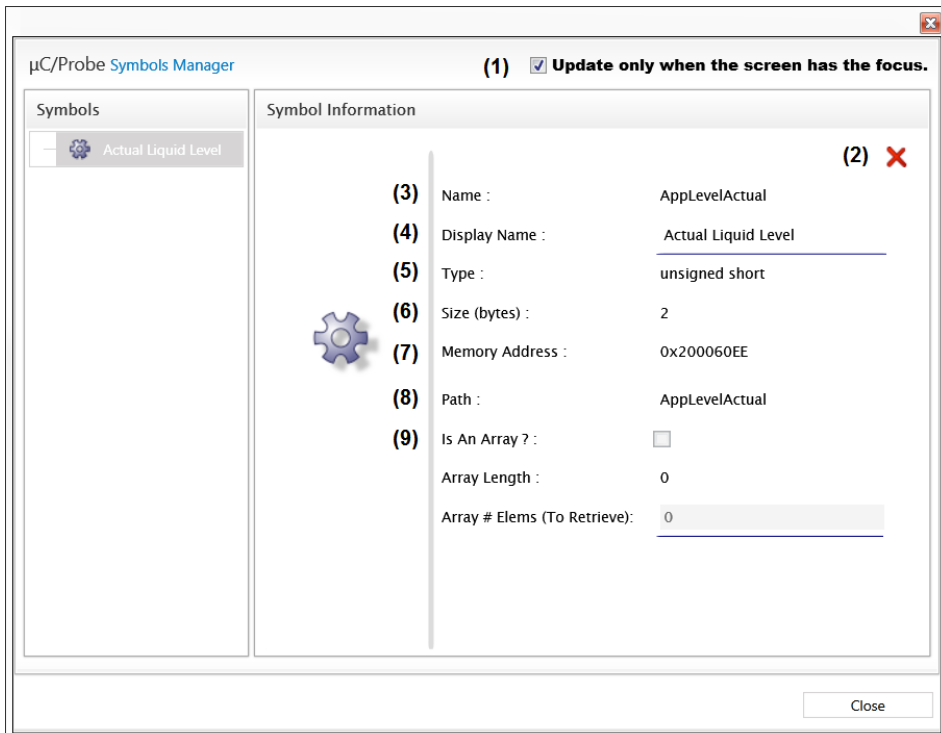


Figure 8-3 Symbols Manager

---

F8-3(1) Tick the checkbox to have  $\mu$ C/Probe update the control's value only when it is visible and in focus.

Untick this checkbox for controls such as charts which you might want to keep updated even when not visible to avoid any gaps in the plot.

F8-3(2) Click the red X to remove the symbol's association from the virtual control or indicator.

F8-3(3) Displays the name of the variable as declared in your C file.

F8-3(4) By default displays the name of the variable as declared in your C file, but this text box allows you to create an alias for display purposes.

F8-3(5) Displays the data type of the variable as declared in your C file.

F8-3(6) Displays the size of the variable in number of bytes.

F8-3(7) Displays the memory location of the variable in the embedded target's memory.

F8-3(8) The variable path displays the full variable name in those cases where the variable you selected is a member of a data structure.

F8-3(9) In case the variable is an array, you can enable indexing of just a certain amount of data.

## 9-1 RUN-TIME CHECKLIST

Before setting  $\mu$ C/Probe in Run-Time mode you should verify each of the following items:

Item #	Description	Reference
1	The embedded target has been programmed with an output file (ELF file) with debug information in the DWARF-2, -3 or -4 format or with an XML-based Custom-Symbol File or Chip-Definition File.	$\mu$ C/Probe Target Manual: Chapter 5, on page 21, Appendix C, on page 55 and Appendix D, on page 61
2	The embedded target is running and connected to the Windows PC through the communication interface of your choice.	$\mu$ C/Probe Target Manual: Chapter 3, on page 14
3	$\mu$ C/Probe has been configured with the latest symbol file (ELF, CDF, CSF or MQTT) that the embedded target is actually running.	$\mu$ C/Probe User's Manual: Chapter 3, on page 15
4	$\mu$ C/Probe has been configured with the proper communication interface and settings.	$\mu$ C/Probe User's Manual: Chapter 4, on page 29
5	$\mu$ C/Probe contains at least one virtual control or indicator on the data screen.	$\mu$ C/Probe User's Manual: Chapter 7, on page 55
6	$\mu$ C/Probe has been configured to associate the virtual control or indicator with one of the embedded target's variables displayed in the symbol browser.	$\mu$ C/Probe User's Manual: Chapter 8, on page 58

Table 9-1 Run-Time Mode Checklist

In order to set  $\mu$ C/Probe in Run-Time mode, click on the **run** button indicated in Figure 9-1:

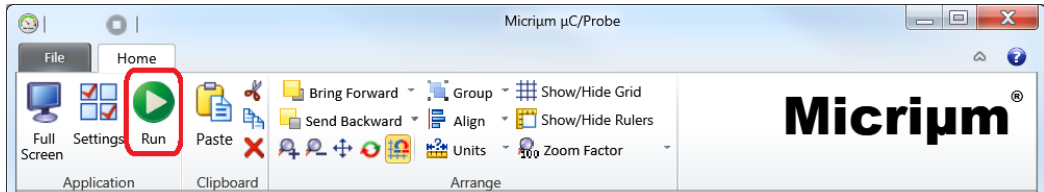


Figure 9-1 Setting  $\mu$ C/Probe in Run-Time mode

$\mu$ C/Probe should start updating your data screens immediately and the application displays all kinds of status information in the status bar at the bottom of the  $\mu$ C/Probe window as shown in Figure 9-2:

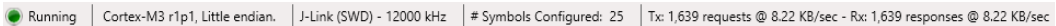


Figure 9-2  $\mu$ C/Probe Status Bar

## 9-2 RUNNING $\mu$ C/PROBE AND YOUR DEBUGGING SOFTWARE AT THE SAME TIME

Your debugging software for embedded applications usually comes integrated with your IDE and at a minimum, allows you to step through the code, set breakpoints, display register and memory windows, display call stack information, and monitor variables and expressions. Examples of debugging software include IAR's C-SPY and GNU's GDB.

You can also use  $\mu$ C/Probe to extend the capabilities of your debugging software by running both at the same time.  $\mu$ C/Probe allows you to have instant control over your global variables in a real-time and non-intrusive way. From your debugger software, you can set breakpoints at locations of particular interest in the application being debugged and  $\mu$ C/Probe will stop updating the virtual controls and indicators at the same time.

This feature is accomplished by sharing the connection between the Windows PC and the Embedded Target being debugged. Whether the debugger of your choice is IAR's C-SPY, GNU's GDB or any other debugging software that supports J-Link, Figure 9-3 illustrates an example of running  $\mu$ C/Probe and the debugger of your choice at the same time:

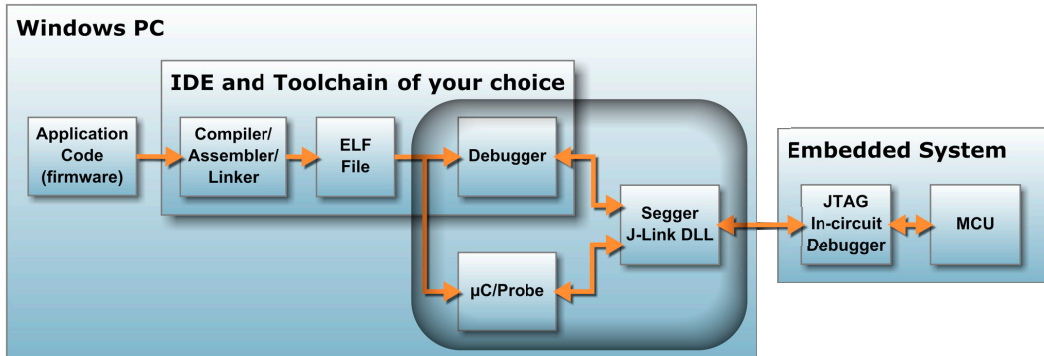


Figure 9-3 **Running  $\mu$ C/Probe and your debugging software at the same time**

Notice how  $\mu$ C/Probe and the Debugger Software not only share the same ELF file but also the same logic and physical interface through the Segger's J-Link DLL and JTAG in-circuit debugger respectively.

### 9-3 IAR SYSTEMS C-SPY PLUGIN FOR $\mu$ C/PROBE

$\mu$ C/Probe is tightly integrated with IAR Embedded Workbench<sup>®</sup> thanks to a TCP/IP bridge between C-SPY<sup>®</sup> and  $\mu$ C/Probe. This bridge gives  $\mu$ C/Probe access to not only its native supported platforms but also all the devices and processor architectures supported by IAR Systems without the need to write any target resident code in the form of communication routines, because C-SPY<sup>®</sup> handles all communication needed as illustrated in the following Figure 9-4:

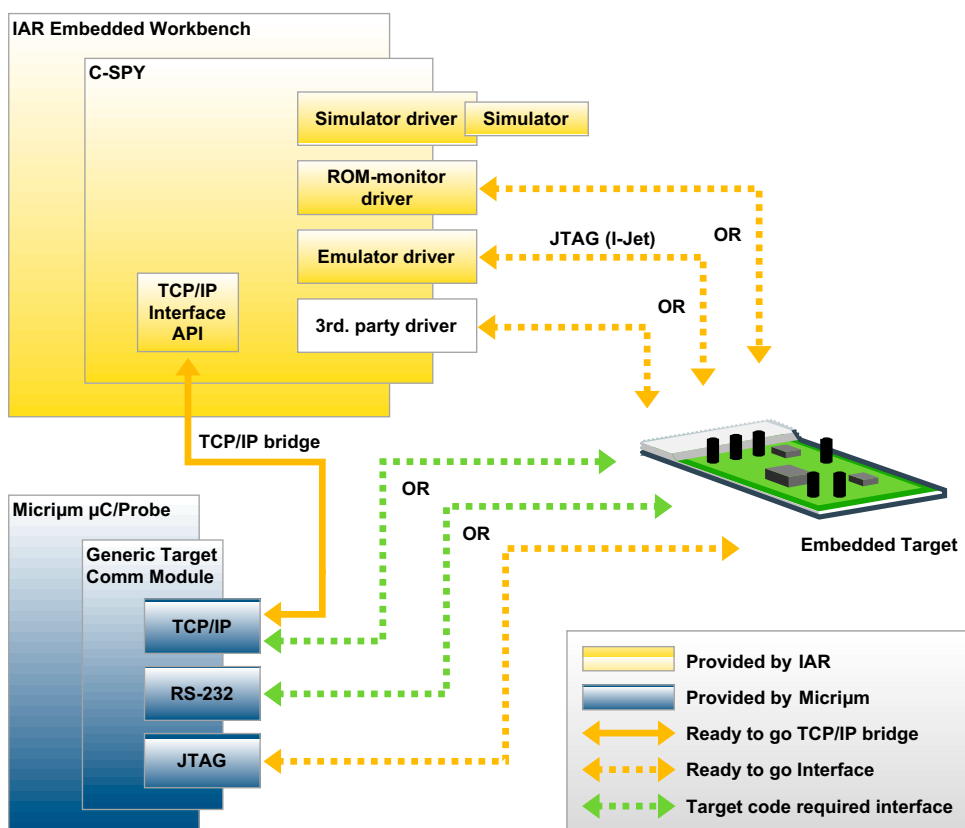


Figure 9-4 Communication Interfaces Supported by IAR Systems C-SPY and  $\mu$ C/Probe

In regards to the communication interface options illustrated in Figure 9-4, some of them, such as the ones based on JTAG are ready-to-go assuming your embedded target has either an in-circuit debugger or an external JTAG probe (i.e. IAR's I-Jet). Other communication interfaces require some resident code running in the embedded target, which is available by Micrium for most platforms.



## 9-3-1 CONFIGURING THE TCP/IP BRIDGE BETWEEN IAR C-SPY AND $\mu$ C/PROBE

The TCP/IP bridge between C-SPY® and  $\mu$ C/Probe is built in the form of a plugin module delivered with the Embedded Workbench product installation.

In order to configure Embedded Workbench to load the plugin, you open your project's debugger options and select the plugin from the list of available plugins as shown in Figure 9-5:

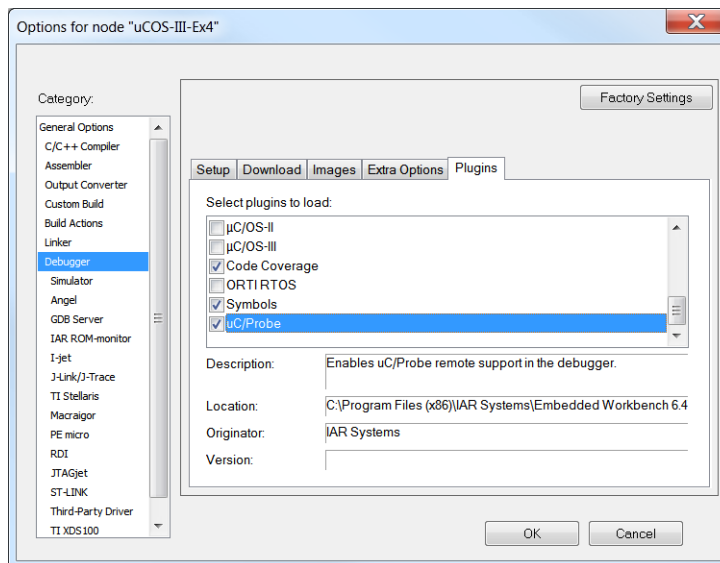


Figure 9-5 Configuring the IAR Systems Embedded Workbench

The plugin module gets loaded during a debug session and opens a TCP/IP socket on localhost to listen for  $\mu$ C/Probe requests.

Depending on your network security settings, the first time you launch a debug session you may be asked to allow Embedded Workbench to open a TCP/IP connection.

At the same time,  $\mu$ C/Probe needs to be configured to connect through its TCP/IP interface to localhost on port 9930 as described in section 4-3-5 "TCP/IP" on page 43 and as shown in Figure 9-6:

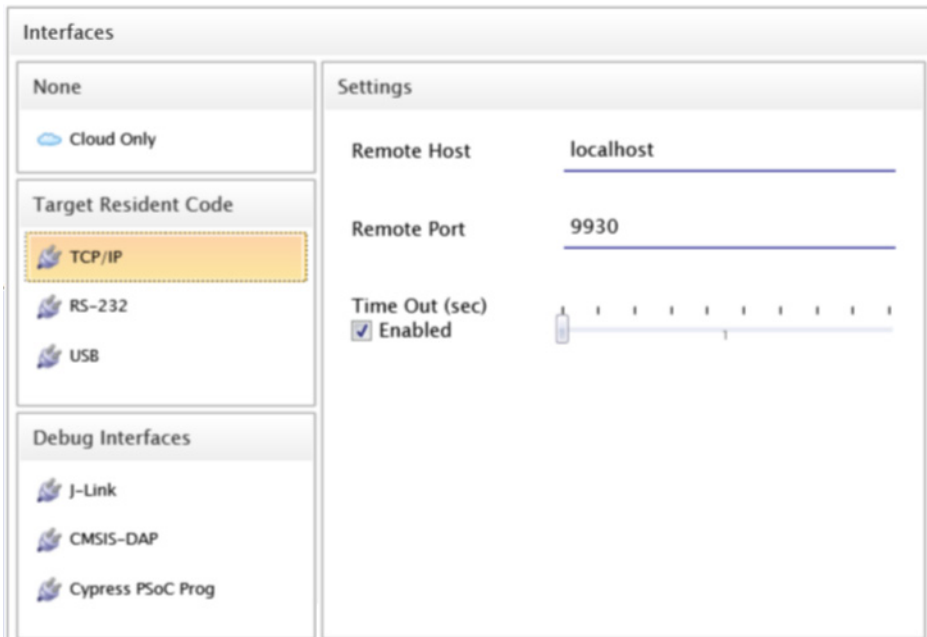


Figure 9-6 Configuring  $\mu$ C/Probe to Interface with IAR Systems C-SPY plugin for  $\mu$ C/Probe

## A

## Configuring Virtual Controls and Indicators

Once you drag and drop one of the virtual controls or indicators onto the data screen and associate it with one of the embedded target's symbols from the symbol browser, you can access the properties tool bar by moving the mouse over the virtual control or indicator.

The tool bar shown in Figure A-1 appears for you to select between one of the three configuration categories:

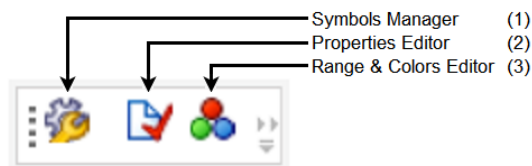


Figure A-1 **Virtual Controls and Indicators Toolbar**

FA-1(1) The Symbols Manager is common for all virtual controls and indicators, see Chapter 8, “Associating Symbols to Virtual Controls and Indicators” on page 58 for more information on the Symbols Manager.

FA-1(2) The Properties Editor is similar among most of the virtual controls and indicators and the next sections will describe how to use the Properties Editor for just a few of the most representative virtual controls and indicators.

FA-1(3) The Range and Colors Editor is only available to those virtual indicators that feature a multi-color scale. The next sections will describe how to use the Range and Colors Editor for a few of the most representative virtual indicators.

---

## A-1 VIRTUAL INDICATORS

### A-1-1 FORMATTING PROPERTIES EDITOR

The virtual indicators formatting category applies to linear gauges, half donuts, cylinders, numeric indicators, thermometers, graphs and any virtual indicator capable of showing the symbol's value in a graphical or text format. Figure A-2 shows the formatting category of a linear gauge:

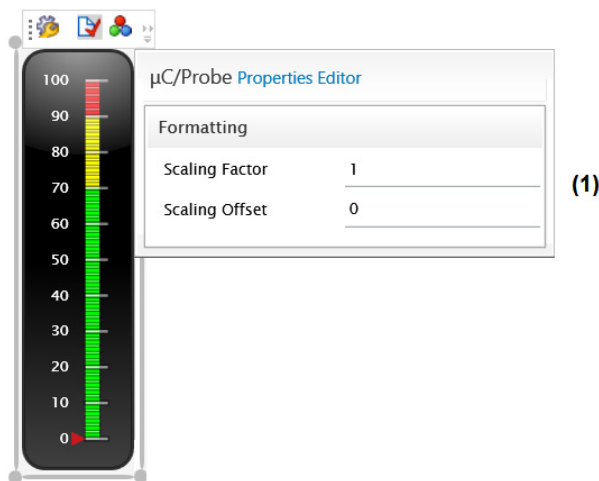


Figure A-2 **Formatting Properties Editor**

FA-2(1) In case you need to convert the value to Engineering Units (EU) before displaying in the virtual indicator, you can use the scaling factor and offset to specify the parameters of a linear conversion function. For example, if the embedded target's symbol you need to display is a 4-20mA value, you can implement the standard linear equation  $y = mx + b$  where  $m$  is the scaling factor,  $x$  is the 4-20mA value,  $b$  is the offset and  $y$  is the resulting Engineering Units (EU) value to display.

---

## A-1-2 RANGE AND COLORS EDITOR

The Range and Colors Editor applies to linear gauges, half donuts, cylinders and any virtual indicator capable of displaying the symbol's value in a graphical format along a multi-color scale. Figure A-3 shows the Range and Colors Editor for a linear gauge:

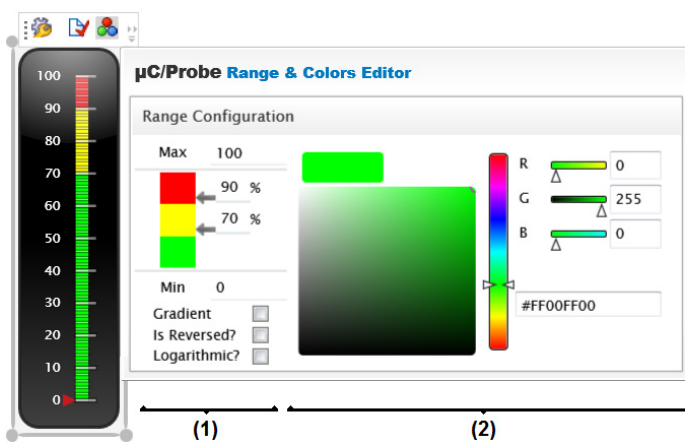


Figure A-3 Range and Colors Editor

FA-3(1) Start by setting the **Max** limit, then click on the text boxes next to the percentage signs and enter the thresholds in terms of percentage.

At the bottom of this section, the checkboxes allow you to configure the type of scale and its appearance.

FA-3(2) Each time you click on one of the text boxes to set the percentages, the color picker allows you to choose the color for that gauge band. You can enter the color you want in hex format or by selecting a color from the vertical slider and then fine tuning with the palette.

---

### A-1-3 NUMERIC INDICATOR PROPERTIES EDITOR

The Numeric Indicator category from the Properties Editor only applies to numeric indicators. Figure A-4 shows the numeric indicator's properties. Font styles, alignment and the thousand separator, they all apply to the number 0 shown in white:

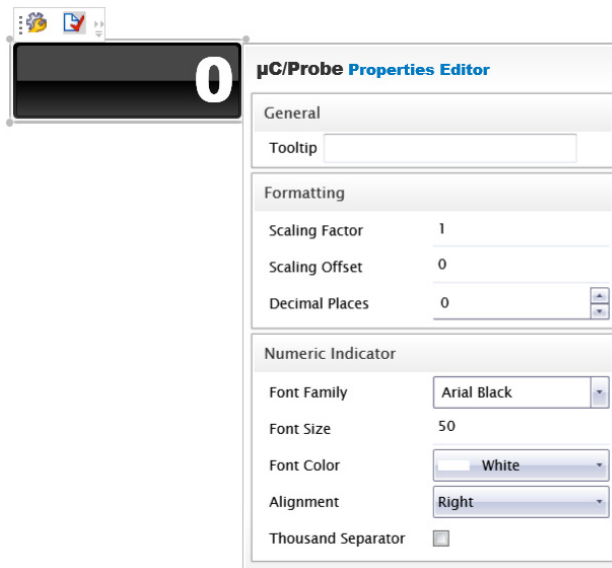


Figure A-4 Numeric Indicator Properties Editor

## A-1-4 LED PROPERTIES EDITOR

The LED control allows you to display a circle, triangle or rectangle with its color mapped to the value of one of your embedded target's global variable. This control works as a virtual LED as illustrated in the example in Figure A-5 where the LED control is configured to turn bright red when the value is 1 and dark red when the value is 0.

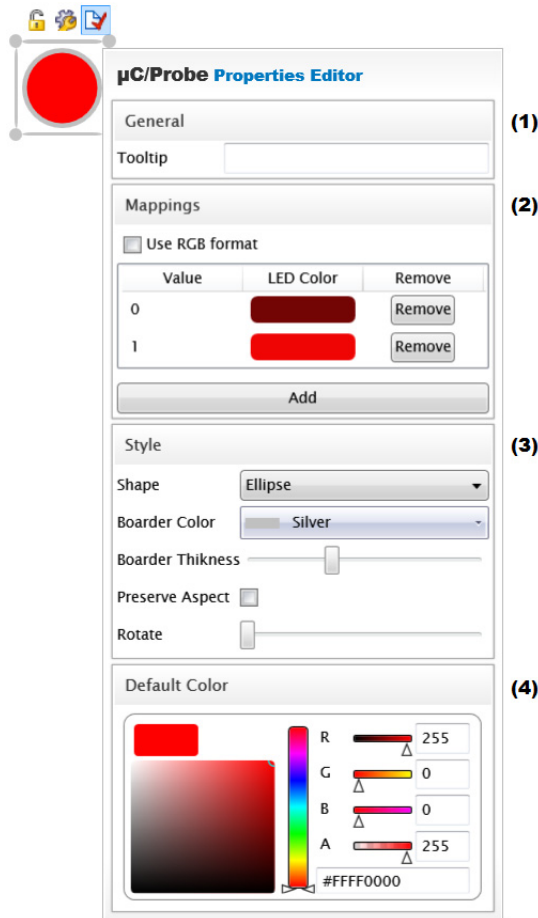


Figure A-5 LED Properties Editor

FA-5(1) You can include an optional **Tooltip** message that will appear with any information you want to display about the item being hovered over.

---

FA-5(2) The **Mappings** section is where you get to make the association between the LED's color and the value of the symbol.

The checkbox labeled as **Use RGB Format** can be used to associate this control to a global variable that stores a color code encoded in the ARGB additive color model format. The ARGB format is a 32-bit value where each byte is a channel representing the intensity of the channels Alpha, Red, Green and Blue. For example, if the variable's value is 0xFFFFF00 then the LED would turn Yellow.

FA-5(3) The **Style** section allows you to configure the LED's appearance by modifying the shape and border.

FA-5(4) You can also specify which color to show by default when communication with the embedded target has not been established yet.

## A-1-5 BITMAP ANIMATION PROPERTIES EDITOR

The bitmap animations are part of the toolbox's advanced group. They are one of the most powerful virtual indicators because you have the freedom to customize it however you want by providing your own images.

Imagine you want to display the state of a valve to be either open or closed in a graphical way by using the bitmap images shown in Figure A-6 and an embedded target's application variable named `AppValveOutPct` that stores the state of the outflow valve (0%:open and 100%:closed).

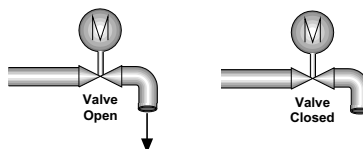


Figure A-6 **Bitmaps to Animate**

Figure A-7 shows the properties editor for the bitmap animation:



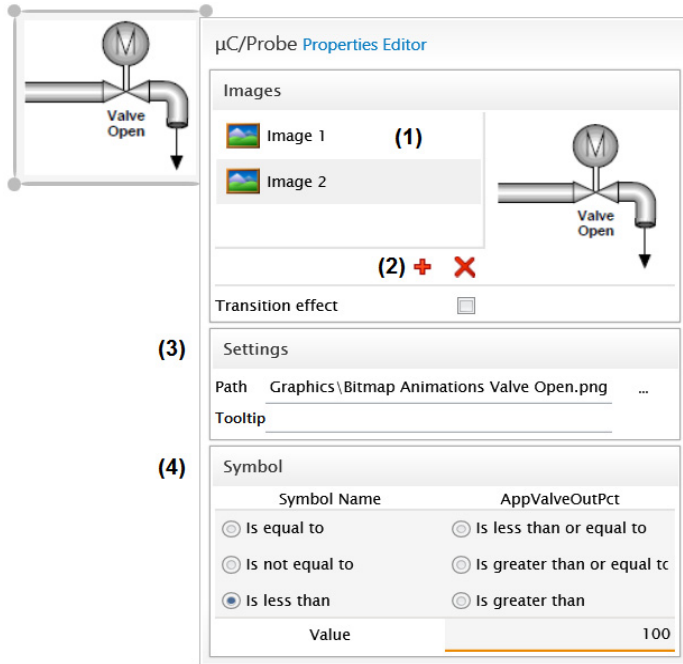


Figure A-7 **Bitmap Animation Properties Editor**

- FA-7(1) The list of images is initialized with two images. Start by selecting the image you want to work with. A preview of the image is shown on the right side.
- FA-7(2) You can add or delete more images into the list by making click on the red **+** or **x** buttons respectively.
- Select the transition effect check box if you want to add a fade-in and fade-out effect between image transitions.
- FA-7(3) Specify the bitmap file **Path** or browse to it. Additionally you can include an optional **Tooltip** message that will appear with any information you want to display about the item being hovered over.
- FA-7(4) Specify the rules. In this case, if `AppValveOutPct < 100` then the valve is open, and if `AppValveOutPct = 100` then the valve is all the way closed.

---

## A-2 VIRTUAL CONTROLS

### A-2-1 SLIDER CONTROL PROPERTIES EDITOR

The slider control is one of the few writable controls. It allows you to gradually modify an adjustable embedded target symbol's value. The user gets to select from a range of values by moving a value indicator up and down a track. For example, you typically create a volume control by using a slider control.

Figure A-8 shows the slider control properties editor:

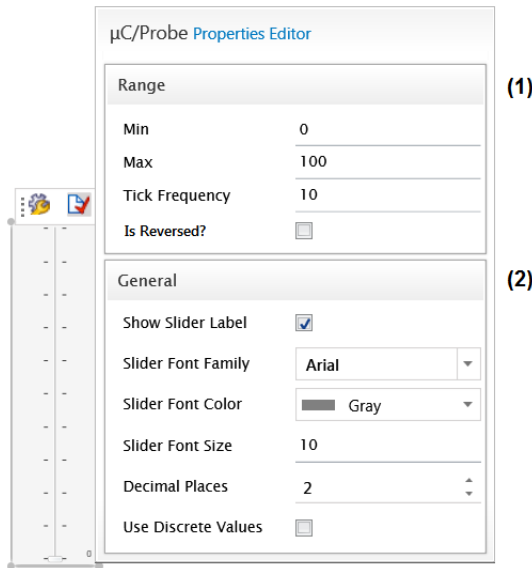


Figure A-8 Slider Control Properties Editor

FA-8(1) The slider control has a minimum, a maximum, and an increment value. The **Tick Frequency** not only determines the increment value but also the number of tick marks along the track. You can reverse the scale by ticking the checkbox.

FA-8(2) The **General** category includes the formatting properties that affect the slider's tick labels. Select the check box **Use Discrete Values** if you want the slider to adjust the associated symbol by making discrete increments.

## A-2-2 CUSTOM SLIDER PROPERTIES EDITOR

The custom slider is similar to the one from section A-2-1, except that it also allows you to include two images to the left and right side of the slider's track as shown in Figure A-9:

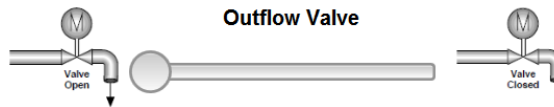


Figure A-9 Custom Slider Example


Imagine you want to control the state of a valve to be either open or closed by modifying an embedded target's application variable named `AppValveOutPct` that stores the state of the outflow valve (0%:open and 100%:closed). Figure A-10 shows the custom slider properties editor for such example.:

**µC/Probe Properties Editor**

(1) **General**  
Tooltip

(2) **Range**  
Min 0  
Max 100

(3) **Images**  
Show Images

(4) **Image on the left side**  
Path    


**Image on the right side**  
Path    

Figure A-10 Custom Slider Properties Editor

- 
- FA-10(1) You can include an optional **Tooltip** message that will appear with any information you want to display about the item being hovered over.
  - FA-10(2) The custom slider control has a minimum and maximum range.
  - FA-10(3) You can turn on and off the images.
  - FA-10(4) Specify the path or browse to the bitmap you want to be placed on the left and right sides of the track.

### A-2-3 CUSTOM SWITCH PROPERTIES EDITOR

The custom switch control is a two state button. You can modify the value of an embedded target's symbol by specifying the values you want to write when the button is switched between the **On** and **Off** states as shown in Figure A-11:

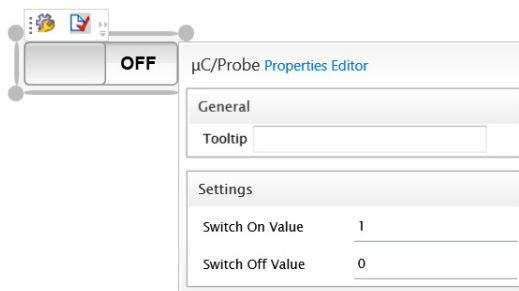


Figure A-11 Custom Switch Properties Editor

---

## A-2-4 CHECKBOX PROPERTIES EDITOR

The checkbox control is similar to the custom switch but it also allows you to specify a label to display when the checkbox is selected and not selected. In the example shown in Figure A-12 such labels are set to Enabled during the On state and Disabled during the Off state.

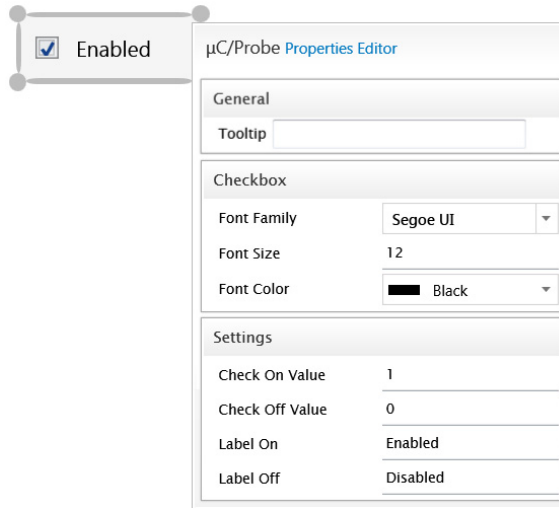


Figure A-12 Checkbox Properties Editor

---

## A-2-5 PUSH BUTTON PROPERTIES EDITOR

The push button control is a momentary switch that switches between the states **On** while held down and **Off** when released. The properties window is shown in Figure A-13.



Figure A-13 Push Button Properties Editor

- FA-13(1) You can include an optional **Tooltip** message that will appear with any information you want to display about the item being hovered over.
- FA-13(2) You can specify the labels to display during the **On** and **Off** states. These properties affect the label's font family, font size and font color properties.
- FA-13(3) During run-time, when the user holds down the push button, µC/Probe writes the **Push On Value** one single time to the embedded target.
- FA-13(4) When the user releases the button, µC/Probe writes the **Push Off Value** one single time to the embedded target.

FA-13(5) Here you specify the labels you want to display during the **On** and **Off** states.

FA-13(6) The keyboard shortcut is a sequence or combination of keystrokes on the keyboard which will invoke the Click event on the Push Button.

FA-13(7) You can also specify the border colors you want to display during the **On** and **Off** states and whether or not you want to show the **Push** label on the corner.

## A-2-6 TOGGLE BUTTON PROPERTIES EDITOR

The toggle button control is a button that switches between the states **On** and **Off** when clicked. The properties window for a relay's toggle button is shown in Figure A-14:

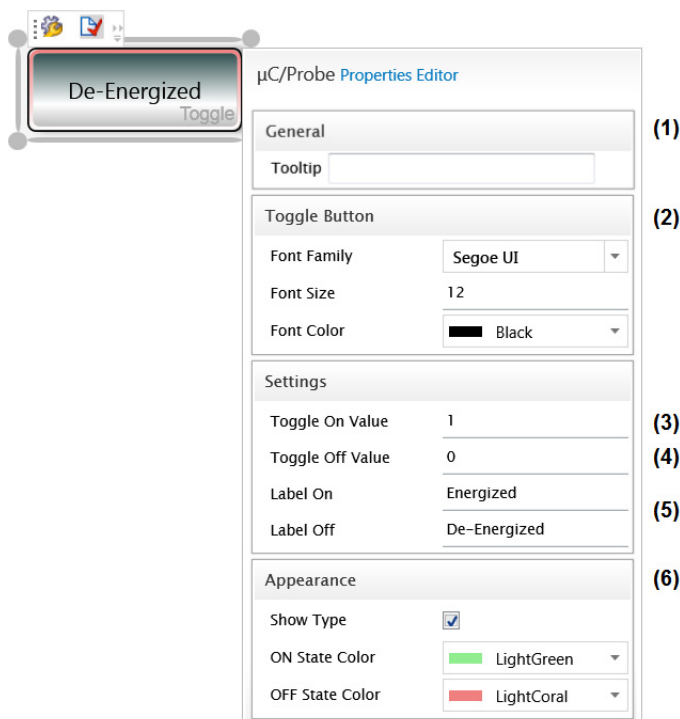


Figure A-14 Toggle Button Properties Editor

FA-14(1) You can include an optional **Tooltip** message that will appear with any information you want to display about the item being hovered over.

- 
- FA-14(2) You can specify the labels to display during the **On** and **Off** states. These properties affect the label's font family, font size and font color properties.
- FA-14(3) During run-time, when the user clicks and releases the button,  $\mu$ C/Probe writes the **Toggle On Value** one single time to the embedded target.
- FA-14(4) When the user clicks and releases the button again,  $\mu$ C/Probe writes the **Toggle Off Value** one single time to the embedded target.
- FA-14(5) Here you specify the labels you want to display during the **On** and **Off** states.
- FA-14(6) You can specify the border colors you want to display during the **On** and **Off** states and whether or not you want to show the **Toggle** label on the corner.



---

## A-2-7 REPEAT BUTTON PROPERTIES EDITOR

The repeat button control is a button that switches between the states **On** while held down and **Off** when released. The properties window is shown in Figure A-15:

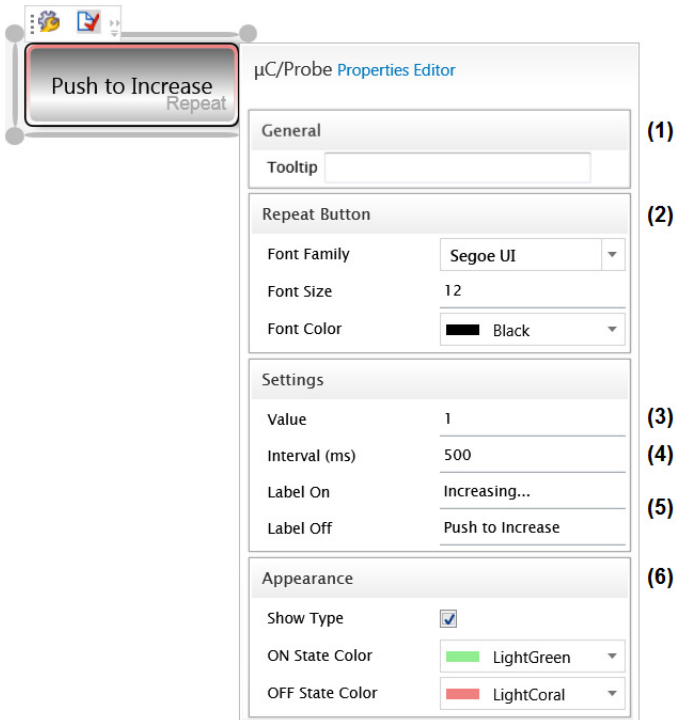


Figure A-15 Repeat Button Properties Editor

- FA-15(1) You can include an optional **Tooltip** message that will appear with any information you want to display about the item being hovered over.
- FA-15(2) You can specify the labels to display during the **On** and **Off** states. These properties affect the label's font family, font size and font color properties.
- FA-15(3) During run-time, when the user holds down the repeat button, µC/Probe writes the **Value** multiple times to the embedded target until the button is released.
- FA-15(4) The value gets written to the embedded target multiple times at the specified interval in milliseconds.

FA-15(5) Here you specify the labels you want to display during the **On** and **Off** states.

FA-15(6) You can specify the border colors you want to display during the **On** and **Off** states and whether or not you want to show the **Repeat** label on the corner.

## A-2-8 BIT CONTROL PROPERTIES EDITOR

The bit control is part of the writable controls category in  $\mu$ C/Probe's toolbox. It allows you to read and write to a symbol by either toggling its bits on and off or entering the value in either decimal or hexadecimal format.

This control is perfect to represent peripheral I/O registers and the properties window is shown in Figure A-16:

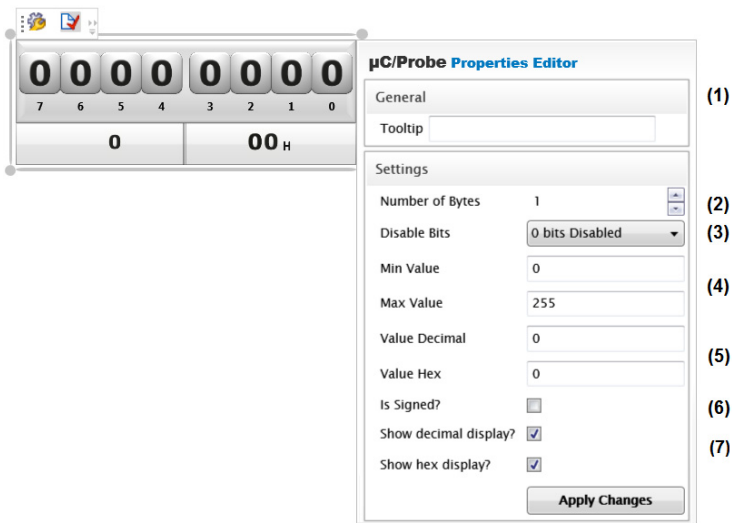


Figure A-16 Bit Control Properties Editor

FA-16(1) You can include an optional **Tooltip** message that will appear with any information you want to display about the item being hovered over.

FA-16(2) You can specify the number of bytes in case you want to override the original data size.

- 
- FA-16(3) You can also disable bits, in those cases where some of the bits are reserved which is typically the case with peripheral I/O registers. This drop-down gives you a list of all the bits along with a checkbox to enable and disable each bit.
  - FA-16(4) You can specify the minimum and maximum value range allowed.
  - FA-16(5) You can specify an initial value in either decimal or hexadecimal format.
  - FA-16(6) You can override the data type and specify whether it is signed or unsigned.
  - FA-16(7) Finally, you can hide the decimal and hexadecimal displays in case you want to show the bit buttons only.

## A-2-9 NUMERIC UP/DOWN CONTROL PROPERTIES EDITOR

The numeric up/down control is part of the writable controls category in  $\mu$ C/Probe's toolbox. It allows you to write a number to the embedded target. It consists of a single line input text field and a pair of arrow buttons for stepping up or down as shown in Figure A-17:

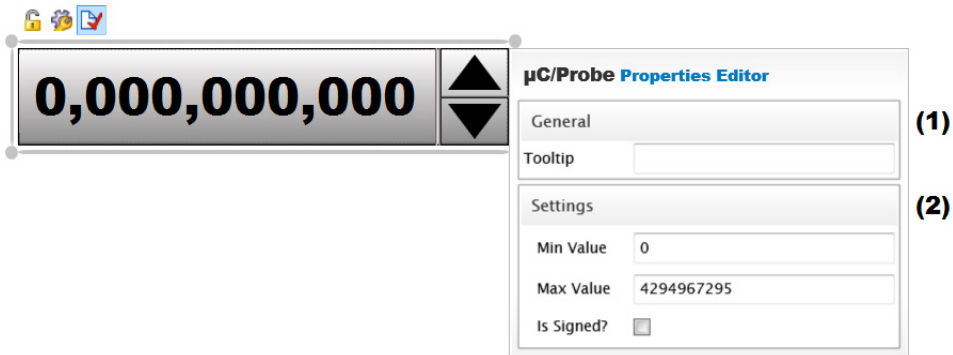


Figure A-17 Numeric Up/Down Control Properties Editor

- FA-17(1) You can include an optional **Tooltip** message that will appear with any information you want to display about the item being hovered over.
- FA-17(2) You can specify the maximum and minimum range.

## A-2-10 TEXT BOX CONTROL PROPERTIES EDITOR

The text box control is part of the writable controls category in  $\mu$ C/Probe's toolbox. It allows you to write a number to the embedded target. It consists of a single line input text field as shown in Figure A-17:

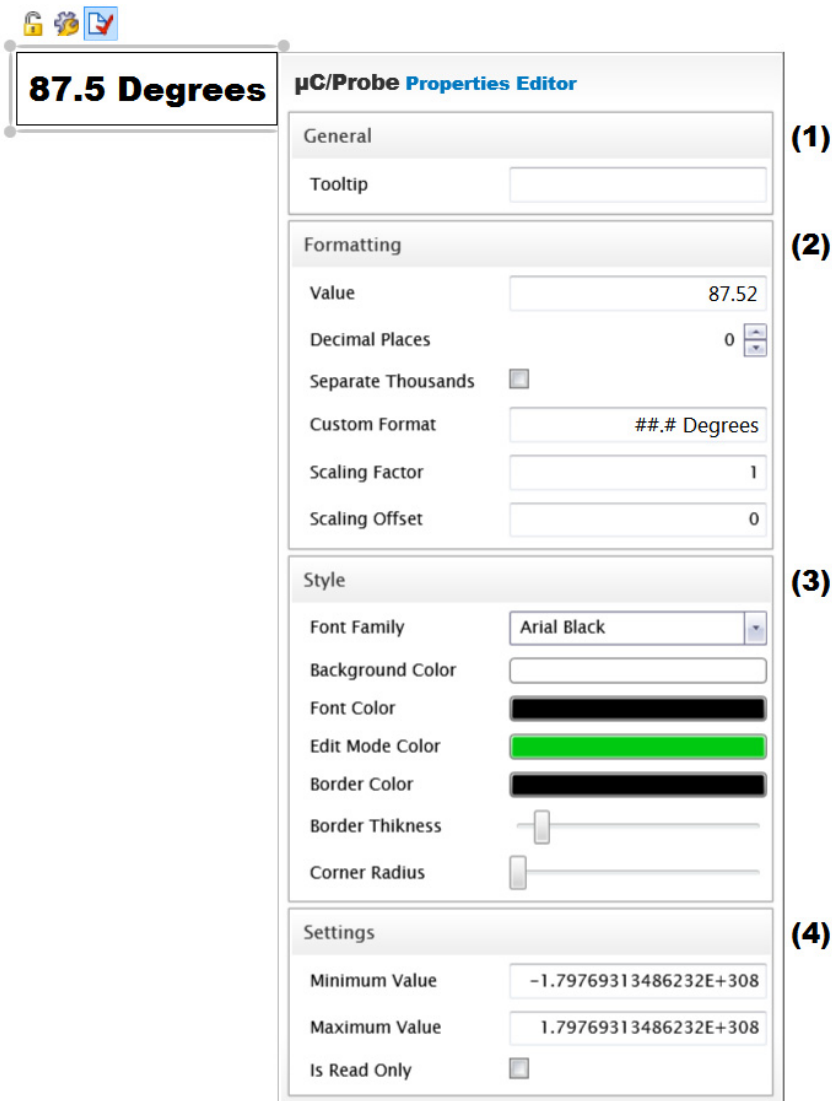


Figure A-18 Text Box Control Properties Editor

---

FA-18(1) You can include an optional **Tooltip** message that will appear with any information you want to display about the item being hovered over.

FA-18(2) You can specify an initial value, scaling options and a custom format. Standard numeric format strings are used to format common numeric types. A standard numeric format string takes the form *Axx*, where:

*A* is a single alphabetic character called the format specifier. Any numeric format string that contains more than one alphabetic character, including white space, is interpreted as a custom numeric format string.

*xx* is an optional integer called the precision specifier. The precision specifier ranges from 0 to 99 and affects the number of digits in the result. Note that the precision specifier controls the number of digits in the string representation of a number. It does not round the number itself.

Here is some of the most typical examples of custom format strings:

The value 123.456 with a custom format string of “C” results in “\$123.46”.

The value 1052.0329112756 with a custom format string of “E” results in “1.052033E+003”.

The value 255 with a custom format string of “X” results in “FF”.

Or, as shown in the example: The value 87.52 with a custom format string of “##.# Degrees” results in “87.5 Degrees”.

For more information on custom format strings search the Microsoft documentation on *Custom Format Strings*.

FA-18(3) You can configure the style by modifying font, colors and borders.

FA-18(4) You can also specify the maximum and minimum ranges and whether or not you want to restrict access to read-only.

---

## A-2-11 RGB COLOR PALETTE PROPERTIES EDITOR

This control is a color picker that allows you to select a color from a palette. When you select a color, the color is encoded in the ARGB additive color model format and written to the associated global variable in the embedded target.

The ARGB format is a 32-bit value where each of the 4 channels (Alpha, Red, Green and Blue) is a number between 0 and 255 that specifies the intensity of each color in the mixture, from fully-off (0) to fully-on (255). The alpha channel represents the opacity of the entire mixture of colors.

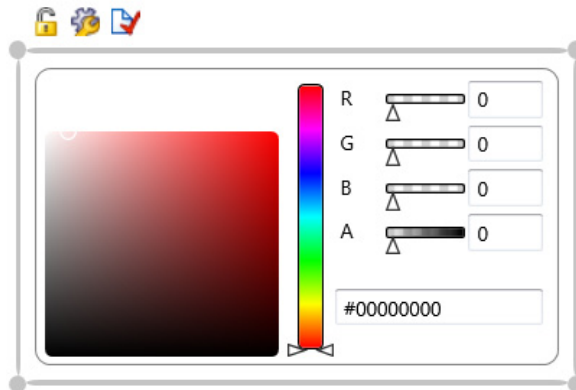


Figure A-19 RGB Color Palette

---

## A-3 CHARTS

$\mu$ C/Probe supports two types of charts; timeline charts and scatter x-y charts.

### A-3-1 TIMELINE CHARTS

Timeline charts are those whose samples are considered events in time and the horizontal axis represents the time. Figure A-20 shows the three types of timeline charts supported by  $\mu$ C/Probe:

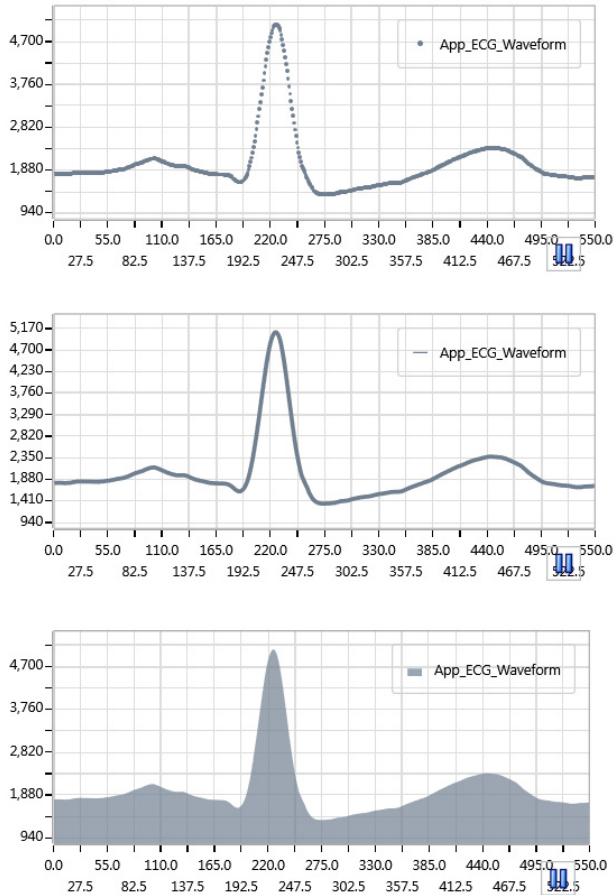


Figure A-20 Marker, Line and Area Charts

## TIMELINE CHART PROPERTIES EDITOR

All three timeline chart types share the same properties editor as shown in Figure A-21:



Figure A-21 **Charts Properties Editor**

FA-21(1) You can include an optional **Tooltip** message that will appear with any information you want to display about the item being hovered over.

FA-21(2) In order to change the number at which the vertical axis value starts or ends, enter a different number in the **Min** box or the **Max** box.

In order to change the interval of tick marks and chart grid lines, enter a different number in the **Tick Marks Step** box.

FA-21(3) In order to change the number at which the horizontal axis value starts or ends, enter a different number in the **Offset** box or the **Max Samples** box.



---

FA-21(4)  $\mu$ C/Probe updates charts in one of three modes:

- **Strip** Mode: similar to a chart recorder device that prints over a paper strip,  $\mu$ C/Probe first plots points from the left to the right side of the chart. From there, any new points are plotted at the rightmost side of the chart by shifting old points to the left.
- **Scope** Mode: similar to an oscilloscope,  $\mu$ C/Probe first plots points from the left to the right side of the chart. Once the plot reaches the right side of the chart, it erases the plot and begins plotting again from the left side of the chart.
- **Burst** Mode: this update mode was made for high performance applications where you want to plot array data quickly by plotting the entire array in one sweep.  $\mu$ C/Probe will not shift any points over the plotting area. Instead, it will erase the plot and will plot the same array again, assuming that the array is being updated by the embedded target.

FA-21(5)  $\mu$ C/Probe supports charts with multiple data series. That means that you can associate multiple symbols from your embedded target into one single chart.

In order to tell which trace represents a symbol in your chart, a color-coded legend with the name of the symbol is displayed over the chart. Select the legend position that better suits your needs.

FA-21(6) In case you need to convert the value points to Engineering Units (EU) before plotting in the chart, you can use the scaling factor and offset to specify the parameters of a linear conversion function. For example, if the embedded target's symbol you need to display is a 4-20mA value, you can implement the standard linear equation  $y = mx + b$  where  $m$  is the scaling factor,  $x$  is the 4-20mA value,  $b$  is the offset and  $y$  is the resulting Engineering Units (EU) value to be plotted.

## TIMELINE CHART SERIES EDITOR

The charts series editor allows you to configure each trace in the plotting area. The series editor is shared by all three types of charts as shown in Figure A-22:

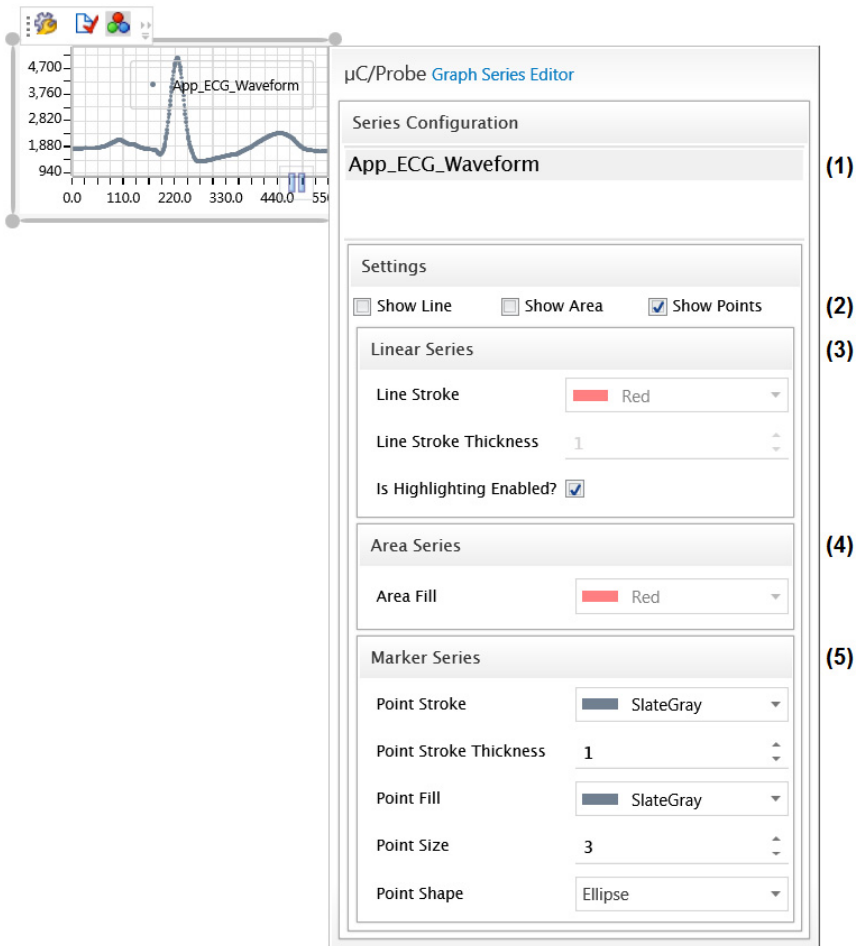


Figure A-22 **Charts Series Editor**

FA-22(1) Select the data series you want to work with. By default, the name of the data series is the same of the symbol as declared in your embedded target C files.

- 
- FA-22(2)  $\mu$ C/Probe supports three types of charts: marker, line and area charts. You can mix and match chart types in one single chart by selecting the corresponding check box.
- FA-22(3) If the **Show Line** check box is selected, this section allows you to configure the color and thickness of the line.
- FA-22(4) If the **Show Area** check box is selected, this section allows you to configure the color of the area.
- FA-22(5) If the **Show Points** check box is selected, this section allows you to configure the color, thickness and shape of the points.

## **A-3-2 SCATTER X-Y CHARTS**

Contrary to timeline charts where the horizontal axis is a representation of time, the scatter x-y charts allow you to use two different arrays to specify the x-y coordinates of each data point.

Use a scatter x-y chart if the data you want to plot includes pairs of values and you want to compare data points without regard of time.

The scatter x-y chart supports two modes of operation; burst and plot mode.

### **SCATTER X-Y CHART IN BURST MODE**

Use the *burst mode* to compare sets of values stored in two different arrays.

For example, imagine a blood pressure monitor that stores the diastolic pressure in mmHg for each patient. You can use the scatter x-y chart in burst mode to compare the diastolic pressure among the patients age groups as follows.

Code Listing A-1 shows the two arrays that contain the data points.

```

static const CPU_INT08U AppAgeTbl[20] =
{
    20, 28, 32, 45, 26, 64, 23, 54, 32, 54, 23, 33, 44, 21, 43, 52, 56, 62, 23, 45
};

static const CPU_INT08U AppDiastolicPressureTbl[20] =
{
    64, 67, 72, 76, 65, 100, 64, 94, 80, 90, 70, 73, 81, 62, 87, 89, 91, 98, 68, 87
};

```

Listing A-1 **Two Arrays**

First you drag-and-drop an instance of the scatter x-y chart into a data screen. Then you open the symbols file (i.e. ELF file), search for the two arrays `AppAgeTbl[]` and `AppDiastolicPressure[]`, and drag-and-drop the arrays over the scatter x-y chart.

If you open the properties of the chart you will see something similar to Figure A-23:

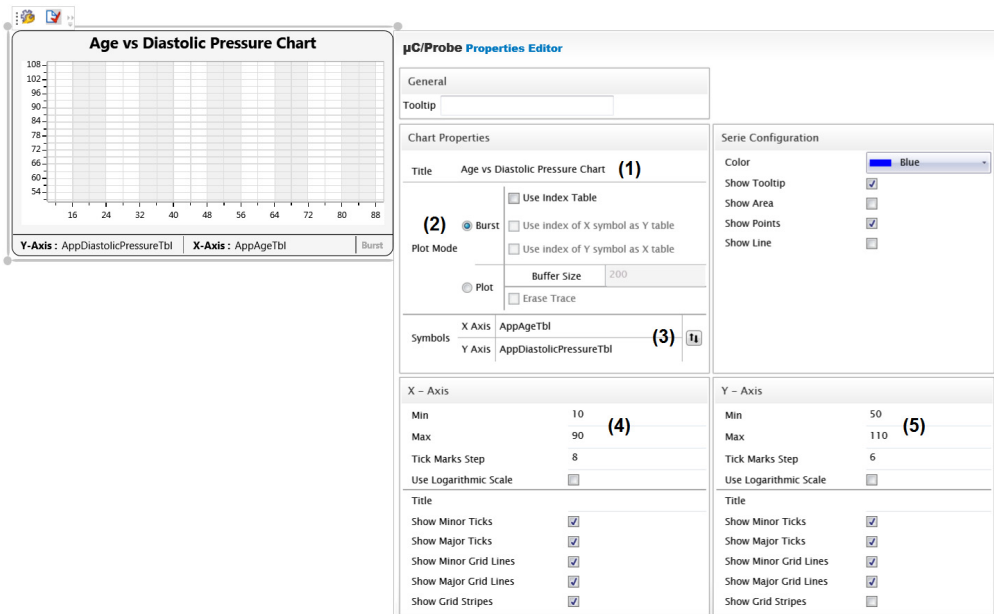


Figure A-23 **Scatter X-Y Chart Properties**

---

FA-23(1) Enter the title for your scatter x-y chart.

FA-23(2)  $\mu\text{C}/\text{Probe}$  will automatically set your plot mode to *Burst Mode* in case the axis have been mapped to arrays.

In case you only have one single array and want to use the array indexes as one of the axis, you can enable the checkbox *Use Index Table* and select which axis you want the index to be.

FA-23(3) You can use this button to swap the axis.

FA-23(4) Set the range of your data points over the horizontal axis.

FA-23(5) Set the range of your data points over the vertical axis.

During run-time,  $\mu\text{C}/\text{Probe}$  and the scatter X-Y chart should plot all the data points at once and will keep refreshing the plot as the values of the arrays change. You should see a scatter chart similar to the one in Figure A-24:

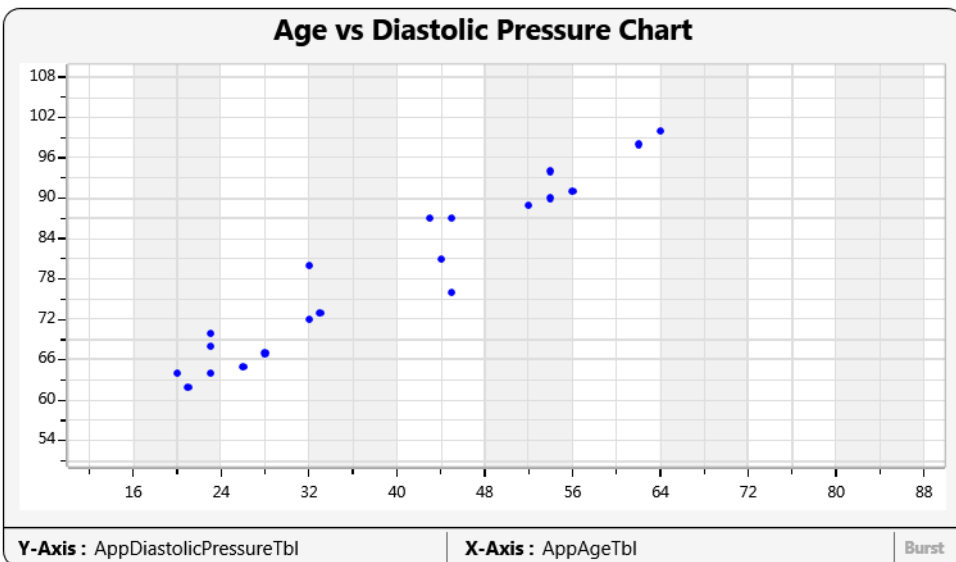


Figure A-24 Scatter X-Y Chart in Burst Mode

## SCATTER X-Y CHART IN PLOT MODE

For the *plot mode* of the X-Y scatter chart, imagine for instance you want to monitor in real-time the power curve of a wind turbine. Your embedded application does not buffer any data and instead, it only keeps the current wind speed (meters/sec) and current power (watts) stored in the global variables `AppWindSpd` and `AppPwr` respectively.

Similar to the previous example, you drag-and-drop an instance of the scatter x-y chart into a data screen. Then, you open the symbols file (i.e. ELF file), search for the two variables `AppWindSpd` and `AppPwr`, and drag-and-drop the them over the scatter x-y chart.

If you open the properties of the chart you will see something similar to Figure A-25:

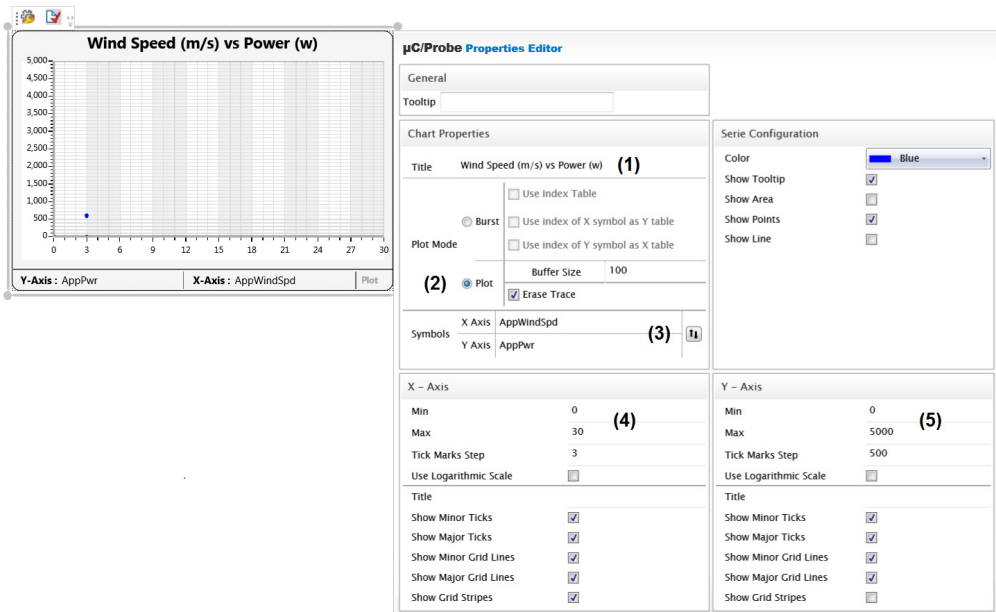


Figure A-25 Scatter X-Y Chart Properties

FA-25(1) Enter the title for your scatter x-y chart.

FA-25(2) µC/Probe will automatically set your chart to *Plot Mode* in case the axis have been mapped to a pair of non-array variables.

---

In this case, you need to specify the size of the host-side buffer that will hold the pair of values and whether or not you want the entire plot to be erased once the buffer is full.

FA-25(3) You can use this button to swap the axis.

FA-25(4) Set the range of your data points over the horizontal axis.

FA-25(5) Set the range of your data points over the vertical axis.

During run-time, the scatter chart should start to fill the plot area with one data point at a time and at the coordinate (AppWindSpd, AppPwr). Depending on the data change rate and the data collection rate, eventually the chart should look similar to the one in Figure A-26:

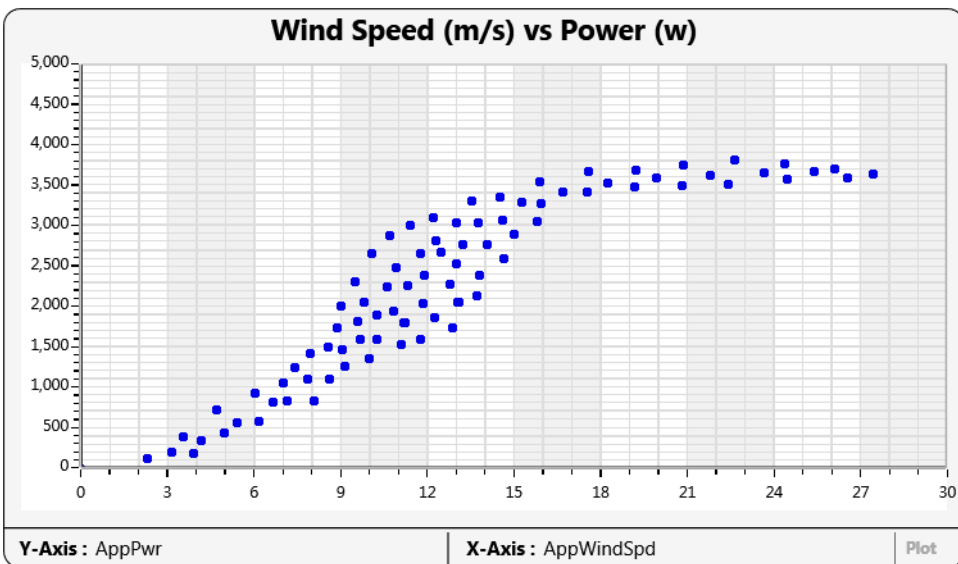


Figure A-26 Scatter X-Y Chart in Plot Mode

## Appendix

# B

## Kernel Awareness Screen

$\mu$ C/Probe allows you to add pre-configured data screens that display some of the most popular Micrium's software modules.

Figure B-1 and Figure B-2 show examples of the Kernel Awareness Screens for  $\mu$ C/OS-III. The Screens display  $\mu$ C/OS-III's internal data structures in a convenient series of windows. This provides you with information about each of the active tasks in the embedded target application among other kernel objects such as semaphores, event flags, mutexes, etc.



μC/OS-III KA		DataScreen1	
CPU Usage		1.42 %	
		μs	
<b>Miscellaneous</b>			
CPU Usage (%)		1.42	
OS Running		Yes	
Idle Task Counter		22,740,203	
Statistic Task Counter		20,382	
Tick Task Counter		100,010	
Timer Task Counter		1,000	
#Context Switches		222,956	
<b>Message Queue Pool</b>			
#Free Messages		99	
#Used Messages		1	
#Used Messages (Max)		Feature Not Enabled	
<b>μC/OS-III Task Execution Times</b>			
Tick Task		1,144	15.9
Timer Task		108	1.5
Statistics Task		35,270	489.9
Interrupt Handler Task		Feature Not Enabled	
<b>Interrupts</b>			
Nesting Counter		0	
Maximum Interrupt Disable Time		1,186	16.5
<b>Interrupt Queue</b>			
#Entries		Feature Not Enabled	
#Entries (Max)		Feature Not Enabled	
#Overflows		Feature Not Enabled	
<b>Scheduler</b>			
Round Robin Enabled?		No	
Lock Nesting Counter		0	
Maximum Lock Time		508	7.1

Figure B-1 Kernel Awareness Screen: Miscellaneous

Figure B-2 shows the information displayed for each task. The columns can be sorted and the tri-color bar graphs highlight those tasks reaching their maximum stack space, which is a typical bug when developing embedded systems:

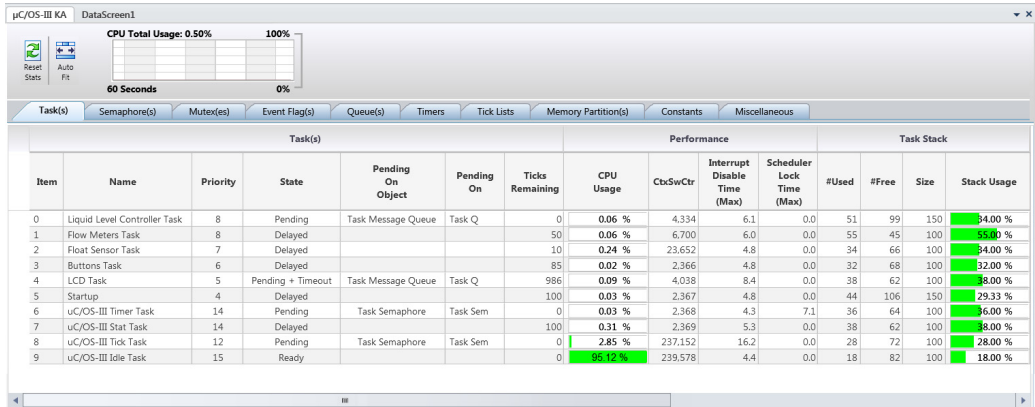


Figure B-2 Kernel Awareness Screen: Task List

## C

## Terminal Window Control

$\mu$ C/Probe provides an option to enable debug traces to output the embedded target's activity via any of the communication interfaces supported by  $\mu$ C/Probe. A trace message is displayed in a terminal window control in  $\mu$ C/Probe, by calling a function `ProbeTermTrcPrint()` from your embedded application as illustrated in Figure C-1. Additionally, you can prefix the messages with special tags that  $\mu$ C/Probe will replace with icons that you get to choose.

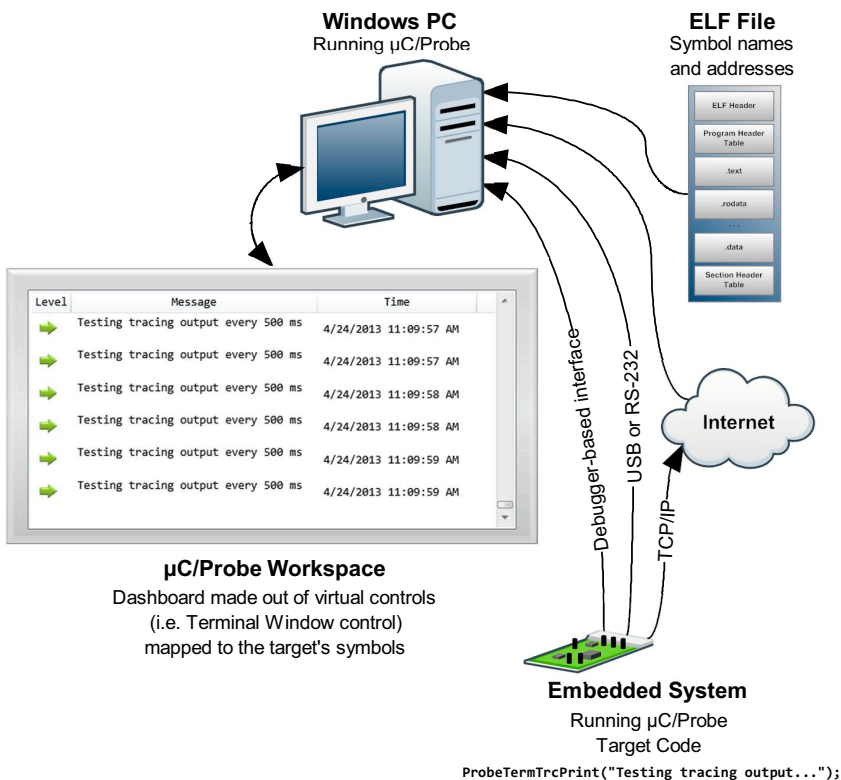


Figure C-1 Terminal Window Control - Trace Interface

At the same time,  $\mu\text{C}/\text{Probe}$  provides the option to enable a command-line interface to the embedded target. A command-line interface allows the user to issue a command to the target from a terminal window control in  $\mu\text{C}/\text{Probe}$ . Examples of command lines are `ipconfig`, `dir` or whatever command the programmer wants to implement in the embedded target.

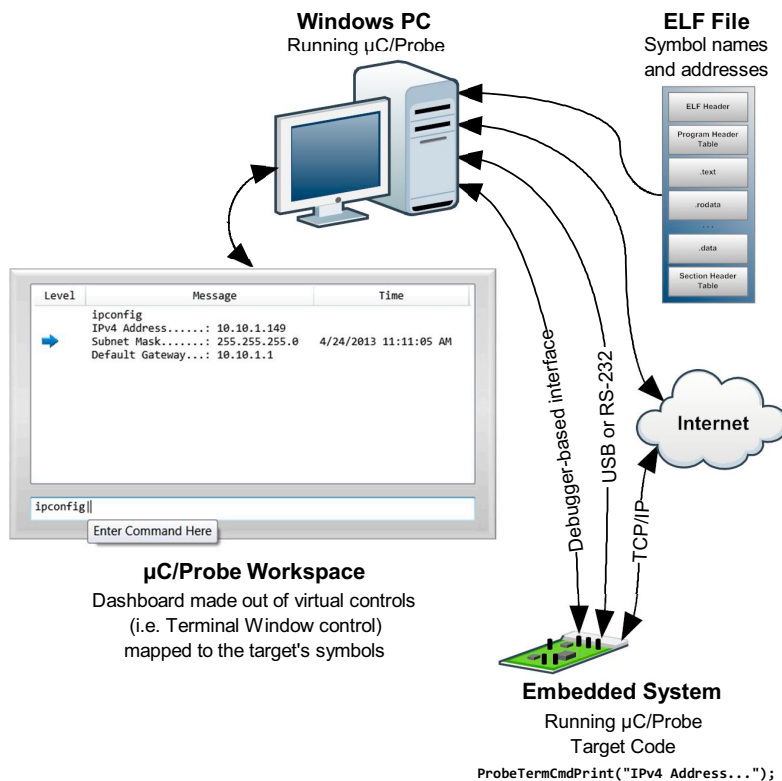


Figure C-2 **Terminal Window Control - Command Line Interface**

This appendix will introduce you to the debug trace and command-line tools available in  $\mu\text{C}/\text{Probe}$ . It will show you how to configure the control in  $\mu\text{C}/\text{Probe}$ . More information on the Terminal Window control such as how to include it in your embedded target code and make use of it are in the target version of the  $\mu\text{C}/\text{Probe}$  manual.

---

## C-1 TERMINAL WINDOW CONTROL CONFIGURATION

The terminal window control is found in the miscellaneous category of  $\mu$ C/Probe's toolbox. Once you drag and drop an instance of this control onto the data screen, you do not need to associate it with any of the embedded target's symbols from the symbol browser, as this is done automatically, assuming you have included the required target code to support terminal window as described in the target version of the  $\mu$ C/Probe manual.

You can access the properties tool bar by moving the mouse over the terminal window control. The tool bar shown in Figure C-3 appears for you to select between one of the two configuration categories:

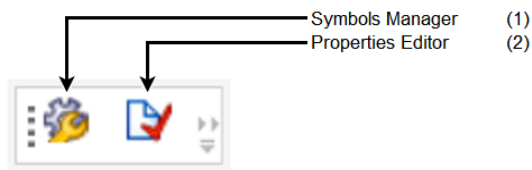


Figure C-3 Terminal Window Control Toolbar

FC-3(1) The Symbols Manager is common for all virtual controls and indicators, see Chapter 8, “Associating Symbols to Virtual Controls and Indicators” on page 37 for more information on the Symbols Manager.

FC-3(2) The Properties Editor is similar among most of the virtual controls and indicators and the next sections will describe how to use the Properties Editor.

---

## C-2 PROPERTIES EDITOR

The properties editor for the terminal window control is shown in Figure C-4:

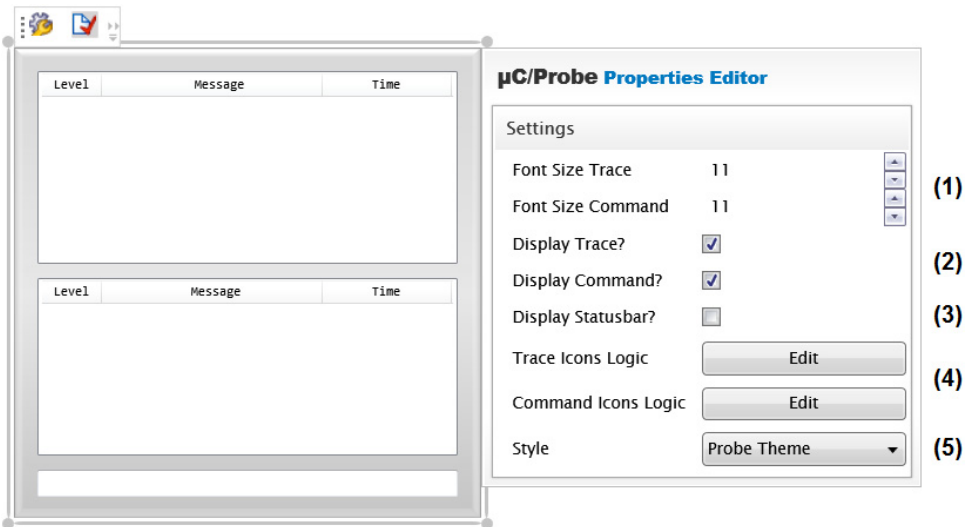


Figure C-4 Terminal Window Control Properties Editor

- FC-4(1) You can change the font size for each panel.
- FC-4(2) You can hide one of the interfaces.
- FC-4(3) The terminal window control has a status bar that allows you to see the communication status.
- FC-4(4) The icons mapping interface allows you to configure an icon to be displayed whenever a keyword is found in the message back from the target. This is useful if you want to bring more attention to a set of messages, such as warnings or errors.
- FC-4(5) The terminal window control is also available in a high contrast theme.

# D

## μC/Trace Triggers Control

μC/Trace is a runtime diagnostics tool for embedded software systems based on μC/OS-III. μC/Trace gives developers an unprecedented insight into the runtime behavior, which allows for reduced troubleshooting time and improved software quality, performance and reliability. Complex software problems which otherwise may require many hours or days to solve, can with μC/Trace be understood quickly, often in a tenth of the time otherwise required. This saves you many hours of troubleshooting time. Moreover, the increased software quality resulting from using μC/Trace can reduce the risk of defective software releases, causing damaged customer relations.

The insight provided by μC/Trace also allows you to find opportunities for optimizing your software. You might have unnecessary resource conflicts in your software, which are "low hanging fruit" for optimization and where a minor change can give a significant improvement in real-time responsiveness and user-perceived performance. By using μC/Trace, software developers can reduce their troubleshooting time and thereby get more time for developing new valuable features. This means a general increase in development efficiency and a better ability to deliver high-quality embedded software within budget.

μC/Trace provides more than 20 interconnected views of the runtime behavior, including task scheduling and timing, interrupts, interaction between tasks, as well as user events generated from your application as shown in Figure D-1. μC/Trace can be used side-by-side with a traditional debugger and complements the debugger view with a higher level perspective, ideal for understanding the complex errors where a debugger's perspective is too narrow.

μC/Trace is more than just a viewer. It contains several advanced analyses developed since 2004, that helps you faster comprehend the trace data. For instance, it connects related events, which allows you to follow messages between tasks and to find the event that triggers a particular task instance. Moreover, it provides various higher level views such as the Communication Flow graph and the CPU load graph, which make it easier to find anomalies in a trace.

$\mu$ C/Trace does not depend on additional trace hardware, which means that it can be used in deployed systems to capture rare errors which otherwise are hard to reproduce.

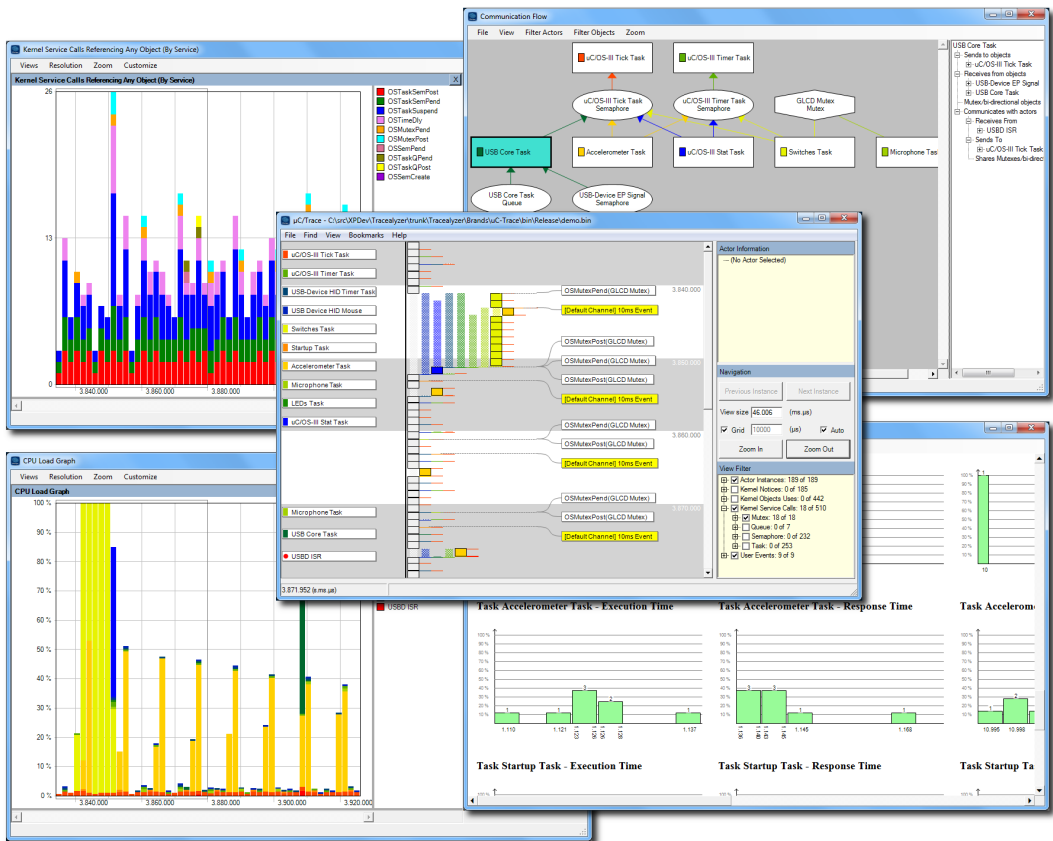


Figure D-1  $\mu$ C/Trace Analyzer Windows

The  $\mu$ C/Trace solution consists of three parts:

- The PC application ( $\mu$ C/Trace), used to analyze the recordings as shown in Figure D-1.
- A trace recorder library that integrates with  $\mu$ C/OS-III, provided in C source code.
- Optionally,  $\mu$ C/Probe can be used for the target system connection.

The PC application  $\mu$ C/Trace has been developed for Microsoft Windows.



---

The trace recorder library stores the event data in a RAM buffer, which is uploaded on request to the host PC using your existing debugger connection or  $\mu$ C/Probe.

And finally, you can use  $\mu$ C/Probe and a special control designed for  $\mu$ C/Trace called  $\mu$ C/Trace Trigger Control, to trigger a recording and launch the  $\mu$ C/Trace analyzer. The  $\mu$ C/Trace Trigger Control is shown in Figure D-2:

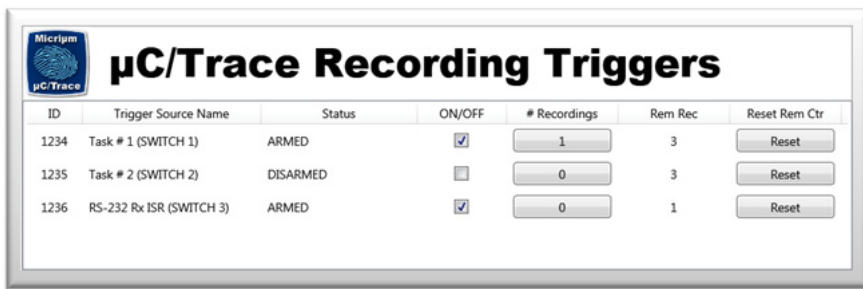


Figure D-2  $\mu$ C/Trace Trigger Control

For more information on  $\mu$ C/Trace, go online to <http://micrium.com/tools/uctrace>

# E

## Spreadsheet Control

Microsoft® Excel® is a very widely used spreadsheet application developed by the Microsoft® Corporation. It features calculation and graphing tools among other features that now you can embed in your  $\mu$ C/Probe data screens thanks to a technology offered by Microsoft® called *Automation*.

Automation to Excel® allows you to programmatically perform actions such as creating a new workbook, adding data to the workbook, or creating charts. Virtually, all of the actions that you can perform manually through the user interface can also be performed programmatically by using automation.

$\mu$ C/Probe makes use of automation to allow you to create a workbook, map the spreadsheet cells to your embedded target symbols and make use of all the features Excel® has to offer.

This appendix will show you how to drag-and-drop an instance of Excel® into a  $\mu$ C/Probe data screen and how to associate your embedded target symbols to the spreadsheet's cells.

*Microsoft®, Excel® and Windows® are either registered trademarks or trademarks of Microsoft® Corporation in the United States and/or other countries. The use of Microsoft® Excel® automation features by  $\mu$ C/Probe does not imply that Micrium and/or  $\mu$ C/Probe have any Microsoft® affiliation, sponsorship, endorsement, certification, or approval.*

## E-1 ADDING AN INSTANCE OF THE SPREADSHEET CONTROL

The Spreadsheet Control is only available on the Professional Edition of  $\mu$ C/Probe. Additionally, you need to have Microsoft<sup>®</sup> Excel<sup>®</sup> version 2003 or newer installed on your Windows<sup>®</sup> PC.

In order to add an instance of Excel<sup>®</sup>, go to the  $\mu$ C/Probe toolbox in the *Advanced* controls category and drag-and-drop the icon labeled *Spreadsheet* into a data screen as illustrated in Figure E-1:

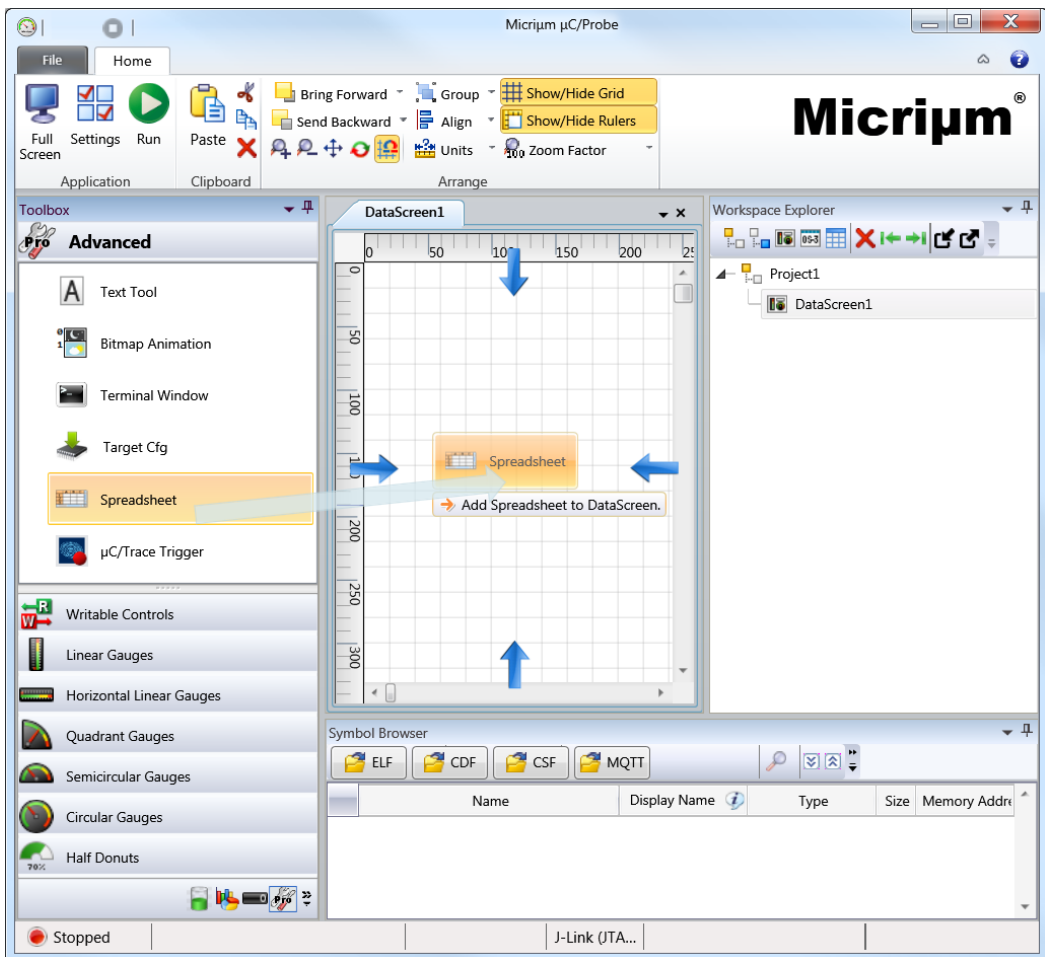


Figure E-1 Adding an Instance of the Spreadsheet Control

## E-2 CONFIGURING THE SPREADSHEET

Once you have successfully added an instance of Excel<sup>®</sup> into a data screen, it is time to associate your embedded target symbols from the *Symbol Browser* to any cell in the spreadsheet as illustrated in Figure E-2:

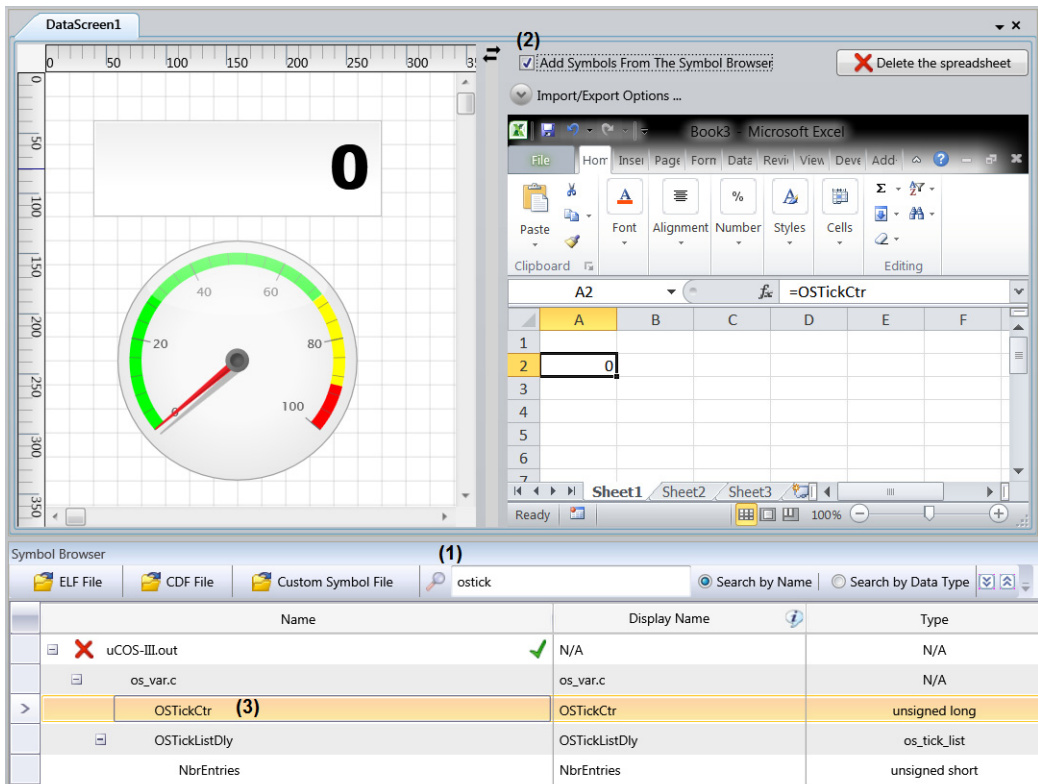


Figure E-2 Configuring the Spreadsheet

- FE-2(1) You open a symbols file (i.e. ELF file) and browse through it as described in Chapter 3, “*µC/Probe Symbol Browser*” on page 15.
- FE-2(2) Make sure the checkbox labeled *Add Symbols From The Symbol Browser* at the top is ticked and click over the cell that you want to configure.
- FE-2(3) Go back to the *Symbol Browser* and double-click over the symbol you want to configure.

## E-3 OTHER FEATURES

Figure E-3 shows you other features:

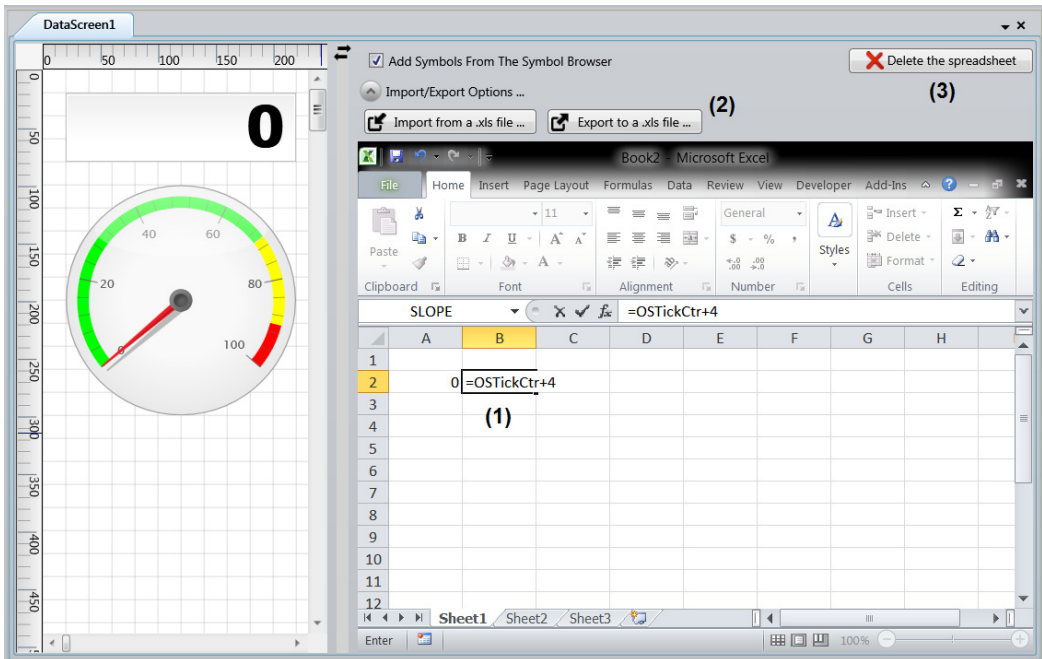


Figure E-3 Other Features

FE-3(1) Excel<sup>®</sup> formulas start by typing in the equal sign in the cell where you want the formula result to appear. As soon as you type in the equal sign, start entering the first few letters of the name of the symbol you want to use in the expression, and for your convenience, Excel<sup>®</sup> will show you a list of the symbol names you have previously added to the spreadsheet.

When you click on a cell containing a formula in Microsoft<sup>®</sup> Excel<sup>®</sup>, the formula always appears in the formula bar located above the column letters.

FE-3(2) You can import and export the spreadsheet along with the embedded target symbols mapping in the form of a regular Microsoft<sup>®</sup> Excel<sup>®</sup> file (xls).

FE-3(3) Press the button labeled *Delete Spreadsheet* to remove the spreadsheet from the data screen.

## E-4 APPLICATION EXAMPLE

A great way to use  $\mu\text{C}/\text{Probe}$  and the Spreadsheet Control is to test and calibrate an onboard accelerometer. Imagine your embedded application is reading an accelerometer's X and Y axis and storing them in the global variables `AppAcceLX` and `AppAcceLY` respectively as shown in Figure E-4:

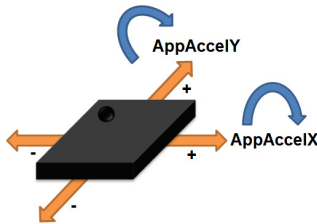


Figure E-4 Accelerometer

Very quickly you can add an instance of the spreadsheet control, map two cells to the variables `AppAcceLX` and `AppAcceLY`, insert a bubble chart from Excel<sup>®</sup> and there you have a bubble level as shown in Figure E-5:

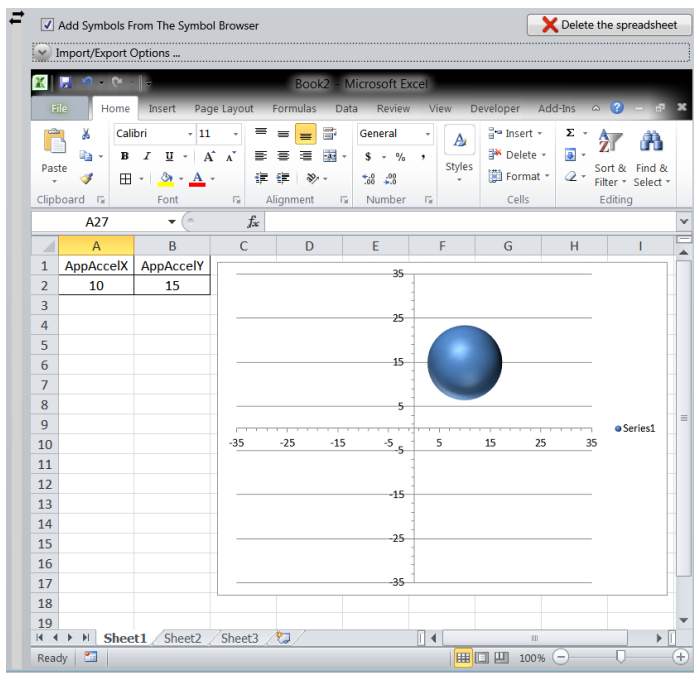


Figure E-5 Bubble Level Chart

# F

## Scripting Control

The Professional Edition of  $\mu$ C/Probe comes with a control that helps you programmatically read and write your embedded target symbols on the fly.

You write a script in a C style language to read and write your embedded target symbols and this control will execute your script whenever you want.

Use the  $\mu$ C/Probe Scripting Control to create configuration files, test scripts and to automate the execution of any other common tasks which could alternatively be executed manually by a human operator.

### **F-1 WRITING A SCRIPT**

The  $\mu$ C/Probe Scripting Control supports a programming language that is very similar to C. You write the script as a text file to manipulate variables in two different scopes; the script-side and the embedded target side as described in the following sections.

---

## F-1-1 SCRIPT-SIDE VARIABLES

You declare a variable on the script-side (host PC) by prefixing it with the keyword `var` or with the data type as shown in Code Listing F-1:

```
var max_ch = 3;                /* Declare a script-side variable by using the keyword */
                               /* 'var' and initialize it before using it.      */

string lcd_part_nbr = "";     /* Alternatively, you can declare script-side variables */
                               /* by specifying its data type (char, string, short,    */
                               /* int, uint, long, ulong, float, double, bool, etc.). */

int gain = 10;

string filter_type;          /* Furthermore, when the data type is specified, you do */
                               /* not require to initialize the variable.              */

if (max_ch == 8) {
    ...
} else {
    ...
}
```

Listing F-1 Script-Side Variables

## F-1-2 EMBEDDED TARGET SIDE VARIABLES

The variables on the embedded target side are referenced by prefixing the name of the variable with the dollar sign `$` as shown in the following Code Listing F-2:

```
$AppTimeout = 100;           /* Target-side variables are prefixed with the $ sign */
$AppFrequency = 50;         /* and they must be present in the target's ELF file. */

for (int i = 0; i < max_ch; i++) {
    $AppChannels.Ch[i].Gain = gain;
}
```

Listing F-2 Embedded Target Side Variables



---

## F-1-3 FLOW CONTROL STATEMENTS

The  $\mu$ C/Probe Scripting Control provides two styles of flow control; branching and looping.

### BRANCHING

- `if` statement
- `switch/case` statement
- `?` operator

### LOOPING

- `while` loop
- `for` loop
- `do/while` loop

It is your responsibility to ensure the loops have a terminating condition and that the terminating condition can be met. Code Listing F-3 shows an example of using some of these flow control statements:

```
if (max_ch == 8) {
    ...
} else {
    ...
}

switch ($AppSwitches) {
    case 1:
        ...
        break;
}

while ($AppStateCalibrating == true) {
    Sleep("Calibrating", 1000);
}
```

Listing F-3 Flow Control Example

---

## F-1-4 BUILT-IN INSTRUCTIONS

Besides the standard flow control statements previously mentioned, the  $\mu$ C/Probe Scripting Control provides a set of built-in instructions to help you create a better user interface.

Code Listing F-4 shows you how to use the `Sleep()` and `Pause()` built-in instructions.

```
for (int i = 0; i < max_ch; i++) {
    $AppChannels.Ch[i].Gain = 0;
                                /* The Sleep instruction allows you to delay the script */
                                /* execution (milliseconds) with a custom message.      */
    Sleep("Reaching Steady-State...", 200);
}

$AppSelfTestStart = true;

Pause("Self-Test in progress..."); /* The Pause instruction allows you to pause the script */
                                   /* execution until you press a button in  $\mu$ C/Probe.      */
if ($AppSelfTestResult < 0) {
    Abort(); /* The Abort instruction allows you to stop the script */
              /* execution.                                          */
} else {
    ...
}
```

Listing F-4 **Built-in Instructions**

## F-1-5 INCLUDING OTHER SCRIPT FILES

Similar to the directive `#include` in the C language, the  $\mu$ C/Probe Scripting Control allows you to include other script files to help you organize your scripts in modules.

When a script file is included, the code it contains inherits the variable scope of the line on which the include occurs. Any variables available at that line in the calling file will be available within the called file, from that point forward. However, all functions defined in the included file have the global scope.

## F-2 ADDING AN INSTANCE OF THE SCRIPTING CONTROL

To add an instance of the  $\mu$ C/Probe Scripting Control go to the  $\mu$ C/Probe toolbox in the *Advanced* Controls category and drag-and-drop the icon labeled *Scripting* into a data screen as shown in Figure F-1:

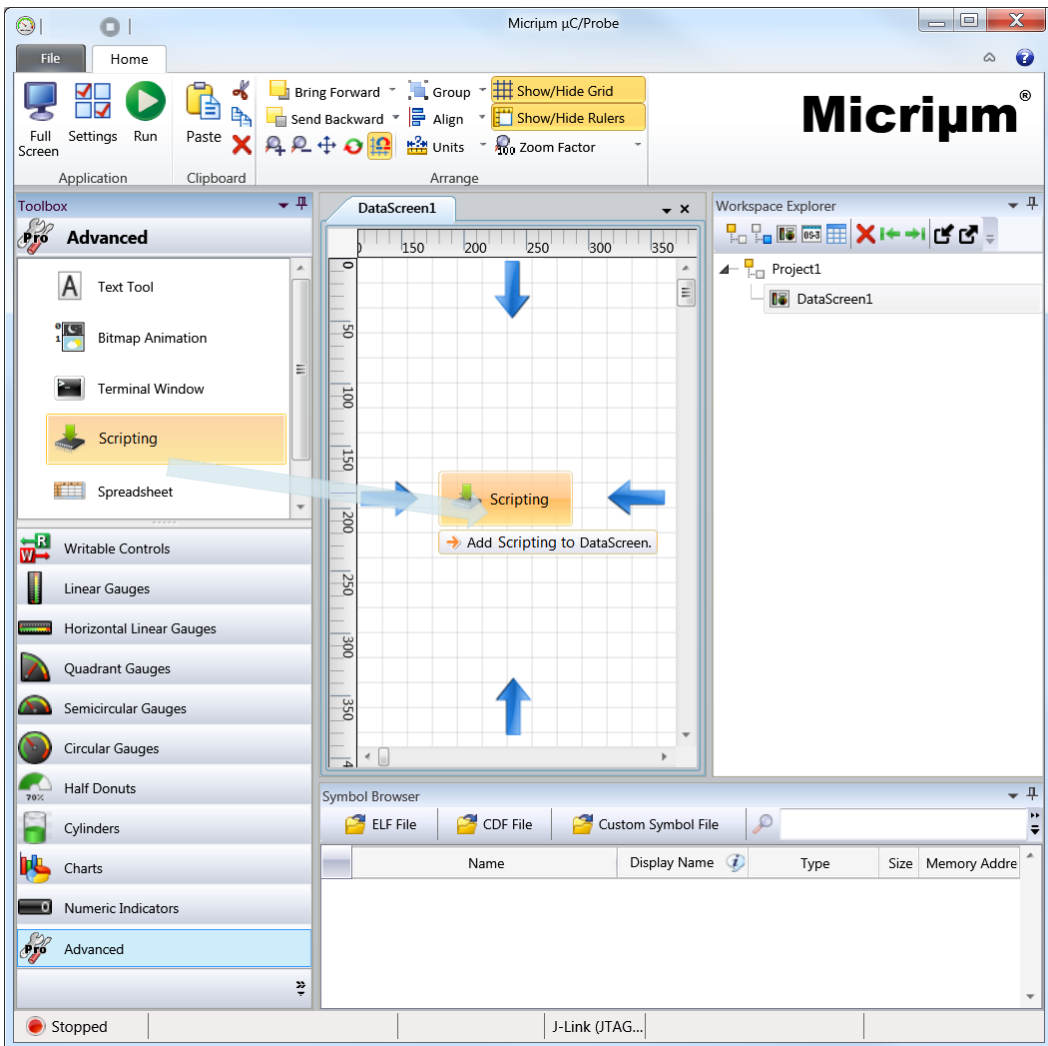


Figure F-1 Adding an Instance of the Scripting Control

---

### F-3 CONFIGURING THE SCRIPTING CONTROL

Once you have successfully added an instance of the Scripting Control into a data screen, you have to specify the path to the script file by making click on the properties editor button, which in turn opens a file dialog for you to specify the location of your script file as shown in Figure F-2:

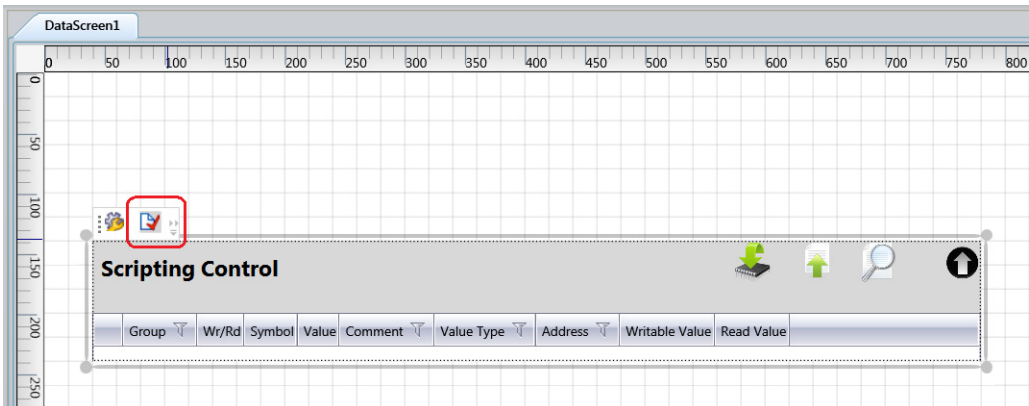


Figure F-2 Configuring the Scripting Control

---

## F-4 EXECUTING THE SCRIPT

Suppose you create a configuration script that not only sets the gain of each channel in your data acquisition system, but also activates a system self test and then evaluates the results.

Code Listing F-5 shows the script for such an example.

```

                                                                    /* Configure the system.          */
var max_ch = 3;
int gain   = 10;

for (int i = 0; i < max_ch; i++) {
    $AppChannels.Ch[i].Gain = gain;
}

$AppSelfTestStart = true;          /* Gets the system self test started.    */

                                    /* The Sleep instruction allows you to   */
                                    /* delay the script execution (milli-  */
                                    /* seconds) with a custom message.     */
Sleep("Self-Test in progress...", 5000);

                                    /* Evaluate the self test results.     */
if ($AppSelfTestErr == 0) {
                                       /* The Pause() instruction allows you  */
                                       /* to suspend execution until you     */
                                       /* press a button in  $\mu$ C/Probe.      */
    Pause("System Configured OK.");
} else {
    Pause("Unable to Configure the System.");
}

```

Listing F-5 Configuration Script Example

Once you configure the Scripting Control with the script in Listing F-5, the control will parse the code and will display each of the instructions the control will execute for you as shown in Figure F-3:

Group	Wr/Rd	Symbol	Value	Comment	Value Type	Address	Writable Value	Read Value
config1.txt	!	AppChannels.Ch[0].Gain	10		Numeric	N/A	N/A	N/A
config1.txt	!	AppChannels.Ch[1].Gain	10		Numeric	N/A	N/A	N/A
config1.txt	!	AppChannels.Ch[2].Gain	10		Numeric	N/A	N/A	N/A
config1.txt	!	AppSelfTestStart	True	/* Gets the system self test started. */	Boolean	N/A	N/A	N/A
config1.txt	!	Sleep		5000	Unknown	N/A	N/A	N/A
config1.txt	!	AppSelfTestErr			Unknown	N/A	N/A	N/A

Figure F-3 Configuration Script Example - Design Time

When you run  $\mu$ C/Probe, the control will execute your script when you click the button with the chip icon and will display the status of each instruction's execution as shown in Figure F-4:

Group	Wr/Rd	Symbol	Value	Comment	Value Type	Address	Writable Value	Read Value
config1.txt	!	AppChannels.Ch[0].Gain	10		Numeric	N/A	N/A	N/A
config1.txt	!	AppChannels.Ch[1].Gain	10		Numeric	N/A	N/A	N/A
config1.txt	!	AppChannels.Ch[2].Gain	10		Numeric	N/A	N/A	N/A
config1.txt	!	AppSelfTestStart	True	/* Gets the system self test started. */	Boolean	00015376	01	01
config1.txt	!	Sleep		0	Unknown	N/A	N/A	N/A
config1.txt	!	AppSelfTestErr			Unknown	00015377	N/A	00
config1.txt	!	Pause		System Configured OK.	Unknown	N/A	N/A	N/A

Figure F-4 Configuration Script Example - Run Time

## G

## Data Logging Control

$\mu$ C/Probe provides an option to log the values of any variable(s) in your symbols browser to a CSV file. The Data Log Control is part of the **Advanced** category of  $\mu$ C/Probe's toolbox.

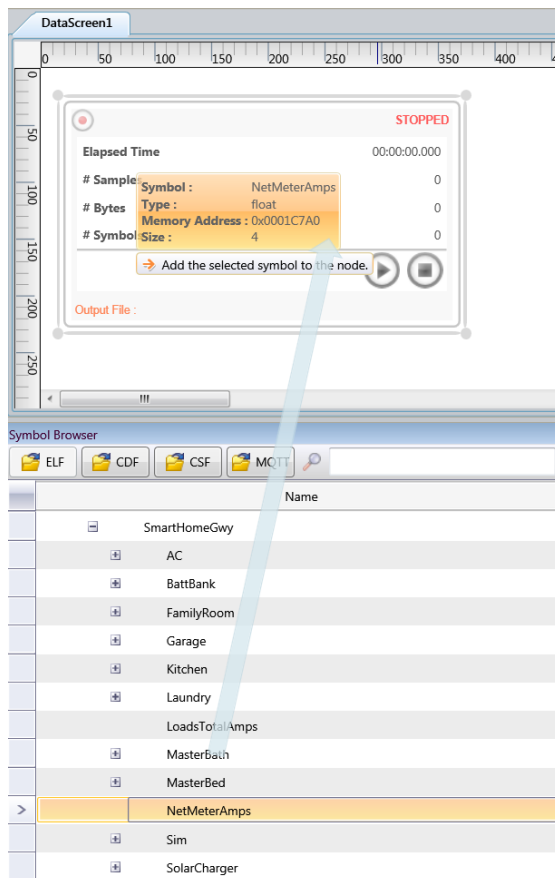


Figure G-1 Data Log Control

---

Figure G-1 illustrates how to log a variable called **NetMeterAmps**. All you have to do is drag and drop an instance of the Data Log Control from the Advanced category in your toolbox and then search the variable you want to data log in the Symbol Browser. You add the variable to the data log by dragging and dropping it onto the Data Log control. You can add as many variables as you want and they will be stored in a CSV file, one column per variable.

You can configure the Data Log Control from the Properties Editor as shown in Figure G-2:

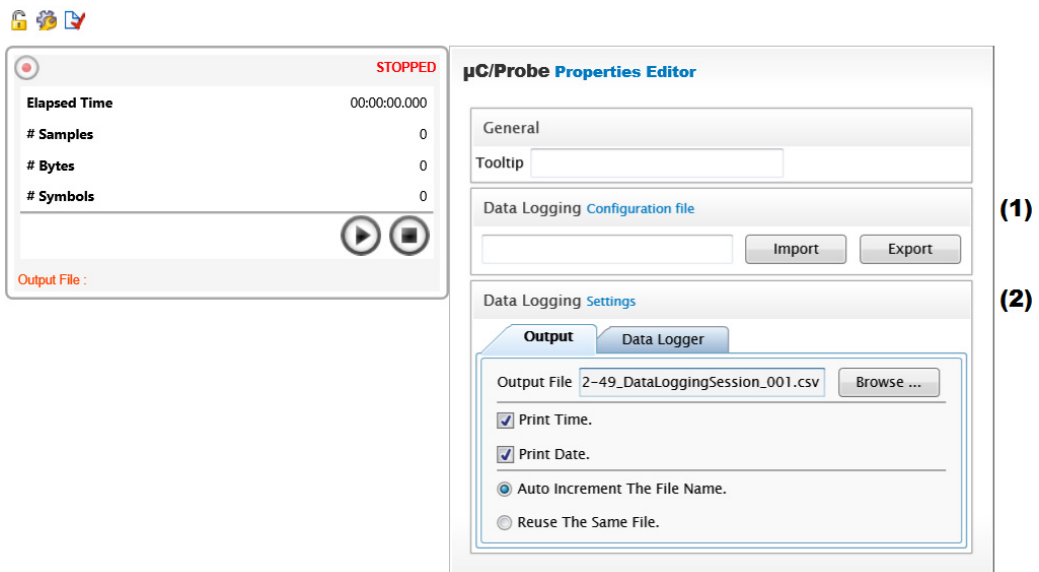


Figure G-2 **Data Log Control Properties Editor**

FG-2(1) You can either **Import** a Data Log Configuration file from a past logging session or you can **Export** your current configuration to a file.

FG-2(2) The Data Logging Settings panel consists of two tabs; The **Output** tab and the **Data Logger** tab. From the **Output** tab you can configure the output file. You can specify the path and name of the output file along with other settings such as the timestamps and whether or not reusing the same file or create a new one by using an auto-increment number as part of the file name. By default the output files are stored in the µC/Probe installation directory in the subfolder **DataLogging**.



During run-time, the value of NetMeterAmps will be logged in the CSV file as shown in Figure G-3:

	A	B	C	D	E
1	Date Time	NetMeterAmps			
2	2014-06-26 04:04:52:300	19.0423851			
3	2014-06-26 04:04:52:409	19.04474258			
4	2014-06-26 04:04:52:519	19.04695129			
5	2014-06-26 04:04:52:627	19.04544449			
6	2014-06-26 04:04:52:737	19.04297447			
7	2014-06-26 04:04:52:847	19.04819489			
8	2014-06-26 04:04:52:955	19.04792976			
9	2014-06-26 04:04:53:065	19.04913712			
10	2014-06-26 04:04:53:174	19.04587364			
11	2014-06-26 04:04:53:284	19.04684448			
12	2014-06-26 04:04:53:392	19.04320526			
13	2014-06-26 04:04:53:502	19.04488373			
14	2014-06-26 04:04:53:610	19.04523087			
15	2014-06-26 04:04:53:719	19.04252052			
16	2014-06-26 04:04:53:830	19.04655838			
17	2014-06-26 04:04:53:939	19.04323196			
18	2014-06-26 04:04:54:047	19.0476799			
19	2014-06-26 04:04:54:157	19.04779816			
20	2014-06-26 04:04:54:266	19.04637527			
21	2014-06-26 04:04:54:377	19.04769135			

Figure G-3 Data Log Output File (CSV format)

You can suspend or stop the data logging process by using the buttons in the status screen shown in Figure G-4

**RUNNING**

**Elapsed Time** 00:00:02.847

**# Samples** 26

**# Bytes** 1,040

**# Symbols** 1

Output File : 2014-06-26-16-22-49\_DataLoggingSession\_009.csv

Figure G-4 Data Log Control During Run-Time

---

Additionally, you can configure **Start** and **Stop** conditions to trigger the data logging process. For example, Figure G-5 illustrates how to configure the Data Logger to Start logging when the value of **NetMeterAmps** is between 20 and 50 amps.

The sampling period can also be configured from the same screen in terms of hours, minutes, seconds and milliseconds

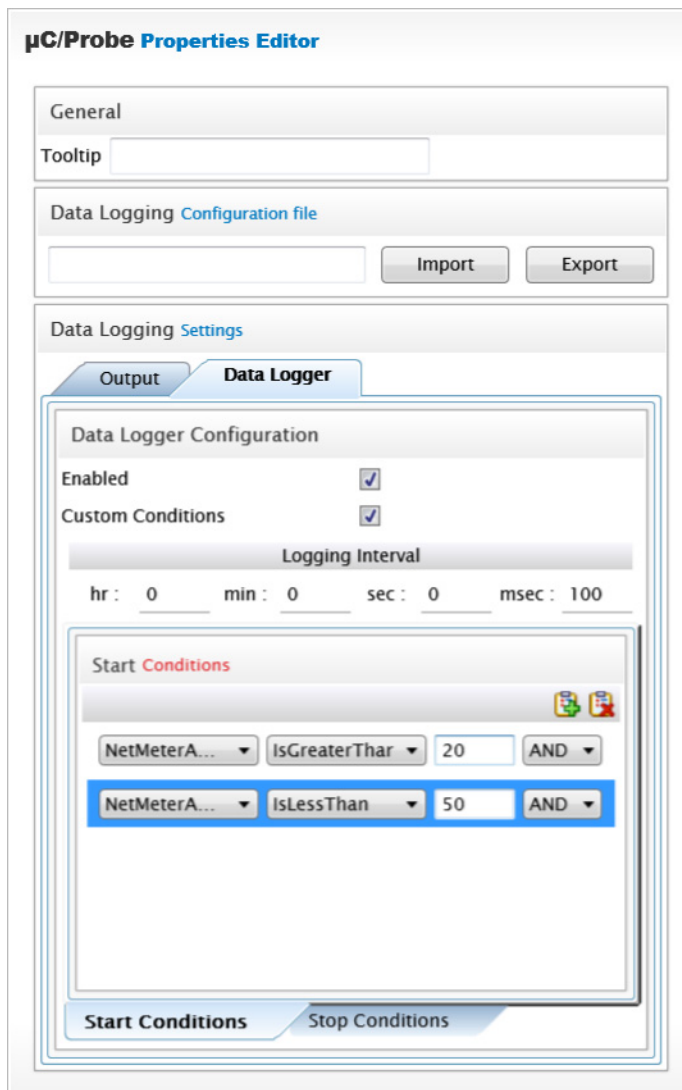


Figure G-5 Data Logger Start and Stop Conditions

## H

## Human Machine Interface (HID) Control

$\mu$ C/Probe allows you to use a USB HID control such as a gamepad, joystick or steering wheel to control your embedded target. The HID Control is part of the **Advanced** category of  $\mu$ C/Probe's toolbox.

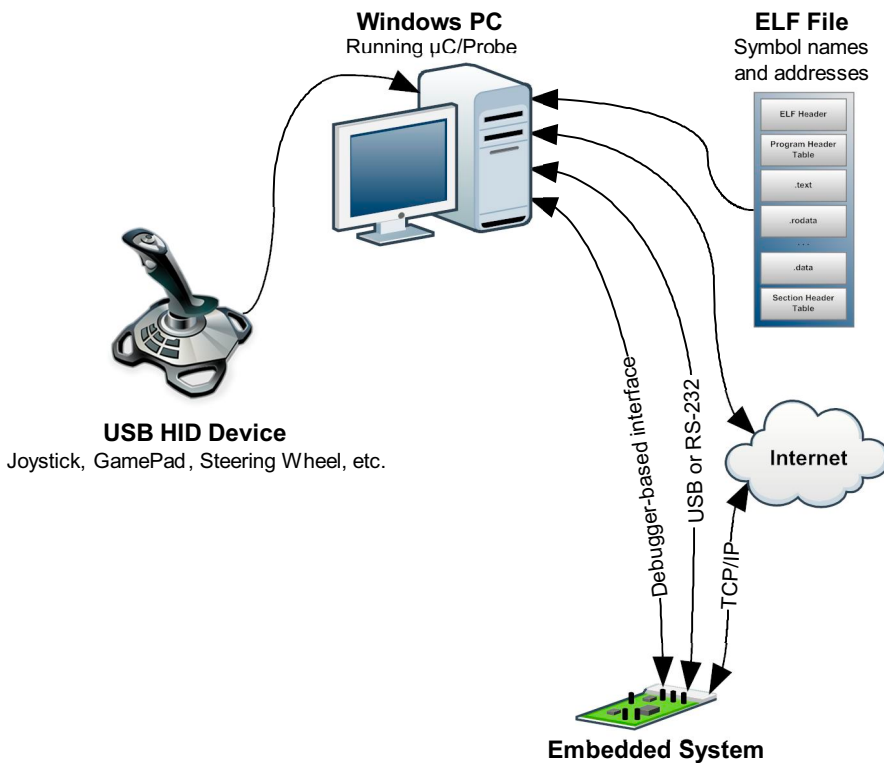


Figure H-1 HID Control Block Diagram

You simply declare in your embedded target the variables that will store the output from your HID device.

For example, if your HID device is a Joystick, then you would declare variables such as:

- AppJoystickButton1
- AppJoystickButton2
- AppJoystickButton3
- AppJoystickButtonX
- AppJoystickButtonY

Then you drag and drop an instance of the HID control onto your Data Screen and start associating the embedded variables to the control as shown in Figure H-2:

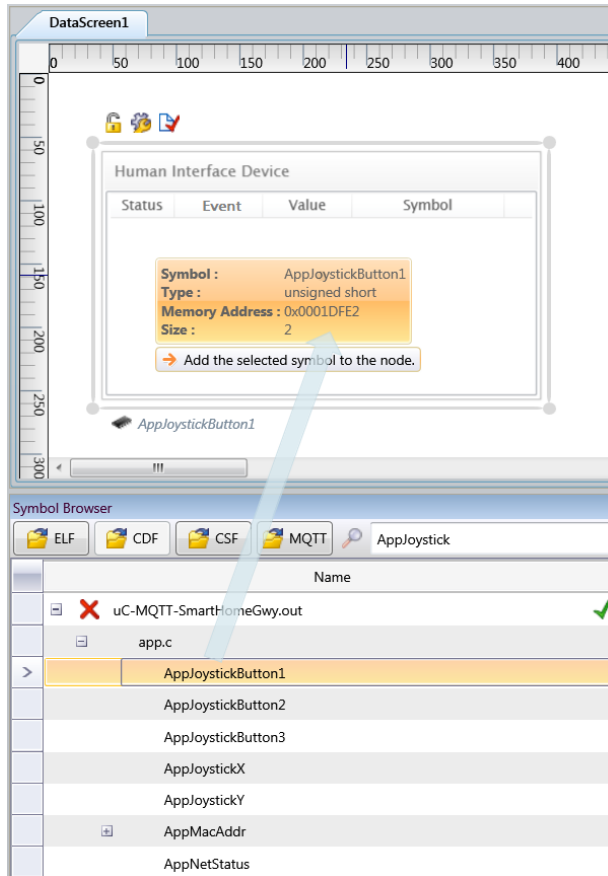


Figure H-2 Symbol Browser and HID Control

The next step is to configure each possible event from the HID device and map it to a unique embedded target symbol. You first open the HID Control's Properties Editor and select your HID device as shown in Figure H-3:

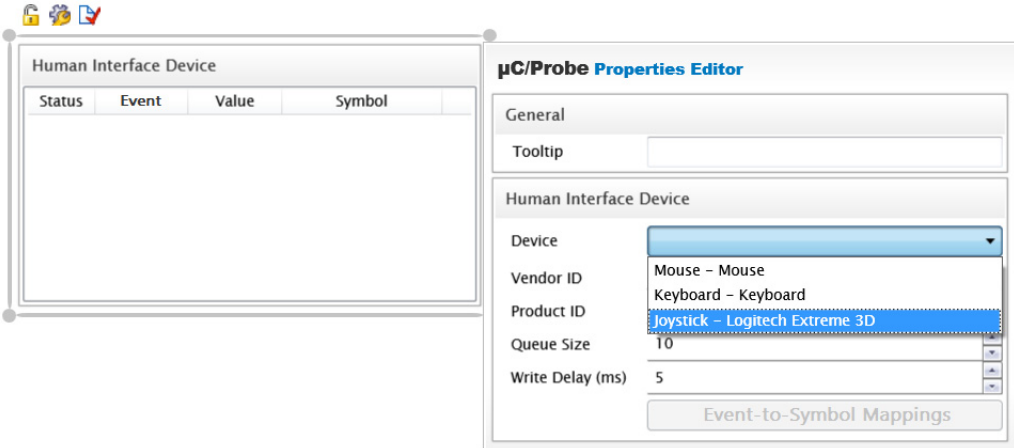


Figure H-3 HID Control Properties Editor

Once you select your HID Device, the button labeled as **Event-to-Symbol Mappings** will be enabled and you can proceed to map each event to an embedded target symbol. As you move the joystick, μC/Probe will catch all the possible events and create a new option to configure a mapping as shown in Figure H-4:

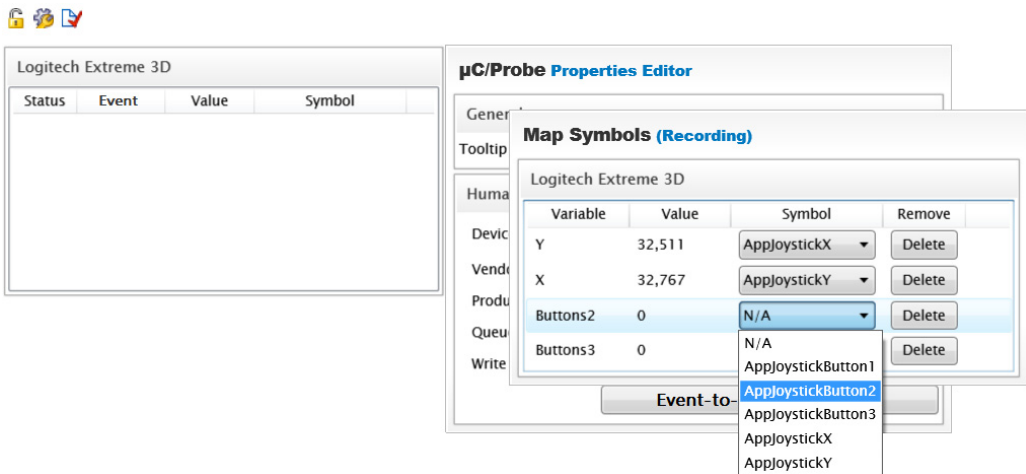


Figure H-4 HID Control Event-to-Symbol Mappings

---

During run-time  $\mu$ C/Probe displays the status of your HID Device with the table shown in Figure H-5. The LED remains green during idle time and turns red once the event is caught. At each event the value is written to the associated target symbol.

Status	Event	Value	Symbol
●	Y	32,511	AppJoystickX
●	X	32,511	AppJoystickY
●	Buttons2	0	AppJoystickButton2
●	Buttons3	128	AppJoystickButton3

Figure H-5 **HID Control Status**



## Licensing

The Educational Edition of  $\mu$ C/Probe is freely available for students, teachers, and academic organizations. You can purchase  $\mu$ C/Probe for your personally owned computer or those owned by your employer by either a permanent license or a subscription based license as follows:

- Monthly Subscription License

Our e-commerce partner FastSpring will bill you automatically each month and you may cancel the subscription at any point, no questions asked.

- Yearly Subscription License

Our e-commerce partner FastSpring will bill you automatically each year and you may cancel the subscription at any point, no questions asked.

- Permanent License

Our e-commerce partner FastSpring will bill you only once and you will own the license for an unlimited period of time.

*Note: Keep in mind that all  $\mu$ C/Probe sales are final and non-refundable. The Educational Edition of  $\mu$ C/Probe is available for free to enable you to “try before you buy”.*

*For example, if you purchase a yearly subscription and wish to cancel the subscription after a couple of months, we will cancel your subscription and any future billing. However, we will not be able to make any partial refunds and you would still be charged for the first year’s subscription.*

---

## **I-1 ORDERING**

If you decide to purchase a license of  $\mu$ C/Probe you can go to our online store at:

<http://micrium.com/tools/ucprobe/buy>



The table below summarizes the main differences among the different editions of  $\mu$ C/Probe:

<b>Feature</b>	<b>Educational Edition</b>	<b>Basic Edition</b>	<b>Professional Edition</b>
Design Mode	x	x	x
Run-Time Mode (timeout in minutes)	x (1)	x (no-timeout)	x (no-timeout)
Maximum Number of Data Screens	1	unlimited	unlimited
Maximum Number of Gauge Styles	5	unlimited	unlimited
Maximum Number of Numeric Indicator Styles	3	unlimited	unlimited
Thermometer	x	x	x
Cylinder Indicators	x	x	x
Button Controls	x	x	x
Slider Controls	x	x	x
Bit Control	x	x	x
Marker Chart	x	x	x
Line Chart	x	x	x
Area Chart	x	x	x
Scatter X-Y Chart			x
$\mu$ C/OS-III Kernel Awareness	x	x	x
Terminal Window Control			x
Scripting Control			x
Microsoft <sup>®</sup> Excel <sup>®</sup> Bridge			x
$\mu$ C/Trace Trigger Control			x
Import/Export Data Screens		x	x
Numeric Up/Down Control			x
Textbox Control			x
Data Log Control			x
HID Control			x

Table I-1  $\mu$ C/Probe Editions Comparison Table

---

## I-2 ACTIVATING

Once you place an order on our online store you will receive an e-mail message with your license key.

To activate your copy of  $\mu$ C/Probe, you need to have Internet access. Click on *File* -> *Activation* and a window similar to the one shown below will be displayed:

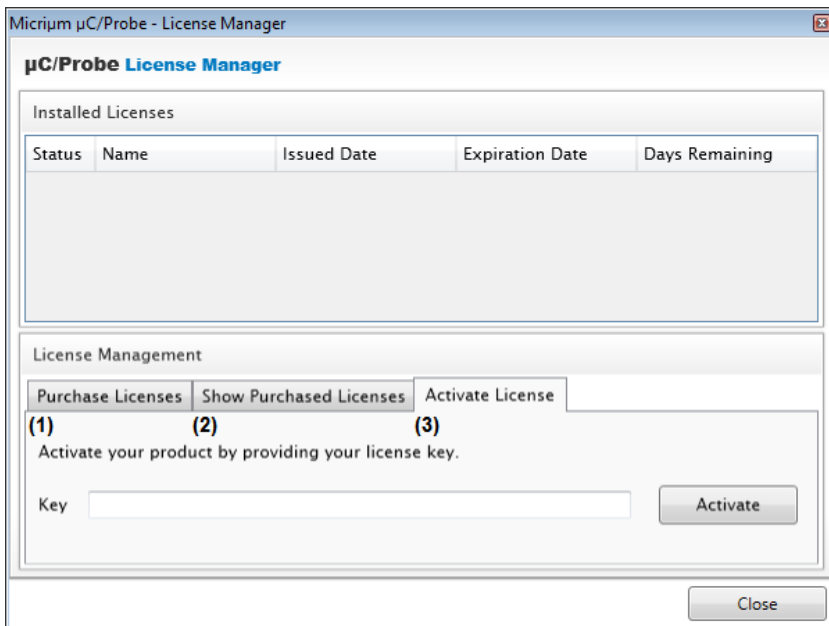


Figure I-1 License Manager

- FI-1(1) Use this tab to go online and purchase a license of  $\mu$ C/Probe.
- FI-1(2) Use this tab to obtain a list of all your purchased licenses. The list will be sent via e-mail to the e-mail address registered when you placed the order. All you need to do is provide the invoice number.
- FI-1(3) Use this tab to activate a license key and node-lock it to your computer. All you need to do is copy and paste the license key from the e-mail message you received and press the button *Activate*.

---

Please contact us for further information about pricing, ordering options, license activation or cancelling a subscription at:

Micrium  
1290 Weston Road, Suite 306  
Weston, FL 33326

+1 954 217 2036  
+1 954 217 2037 (FAX)

E-Mail : [sales@micrium.com](mailto:sales@micrium.com)  
Website : [www.micrium.com](http://www.micrium.com)

## Appendix

# J

## Bibliography

- Labrosse Jean. *μC/OS-II The Real-Time Kernel*. R&D Technical Books, ISBN 1-57820-103-9, 2002.
- Labrosse Jean. *μC/OS-III The Real-Time Kernel*. Micrium Press, ISBN 978-0-98223375-3-0, 2009.
- Légaré Christian. *μC/TCP-IP The Embedded Protocol Stack*. Micrium Press, 2011.

# Index

## A

animation .....	72
animation properties editor .....	72–73
area charts .....	87

## C

chart properties editor .....	88
chart series editor .....	90
charts .....	52, 87
charts properties editor .....	88
charts series editor .....	90
checkbox properties editor .....	77
circular gauges .....	51
communication settings .....	31, 37
J-Link .....	39
RS-232 .....	44
TCP/IP .....	42–43
configuration	
terminal window control .....	101
custom slider	
example .....	75
properties editor .....	75
custom switch	
properties editor .....	76
cylinders .....	52

## D

data flow .....	6
design time .....	11

## E

ELF file .....	16
browsing .....	16
example .....	57

## F

formatting	
properties editor .....	68

## G

general settings .....	30
------------------------	----

## H

half donuts .....	51
horizontal linear gauges .....	49

## I

indicators .....	58
------------------	----

## J

J-Link .....	38
--------------	----

## K

kernel awareness screen	
miscellaneous .....	97
task list .....	98

## L

layout design tools .....	55
line charts .....	87
linear gauges .....	49

## M

marker charts .....	87
miscellaneous (tools) .....	54

## N

numeric indicator	
properties editor .....	70
numerics (tools) .....	53

## O

ordering .....	107, 111, 124
overview .....	9

## P

properties editor	
animation .....	72–73
checkbox .....	77
custom slider .....	75

---

formatting .....	68
numeric indicator .....	70
push button .....	78
repeat button .....	80
slider control .....	74
terminal window control .....	102
toggle button .....	79
push button	
properties editor .....	78

## Q

quadrant gauges .....	50
-----------------------	----

## R

range and colors editor .....	69
repeat button	
properties editor .....	80
RS-232 .....	44
run-time mode .....	62
run-time mode checklist .....	61

## S

Segger J-Link .....	38
semicircle gauges .....	50
slider control	
properties editor .....	74
status bar .....	62
symbol browser	
loading an ELF file .....	15
symbols grouped by C file .....	16
symbols .....	58
symbols manager .....	59

## T

TCP/IP .....	42–43
terminal window control .....	99–101
configuration .....	101
properties editor .....	102
toggle button	
properties editor .....	79
toolbar settings .....	29
toolbox .....	47
charts .....	52
circular gauges .....	51
cylinders .....	52
half donuts .....	51
horizontal linear gauges .....	49
linear gauges .....	49
miscellaneous .....	54
numerics .....	53
quadrant gauges .....	50
semicircle gauges .....	50
writable controls .....	48
trace triggers control .....	103–104

## V

virtual controls .....	58, 67, 74
virtual indicators .....	67–68

## W

Windows application .....	13
workspace explorer .....	45–46
writable controls .....	48

## Z

µC/Probe data client .....	11
design time .....	11
µC/Trace triggers control .....	103–104