

Customizing the I/O Tool Kit

Abstract:

This document describes the methodology for an OEM to add “Custom Commands”, and “Custom Protocols” to the appropriate I/O Tool Kit menus. Custom commands can call external applications and make them seem like they are a part of the I/O Tool Kit. “Custom Protocols” may be added to the list of available protocols for select ports on an IPm station.

Note: The information in this file is applicable for I/O Tool Kit version 2.6 or later to support all of the features described in this document. The IPm must have Firmware version 1.4 or later to support the Custom Protocols.

OEM Custom Commands (Tools Menu):

The OEM command definitions are read from files that OEM's must produce, and place in the **<I/O Tool Kit>\programs** directory. The files must have a **.customize** file extension. Although each OEM file (<filename>.customize) may contain up to six commands, it is requested that each OEM only use a maximum of two commands to allow for the possibility of the user installing multiple third party products that use custom menu items. (“customize” files are read at run time, and after six custom menu items are read the rest will be ignored.)

Custom Command Capabilities / Requirements

1. Custom commands will be displayed on the Tools menu.
2. The custom command must call an executable file (.exe,.bat,...).
3. The text displayed on the tool menu may be independently defined and may contain a “&” character to specify the hot key for the menu item. If the “&” character is included, the following character will be underlined and will become the hot key for the menu item. If the menu already contains a hot key with the same character, the first menu item with that hot key will respond.

File Format:

The commands section lists the commands to be added (starting from 0).

[commands]

```
0=command, hint text, arguments to pass, initial directory
1=command, hint text, arguments to pass, initial directory
2=command, hint text, arguments to pass, initial directory
```

Each entry consists of a comma-delimited string defined as follows:

command: the application name (.exe)

hint text: the text to place in the menu, and tool tip for toolbar buttons

arguments to pass: arguments to pass to the application (See macro list below.)

initial directory: working directory to start the application in

Macro List:

These macros are available to pass details of the user's current selection(s) in the I/O Tool Kit. By using these macros you can pass information about a selected station to external applications such as the current file name, selected station name, ... You do not have to use any of these macros. You may also add your own "hard coded" arguments as well as using any of these macros:

<code>\$(CurDir)</code>	Working directory of sxttools.exe
<code>\$(TargetDir)</code>	Directory of the currently opened project file
<code>\$(CommParams)</code>	Currently selected communications parameters
<code>\$(IPAddr)</code>	Primary IP Address of the currently selected station
<code>\$(Location)</code>	Location of the currently selected station/module
<code>\$(ProjFileName)</code>	Name of currently opened project file (pump1.6pj)
<code>\$(ProjPathName)</code>	Path of currently opened project file (d:\projects\pump1.6pj)
<code>\$(SerNum)</code>	Serial number of currently selected station
<code>\$(StaName)</code>	Name of currently selected station
<code>\$(StaNum)</code>	Station number of currently selected station

sample.customize (sample file):

[commands]

0=D:\stuff\app1.exe,&Application1,-IP=\$(IPAddr),\$(TargetDir)

1=D:\stuff\app2.exe,A&application2,-sta=\$(StaNum) -min,

In the above sample file there are two commands defined for inclusion on the I/O Tool Kit's Tools menu. One for app1.exe, which gets the currently selected IP address passed to it. The second executes app2.exe and passes the currently selected station number along with a "hard coded" -min argument. There is no initial directory specified for app2.exe. The hot key for the first menu item would be 'A', and 'p' for the second menu item.

Customize IPm Port Protocols

The same ".customize" file may also contain custom protocols for IPm stations. Each port of an IPm station can have a different list of protocols added. Typically this feature is used to integrate the choice for a user to select I/O drivers you have created into the I/O Tool Kit configuration dialogs.

In the example that follows, "protocolA" and "protocolB" will be added to the drop down list of available protocols in the I/O Tool Kit "Configure Selected Port" window for authorized com ports. In this manner, a user can make communication choices, including selecting your driver in the standard IPm configuration Window. These choices may be read from the standard ports definition files at run time, as described below.

```
[protocols]
0=protocolA
1=protocolB
```

The [protocols] section lists the number of protocols defined, and the names of each protocol. Entries must begin from 0, and be sequential.

Customizing the I/O Tool Kit

09/16/04

Page 3 of 5

```
[protocol:protocolA]
ports=/dev/ttyS2,/dev/ttyS3
[protocol:protocolB]
ports=Port A,Port B
```

Each protocol has a section designated by “protocol:<name>”. The <name> parameter must be the name as specified in the [protocols] section. There must be one of these sections for each protocol listed in the [protocols] section. The “ports” entry within this section lists the ports that this protocol may be assigned to. The ports are listed by name. The port names may either be the internal (Linux) names (/dev/ttyS0,...), or the name as it appears in the I/O Tool Kit (Port A, Port B,...). Either type of port name will be recognized. A comma separates each port name.

The protocol name is loaded into the station’s port configuration file (/etc/stacfg/portS0.config,...) in the same location as the standard protocols. The station will treat custom protocols as “unassigned”, meaning that the baud rate, data bits, and other settings will not be set by the IPm firmware.

Firmware applications (drivers) must read the port configuration files to determine the protocol selection for each port. The port configuration files are text files located in /etc/stacfg/. The protocol selection can be found by reading the “protocol=” entry under the [settings] section. Port settings such as baud rate, parity, data bits, ... can also be found in the [settings] section of these files. See below for a sample port configuration file.

Port Configuration File Names:

Device	File Name
/dev/ttyS0 (Port B)	/etc/stacfg/portS0.config
/dev/ttyS1 (Port D)	/etc/stacfg/portS1.config
/dev/ttyS2 (Port A)	/etc/stacfg/portS2.config
/dev/ttyS3 (Port C)	/etc/stacfg/portS3.config

Sample port configuration file (/etc/stacfg/portS0.config):

```
[settings]
baud=9600
daniel=none
data_bits=8
flow_control=n
hardware_term=none
lag_time=2
lead_time=10
modbus_word_order=lsb
modem_init=
modem_init_enable=n
parity=n
passthru_port=none
protocol=native
stop_bits=1
```

SOLUTIONS FOR YOUR INDUSTRIAL NETWORKING CHALLENGES

Sixnet, LLC • 331 Ushers Road • Ballston Lake, NY 12019 • USA
1.518.877.5173 • Fax 1.518.877.8346 • www.sixnet.com

Key	Possible Values
baud	300,600,1200,2400,4800,9600,19200, 38400,57600,115200
data_bits	8,7
stop_bits	1,2
parity	n = None, e = Even, o = Odd, m = Mark, s = Space
flow_control	n = None, h = hardware, x = Xon/Xoff, half_duplex = half duplex modem (RTS throttle), full_duplex = full duplex modem (RTS signal)
protocol	native = Sixnet Universal Driver modbus_ascii_slave modbus_rtu_slave user = Unassigned protocol, configured port unassigned = Unconfigured by IPm modbus_ascii_master modbus_rtu_master (other) = OEM defined protocols listed by OEM specified name
daniel	none = not using Daniel extension lf = use Daniel extension for longs, and floats (cannot do longs without floats,...)
lead_time	0 - 65534 (ms)
lag_time	0 - 65534 (ms)
pass_thru_port	none = None /dev/ttyS0 = ttyS0 /dev/ttyS1 = ttyS1 /dev/ttyS2 = ttyS2 /dev/ttyS3 = ttyS3 /dev/eth0 = TCP/IP <i>Passthru is not available to OEM defined protocols via the Toolkit</i>
hardware_term	none = disable HW termination term_rcv = enable HW termination
modem_init_enable	y = use modem initialization n = do not use modem initialization
modem_init	200 bytes for ASCII string sent to serial port on station reset or power up.
modbus_word_order	lsb = least significant word first msb = most significant word first

Adding a Custom Configuration Button to the Port Configuration Window:

Note: This feature specifically requires I/O Tool Kit Software v2.6 or later.

To allow users to configure custom protocols from within the I/O Tool Kit's configuration windows, a button can be provided in the serial port configuration window. This button can be configured to call an external application similar to the way that custom commands are added to the tool menu. To enable the configuration button for a given protocol, add a line in the [protocol:protocolname] section beginning with the key "cmd" (see below for an example):

```
[protocol:myprotocol]
ports=/dev/ttyS2,/dev/ttyS3
cmd=C:\TimeConvert.exe,Convert Time,Sta=$(StaName) SerNum=$(SerNum),
```

The example above creates a button in the serial port configuration window with the text "Convert Time". This button will only be active when the protocol "myprotocol" is selected. When pressed, the button will execute "C:\TimeConvert.exe" and will pass the following arguments: Sta=<selected station name> SerNum=<selected station's serial number> port=<selected port name>)

The format of the "cmd" line is the same as the format of the custom commands described above. The "hint text" field (in this example, "Convert Time") is the text that will be placed on the button. Also, macros can be used for passing arguments as is done with the custom commands (in this example, macros \$(StaName) and \$(SerNum) are used). By default, the port argument, which contains the name of the port being configured (port=/dev/ttyS2), is always passed to the external application. The port names that are used here are the internal/Linux port names beginning with /dev/ttyS*.