



Distribution en France & Belgique

**NETWALKER**

[contact@netwalker.fr](mailto:contact@netwalker.fr)

0 177 628 628

[Plus d'infos ...](#)

# Developer Guide Intermapper 6.3



## Copyright Terms and Conditions

---

The content in this document is protected by the Copyright Laws of the United States of America and other countries worldwide. The unauthorized use and/or duplication of this material without express and written permission from HelpSystems is strictly prohibited. Excerpts and links may be used, provided that full and clear credit is given to HelpSystems with appropriate and specific direction to the original content. HelpSystems and its trademarks are properties of the HelpSystems group of companies. All other marks are property of their respective owners.

201902051033

# Table of Contents

|  |           |
|--|-----------|
| <b>Creating Your Own Probes</b> .....          | <b>13</b> |
| What is a probe? .....                         | 13        |
| Parts of a Probe .....                         | 13        |
| Types of Probes .....                          | 13        |
| <b>Anatomy of a Probe</b> .....                | <b>15</b> |
| Sections common to all probes .....            | 15        |
| Type-Specific Probe Sections .....             | 15        |
| Sections specific to SNMP Probes .....         | 15        |
| Sections specific to SNMP Trap Probes .....    | 16        |
| Sections Specific to TCP Probes .....          | 16        |
| Sections Specific to Command-line Probes ..... | 16        |
| Other Information about Probes .....           | 16        |
| The <header> Section .....                     | 17        |
| Header Parts .....                             | 18        |
| Sample Header Section .....                    | 20        |
| Header Section of Custom SNMP Probes .....     | 21        |
| Flags for Command-line Probes .....            | 21        |
| Probe File Locations .....                     | 22        |
| Probe File Names .....                         | 22        |
| The <description> Section .....                | 22        |
| The <parameters> Section .....                 | 23        |
| Input Field Types .....                        | 24        |
| Text Fields .....                              | 24        |
| Password Fields .....                          | 24        |
| Dropdown Fields .....                          | 24        |
| Check Box Fields .....                         | 25        |
| Parameter Section Example .....                | 25        |
| The <datasets> section .....                   | 26        |

|   |           |
|---|-----------|
| Auto-recording values .....   | 27        |
| Units of Measure .....  | 27        |
| The <autorecord> section (deprecated) .....                                       | 28        |
| Automatically-Recorded Data Values .....  | 29        |
| <b>Probe Status Windows .....</b>   | <b>41</b> |
| Controlling the Status window in SNMP Probes with <snmp-device-display> .....     | 41        |
| Controlling the Status window in TCP Probes with <script-output> .....            | 41        |
| Controlling the Status window in Command-Line Probes with <command-display> ..... | 42        |
| <b>IMML - Intermapper Markup Language .....</b>                                   | <b>43</b> |
| How Markup Tags Are Applied .....   | 43        |
| Markup Tag Summary .....  | 43        |
| Examples .....  | 44        |
| Creating a link .....   | 44        |
| <b>Probe Comments .....</b>   | <b>45</b> |
| One-line Comments .....   | 45        |
| <b>Built-in Probe Variables and Macros .....</b>                                  | <b>46</b> |
| Command Line Probe Variables .....  | 46        |
| The <command-line> and <command-exit> sections .....                              | 46        |
| The <command-display> section .....   | 47        |
| SNMP Probe variables .....  | 47        |
| The <snmp-device-display> section .....   | 48        |
| The <snmp-device-properties> section .....  | 48        |
| The OID column of the <snmp-device-variables> section .....                       | 49        |
| TCP Probe Variables .....   | 49        |
| The <script> section .....  | 49        |
| The <script-output> section .....   | 51        |
| Variables passed to Command-line Notifiers .....                                  | 51        |
| Macros .....  | 51        |
| The \${chartable} macro .....   | 51        |
| The \${eval} macro .....  | 53        |

|   |           |
|---|-----------|
| The \${variablename:legend} macro .....                         | 54        |
| The \${scalable10} and \${scalable2} macros .....               | 54        |
| The \${^stdout} variable and the Reason string .....            | 55        |
| <b>Using Persistent Variables .....</b>                         | <b>56</b> |
| SNMP Probe Example .....  | 56        |
| Command Line Probe Example .....                                | 58        |
| <b>SNMP Probes .....</b>  | <b>62</b> |
| Common Sections of an SNMP probe .....                          | 62        |
| Sections Specific to SNMP Probes .....                          | 62        |
| <b>The &lt;snmp-device-variables&gt; Section .....</b>          | <b>64</b> |
| Sample <snmp-device-variables> Section .....                    | 64        |
| Status Window Text - The <snmp-device-display> Section .....    | 64        |
| SNMP Scalar and Table Values .....                              | 67        |
| Using Variables in OIDs and Legends .....                       | 68        |
| Calculation Variables .....                                     | 68        |
| Built-in Variables .....  | 68        |
| Macros .....  | 68        |
| Enumerated Values .....   | 68        |
| Alternatives to Enumerated Values .....                         | 69        |
| <b>The &lt;snmp-device-thresholds&gt; Section .....</b>         | <b>71</b> |
| Sample <snmp-device-threshold> Section .....                    | 71        |
| Creating Comparisons .....                                      | 71        |
| Numeric Comparisons .....                                       | 72        |
| String Matches .....  | 72        |
| <b>The &lt;snmp-device-properties&gt; Section .....</b>         | <b>73</b> |
| <b>The &lt;snmp-device-variables-ondemand&gt; Section .....</b> | <b>74</b> |
| Background on SNMP Tables .....                                 | 74        |
| Table Indexes .....   | 75        |
| Table Syntax .....  | 75        |
| Augmenting Tables .....   | 76        |

|   |           |
|---|-----------|
| Index-Derived Variables .....   | 77        |
| Calculations within On-demand Tables .....                                    | 78        |
| Displaying On-demand Tables .....   | 79        |
| Limitations .....   | 79        |
| <b>The &lt;snmp-device-display&gt; Section .....</b>                          | <b>81</b> |
| Controlling the Status window in SNMP Probes with <snmp-device-display> ..... | 81        |
| Using Disclosure Widgets .....  | 81        |
| <b>The &lt;snmp-device-alarmpoints&gt; Section .....</b>                      | <b>84</b> |
| Alarm Points - What the User Sees .....                                       | 85        |
| Acknowledgments .....   | 85        |
| Notifications .....   | 86        |
| Log File Messages .....   | 86        |
| Configuring Alarm Points .....  | 86        |
| Alarm Point File Format .....   | 87        |
| Macros .....  | 88        |
| AlarmPoint Facilities .....   | 88        |
| Underscore Feature .....  | 88        |
| State Transitions .....   | 89        |
| Resetting to Neutral Alarm State .....  | 89        |
| Facilities to speed up rule evaluation .....                                  | 90        |
| Sample probe .....  | 90        |
| Alarm Point Notifier List .....   | 92        |
| <b>Probe Calculations .....</b>   | <b>94</b> |
| Expression Syntax .....   | 94        |
| Reserved keywords .....   | 94        |
| Precedence Table (Least to Most) .....  | 94        |
| Built-in Numeric Functions .....  | 95        |
| Built-in String Functions .....   | 95        |
| Function Descriptions .....   | 96        |
| defined .....   | 96        |

|  |            |
|--|------------|
| round .....  | 96         |
| strfind .....  | 97         |
| strifind .....   | 97         |
| strlen .....   | 97         |
| sprintf .....  | 97         |
| strftime .....   | 99         |
| strptime .....   | 100        |
| subid .....  | 101        |
| substr .....   | 101        |
| unpack .....   | 102        |
| Using Regular Expressions In Custom SNMP Probes .....            | 103        |
| <b>Specifying SNMP OIDs in Custom Probes .....</b>               | <b>105</b> |
| Introduction .....   | 105        |
| Numeric OID's .....  | 105        |
| Symbolic OID's .....   | 105        |
| OIDs indexed by Strings .....                                    | 106        |
| Limitations of Symbolic OID's .....                              | 107        |
| Get-Next OID Expressions .....                                   | 107        |
| Trap-Conditional OID Expressions .....                           | 107        |
| Index-Derived OID Expressions .....                              | 108        |
| <b>SNMP Probe Example .....</b>                                  | <b>109</b> |
| <b>SNMP Trap Probes .....</b>                                    | <b>112</b> |
| <b>The &lt;snmp-device-variables&gt; Section For Traps .....</b> | <b>113</b> |
| Packet Trap Variables .....                                      | 113        |
| Positional Variables from the Varbind List .....                 | 114        |
| Named Trap Variables .....                                       | 114        |
| Accessing Trap Variables by Position .....                       | 115        |
| <b>The &lt;snmp-device-display&gt; Section for Traps .....</b>   | <b>117</b> |
| <b>Trap Viewing and Logging .....</b>                            | <b>118</b> |
| <b>Example - Trap Viewer Probe .....</b>                         | <b>119</b> |

|   |            |
|---|------------|
| <b>The Dartware MIB .....</b>                       | <b>126</b> |
| <b>TCP Probes .....</b>                             | <b>130</b> |
| Common Sections of TCP Probes .....                 | 130        |
| Sections Specific to TCP Probes .....               | 130        |
| The Overall Process .....                           | 131        |
| Regular Expressions .....                           | 131        |
| Calculations in Scripts .....                       | 131        |
| Comparisons in Scripts .....                        | 132        |
| Comparing Numeric Values .....                      | 132        |
| Comparing String Values .....                       | 132        |
| Simple Comparisons in Scripts .....                 | 132        |
| <b>The &lt;script&gt; Section .....</b>             | <b>134</b> |
| TCP Probe Scripting Language .....                  | 134        |
| Script Process Flow .....                           | 134        |
| Script Command Format .....                         | 134        |
| String Argument Format .....                        | 135        |
| Special Characters .....                            | 135        |
| String Matching .....                               | 136        |
| Controlling Case-Sensitivity .....                  | 136        |
| Wild-card Character Matching .....                  | 136        |
| Numeric Argument Format .....                       | 137        |
| Using Numeric Arguments with the GOTO Command ..... | 137        |
| Default Values and Script Termination .....         | 138        |
| Using Labels for Program Control .....              | 138        |
| Jumping to a Label .....                            | 138        |
| Using Relative Offsets to Transfer Control .....    | 138        |
| Using Variables .....                               | 138        |
| Built-in Macros .....                               | 139        |
| Handling Script Failures .....                      | 139        |
| Adding Comments to your Script .....                | 140        |



|   |            |
|---|------------|
| Adding text within a command line .....                         | 140        |
| Adding text in Comment format .....                             | 140        |
| <b>The &lt;script-output&gt; Section .....</b>                  | <b>141</b> |
| <b>Measuring TCP Response Times .....</b>                       | <b>142</b> |
| Time Measurement Probe Variables .....                          | 142        |
| TCP Script Commands .....                                       | 142        |
| The <script-output> Section .....                               | 142        |
| A Note on Accuracy .....  | 142        |
| <b>Example TCP Probe File .....</b>                             | <b>144</b> |
| <b>Command Line Probes .....</b>                                | <b>147</b> |
| Common Sections of a Command-Line Probe .....                   | 147        |
| Sections Specific to Command-line Probes .....                  | 147        |
| The <tool> section - embedding a companion script .....         | 148        |
| Command Line Script API .....                                   | 148        |
| Installing a Command-line Probe .....                           | 149        |
| Passing parameters to a command-line probe .....                | 149        |
| Sending Data to STDIN .....                                     | 150        |
| <b>The &lt;command-line&gt; Section .....</b>                   | <b>151</b> |
| The Properties of the <command-line> Section .....              | 151        |
| Sending Data to STDIN .....                                     | 152        |
| <b>The &lt;command-exit&gt; Section .....</b>                   | <b>153</b> |
| <b>The &lt;command-display&gt; Section .....</b>                | <b>154</b> |
| <b>The &lt;tool&gt; Section .....</b>                           | <b>155</b> |
| What happens when you load a probe with a <tool> section? ..... | 155        |
| Calling external scripts and other executables .....            | 156        |
| Python Example .....  | 156        |
| Cscript Example .....   | 158        |
| <b>Command Line Probe Example .....</b>                         | <b>161</b> |
| <b>Intermapper Python Plugins .....</b>                         | <b>163</b> |
| Simple example .....  | 163        |

|   |            |
|---|------------|
| <b>Nagios Plugins</b>   | <b>166</b> |
| Example Return String   | 167        |
| Changes for Intermapper 5.0   | 168        |
| <b>Nagios Plugin Example</b>  | <b>169</b> |
| <b>NOAA Weather Probe Example</b>   | <b>171</b> |
| The NOAA Temperature Probe  | 172        |
| <b>PowerShell_Probe</b>   | <b>175</b> |
| <b>PowerShell Probe Examples</b>  | <b>176</b> |
| Example 1: Installed Software Probe   | 176        |
| Example 2: Windows Disk Space Probe   | 179        |
| <b>Troubleshooting PowerShell Probes</b>  | <b>186</b> |
| <b>Troubleshooting Probes</b>   | <b>188</b> |
| <b>Errors with Custom Probes</b>  | <b>189</b> |
| "Undefined variable" in Debug log   | 189        |
| A device shows a "Reason: No SNMP Response." at the bottom of the status window.        | 189        |
| When I build a custom probe, the status window shows "[N/A]" for certain values.        | 190        |
| When I build a custom probe, the status window shows "[noSuchName]" for certain values. | 190        |
| <b>Debugging with the SNMPWalk Command</b>  | <b>191</b> |
| Examples  | 192        |
| Invoking the snmpwalk command   | 192        |
| snmpwalk stopall command  | 193        |
| Using the SNMPWALK -o Option  | 194        |
| The SNMPWALK Schema   | 195        |
| Table: walks  | 195        |
| Table: results  | 196        |
| Accessing the SQLite Data   | 197        |
| <b>Reference</b>  | <b>198</b> |
| <b>Intermapper HTTP API</b>   | <b>199</b> |
| <b>Importing &amp; Exporting Files</b>  | <b>200</b> |
| HTTP file imports   | 201        |

|   |            |
|---|------------|
| Maps .....  | 201        |
| <b>Importing &amp; Exporting Tables .....</b>         | <b>202</b> |
| Table-based Import/Export .....                       | 202        |
| Importing Table-based Data .....                      | 203        |
| <b>Acknowledging with HTTP .....</b>                  | <b>204</b> |
| <b>HTTP API Scripting Examples .....</b>              | <b>206</b> |
| Unix shell script .....                               | 207        |
| Windows vbscript .....                                | 207        |
| Known bugs .....                                      | 208        |
| clone_im.sh .....                                     | 208        |
| clone_im.vbs .....                                    | 210        |
| <b>Retrieving Collected Data .....</b>                | <b>215</b> |
| Intermapper Database Schemas .....                    | 215        |
| <b>Creating SQL Queries .....</b>                     | <b>216</b> |
| Using the load_data() function .....                  | 216        |
| <b>Customizing Web Pages .....</b>                    | <b>218</b> |
| Reloading Changed Web Page Files .....                | 218        |
| <b>Target Files .....</b>                             | <b>219</b> |
| Target File Example: .....                            | 219        |
| Quoted Links .....                                    | 219        |
| What Happens When a Target File Is Read? .....        | 220        |
| Built-in Target Files .....                           | 220        |
| <b>Template Files .....</b>                           | <b>221</b> |
| Template File Example .....                           | 221        |
| <b>Directives .....</b>                               | <b>222</b> |
| Summary of Directives .....                           | 222        |
| <b>Quoted Links .....</b>                             | <b>226</b> |
| Preventing a Quoted String From Becoming a Link ..... | 226        |
| <b>Macro Reference .....</b>                          | <b>227</b> |
| The Include Macro .....                               | 227        |

|  |            |
|--|------------|
| Macros that generate the "content" of an InterMapper web page .....      | 227        |
| Miscellaneous macros that describe InterMapper and its environment ..... | 229        |
| Macros to place images onto a page .....                                 | 231        |
| Macros that control the interval between page refreshes .....            | 233        |
| Macros related to links and URLs .....                                   | 233        |
| <b>Web Pages Folder .....</b>  | <b>237</b> |
| Overriding the Built-in Pages .....                                      | 237        |
| Contents of BuiltinWebPages.zip .....                                    | 237        |
| How the Web Page Files are Used .....                                    | 238        |
| <b>MIME Types .....</b>  | <b>239</b> |
| <b>Tip for Calling Charts .....</b>                                      | <b>240</b> |
| <b>Command-line Options for InterMapper .....</b>                        | <b>241</b> |
| <b>Index .....</b>   | <b>242</b> |

# Creating Your Own Probes

For many Internet services, simply "pinging" a device is not a sufficient test of whether it is operating correctly. Intermapper has a number of built-in probes that can test different aspects of a device's operation, whether it's a web server, router, database, LDAP server etc.

However, Intermapper's built-in probes may not test the kinds of devices you want to monitor, or may not test them ways that are most useful to you. In such a case, you can create your own probes. Intermapper's probes are defined by *probe files*, simple text files that can be duplicated and modified using any standard text editing utility. When you create your own probe, it becomes a first-class citizen and appears in the Set Probe window along with the built-in probes.

## What is a probe?

A probe is a text file that specifies the way in which Intermapper tests a device. It is essentially a plug-in. All of Intermapper's probes follow the same logic:

- The probe sends one or more queries, as SNMP requests or UDP datagrams or over a TCP connection to the device being tested.
- The device responds (or fails to respond).
- If there is no response, Intermapper sets the device's status to DOWN.
- Intermapper examines the response(s) from the device, and sets the device's status accordingly.

## Parts of a Probe

All of the probe types listed below follow a similar structure. This is outlined in [Anatomy of A Probe](#), which explains the common sections of a probe, and the sections that are specific to a particular probe type.

## Types of Probes

Intermapper has several kinds of probes. You can use Intermapper's built-in probes as-is, you can copy and modify them, or you can develop your own probes from scratch. These are the different probe types:

- [SNMP Probes](#) - Intermapper tests the device's status by sending SNMP queries and comparing the results to user-specified thresholds.

- [SNMP Trap Probes](#) - Intermapper can receive and process SNMP traps, and can set the status of a probe based on the contents of the trap's variables. You can create custom probes that alert you to problems in a certain device based on the contents of specific trap variables.
- [TCP Probes](#) - Intermapper establishes a TCP connection to a device. It then sends certain requests and evaluates the responses to determine the device's status. TCP probes uses the [TCP Probe Scripting language](#) to create a sequence of commands, branching to different parts of the script under specified conditions.
- [Command-line Probes](#) - Intermapper can invoke a program or script (as if "from the command line"), and use the results to determine the device's status.
- **Big Brother Probes** - Big Brother™ is an open-source network monitoring program. Intermapper listens for reports from Big Brother clients and sets the device's status accordingly.

It's fairly straightforward to modify the existing files to produce new probes. If you make a new probe file that might be useful, please consider sending it to us. See [Sharing Probes](#) for more information.

# Anatomy of a Probe

Probe files have several sections, several of which are common to all probe types. These are described below.

Each section contains a number of lines bracketed by

```
<section-name> ... </section-name>
```

You might want to open a separate window with the [example probe file](#) while reading the subsequent sections.

## Sections common to all probes

### [The <header> Section](#)

Learn about the <header> section of the probe definition, including how the probe is identified, how its name appears in the **Probe Type** menu, and the version numbering system.

### [The <description> Section](#)

The <description> section specifies the help text that appears in the Set Probe window and typically explains the function of the probe and the use of its parameters. Format the description using IMML, [InterMapper's Markup language](#).

### [The <parameters> Section](#)

The <parameters> section defines the probe's parameters and how they are presented in the Set Probe window.

### Display Sections

Each probe type has its own output section, which controls what appears in the device's Status window. In all probes, you format the appearance of Status window using IMML, [InterMapper's Markup language](#).

## Type-Specific Probe Sections

Each probe type has sections that are specific to that probe type.

## Sections specific to SNMP Probes

Each custom SNMP probe has:

- [An <snmp-device-variables> section](#) - Specifies which MIB variables to collect from the device.

- [An <snmp-device-thresholds> section](#) - Specifies how those variables are to be tested against thresholds to determine the device's status.
- [An <snmp-device-display> section](#) - Specifies what information about the device and its links should be displayed in the Status window.
- [The <snmp-device-properties> section](#) - Specifies certain aspects of the SNMP queries sent to the device.
- [The <snmp-device-alarmpoints> Section](#) - Allows you to define conditions under which the device changes a particular device state.

## Sections specific to SNMP Trap Probes

SNMP Trap Probes do not probe devices - they simply wait for traps to arrive. They have some sections that are common to SNMP Probes, but work somewhat differently.

- [An <snmp-device-variables> section](#) - Specifies which MIB variables to collect from the device. These are set automatically when a trap is received.
- [An <snmp-device-thresholds> section](#) - Specifies how those variables are to be tested against thresholds to determine the device's status.

## Sections Specific to TCP Probes

Each custom TCP probe can have:

- [A <script> section](#) , which uses a sequence of commands and program flow that is similar to Basic. A rich set of [TCP commands](#) is available.
- [A <script-output> section](#).

## Sections Specific to Command-line Probes

Each command-line probe also has:

- [The <command-line> section](#) - builds the command-line, specifying the command path and any paramters to be sent with the command.
- [The <command-exit> section](#) - allows you to control the state of the device, depending on what is returned from the script.
- [The <tool> section](#) - contains the code for a companion script that is executed by the probe.
- [The <command-display> section](#) - allows you to control what appears in the device's Status window.

## Other Information about Probes

[Comments](#)



All probes use the same format for comments. The comment format is similar to HTML comments.

[Probe File Locations](#) and [Probe File Names](#)

To use probe files, you must import them, and should follow recommended naming conventions.

[Installing and Reloading Probes](#)

Before a modification to a probe becomes effective, you need to click the ***Reload probes...*** button in the Set Probe window (circular arrow icon below the left pane of the window).

## The <header> Section

The <header> section of a probe file contains a formal description of the probe. It is defined as follows, with each header property having a name and a corresponding value.

```
<header>
..[part name] = "[value]"
</header>
```

**Note:** Information by which Intermapper uniquely identifies the probe is contained in the header. While it is not required, HelpSystems strongly recommends that you follow [probe file naming conventions](#) that correspond to the unique identifier in the probe header.

## Header Parts

### type

Describes the type of the probe file. Intermapper supports the following probe types:

- **builtin**
- **tcp-script**
- **custom-snmp**
- **custom-snmp-trap**
- **command-line**
- **cmd-line**

```
type = "cmd-line"
```

For custom SNMP probes, use the *custom-snmp* type.

For custom SNMP Trap probes, use the *custom-snmp-trap* type.

For custom TCP probes, use the *tcp-script* type.

For command-line probes, use the *command-line* or *cmd-line* type.

|                     |  |
|---------------------|--|
| <b>package</b>      | <p>The first part of the probe's full identifier. Typically, the package is made up of the domain name of the organization that created the probe, with the labels reversed.</p> <p>For example, all probes created by HelpSystems, LLC the package statement is as follows:</p> <pre>package = "com.dartware"</pre> <p>The package part guarantees that different organizations can create probes without concern that their probe identifiers will conflict.</p> <p><b>Note:</b> The combination of [package].[probe_name] together form the probe's full identifier. (In the example below, the full identifier is "com.dartware.tcp.custom") By convention, the name of the file that holds the probe definition is the same as the probe's full identifier. This is not required, but it's a good idea. See <a href="#">Probe File Locations</a> and <a href="#">Probe File Names</a> below for more information.</p> |
| <b>probe_name</b>   | <p>The second part of the probe's full identifier. The probe_name may be whatever string the creating organization chooses.</p>  |
| <b>human_name</b>   | <p>The string that appears in the left pane of the Set Probe window. This string helps guides the user to select the probe for a particular device.</p>  |
| <b>version</b>      | <p>Provides a means to determine which probe file is the current one. The format of the version is "#.#".</p>  |
| <b>address_type</b> | <p>A comma-separated list of one or more address types. Intermapper implements "IP" and "AT".</p>  |
| <b>port_number</b>  | <p>The IP port used by this probe.</p>   |

**display\_name**

Specify the display\_name to use with this probe, using forward slashes to specify the hierarchy. To do this, add the following line to the <header> section of the probe:

```
display_name = "[top level]/[next level]/[next level]"
```

Example:

```
display_name = "Custom/Command-  
line"
```

**url\_hint**

Assign a double-click action within the probe (making it the pre-defined double-click action). To do this, add the following line to your <header> section of the probe:

```
url_hint = "url-to-invoke"
```

The following example would invoke the web browser to the device's IP address and port

```
url_hint =  
"http://${address}:${port}"
```

**poll\_interval**

Set the device's default poll interval to the indicated number of seconds. This overrides the map's default setting, and might be used to avoid too-frequent polling for (physical) devices that should not be polled too often.

Setting the poll interval *\*for the device\** will override this poll\_interval setting.

```
poll_interval = "300"
```

## Sample Header Section

This is a sample header from the Custom TCP script.

```
<header>  
  "type"           = "tcp-script"  
  "package"        = "com.dartware"  
  "probe_name"     = "snmp.example"  
  "human_name"     = "Example SNMP probe"  
  "display_name"   = "Miscellaneous/Example SNMP Probe"  
  "version"        = "1.0"
```

```
"address_type" = "IP,AT"  
"port_number" = "161"  
"flags" = ""  
</header>
```

## Header Section of Custom SNMP Probes

The `<header>` section of the probe file is similar to the [standard `<header>` section](#), with the following differences:

- The "type" for a Custom SNMP probe is "custom-snmp".
- There is a `FLAGS=xxx,xxx` command that takes the following optional items as parameters:
  - `NOLINKS` - Intermapper will not poll links (interfaces) with SNMP
  - `SNMPV2C` - Intermapper will use SNMPv2c to poll the device
  - `NOICMPFALLBACK` - Intermapper will not send an ICMP ping to a device if no SNMP responses return.
  - `MINIMAL` - the probe queries only its own (specified) variables.
  - `ALLOW-LOOPS` - In some network equipment, the indices for the `ifTable` and related tables do not proceed in the usual strictly increasing fashion, jumping around instead. Adding this flag to the header will instruct Intermapper to allow this situation. N.B. If the SNMP agent in your network device doesn't stop returning values when every item in the table has been read, set this flag to instruct Intermapper to loop over the table continuously until 5000 reads have occurred, at which point it will stop.
  - `IFINDEX-BUG` - Some network equipment will respond incorrectly to SNMP queries for the `ifTable` and related tables when Intermapper queries only certain entries in a sparse `ifTable`, rather than trying to query each possible index in turn. Add this flag to the header to instruct Intermapper to work around this situation rather than attempting to be efficient.
  - `LINKCRITICAL` - If a link of a device goes down and the flag is set, the device status changes to critical, and not to alarm, which is the default.

**Note:** The `old_protocol` and `old_script` parts, added for backward compatibility, are deprecated, and are ignored in any older probes that use them.

## Flags for Command-line Probes

The following Flag parameters are specific to command-line probes:

- NTCREDENTIALS - tells Intermapper to elevate its credentials using the username and password found in the NT Services server settings panel long enough to run the command line in the probe. (Windows only.)
- NAGIOS3 - Use "flags" = "NAGIOS3" in the `<header>` of a command-line probe to indicate that the return value should be treated as a Nagios Plugin. See the [Nagios Plugins](#) section.

## Probe File Locations

Probes files are saved in the *Probes* folder of the *Intermapper Settings* folder.

## Probe File Names

The probe files are named with two parts separated by a period ("."). The parts are:

- **package name** - must be unique for each organization creating probe files.

By convention, the package is composed of the organization's DNS domain name, with the segments reversed. Thus, the built-in probes for Intermapper all have a package of "com.dartware." Other organizations may create and share their own probes, since the file names will not collide.

- **probe name** - identifies the probe.

For example, the built-in Custom TCP probe is defined in the file named:

```
com.dartware.tcp.custom
```

The probe's package name and probe name are defined in the probe definition's header section. HelpSystems recommends that you name the file with the combination of the package name and probe name, as shown above.

The header section of the probe definition in the example above contains these two lines:

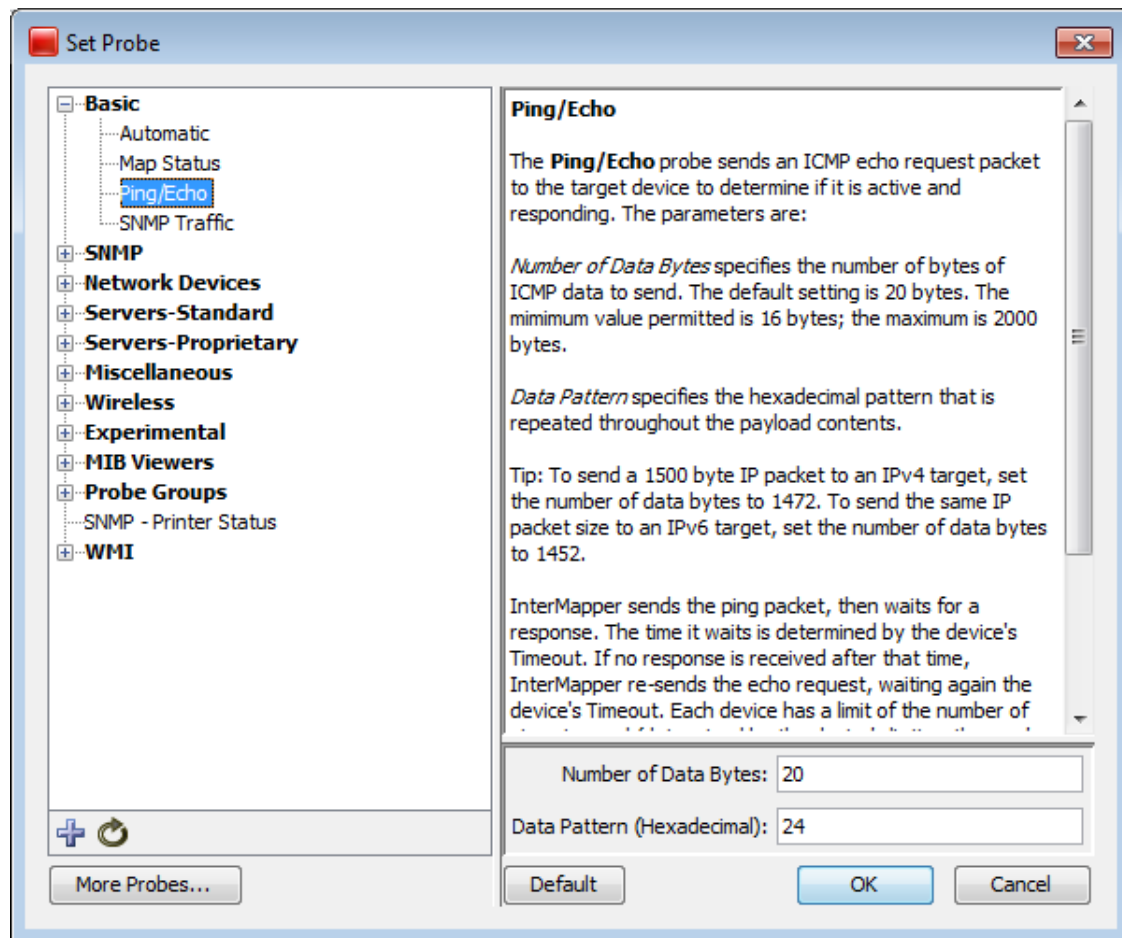
```
package = "com.dartware"  
probe name = "tcp.custom"
```

## The `<description>` Section

The Description section of a probe file contains text that will be displayed as a description of the probe in the Set Probe window. All probe types can have a description section. It is defined using the following tags:

```
<description> ... </description>
```

The description can be formatted using IMML, [InterMapper's Markup Language](#). The [Example Probe File](#) shows a sample description section.



The Set Probe window, showing the description field. Note that the blue underlined links are actually links to the relevant RFC specifications.

## The <parameters> Section

A probe can have one or more parameters. These parameters are set by the user in the Set Probe window (shown below). They are used for specifying numeric thresholds or strings to be sent to or received from the device.

The *parameter* section of the defines a set of name/value pairs with the format:

```
<parameters>
  [parameter name]= "[parameter value]"
</parameters>
```

Each parameter name/value appears in its own entry field in the Set Probe window.

Probe parameters are accessed and used just like variables. They can be used in calculations, alarm/warning thresholds, and displayed in the status window. To refer to a parameter whose name contains one or more spaces, the name will need to be enclosed in curly braces. (Example: "\${Seconds to wait}").

# Input Field Types

Four types of input fields are available:

- **Text** - Input a text string
- **Password** - Input a text string, obscuring the characters
- **Dropdown** - Choose from a dropdown menu.
- **Checkbox** - Set a variable to true or false by selecting or clearing a check box.

## Text Fields

This field type presents a simple text box for entering a string.

```
"Text" = "Text Value"
```

The line above sets the variable \${Text}

## Password Fields

You can create input parameters that conceal the string from casual view (so-called *password parameters*.) The data is displayed as a line of asterisks ("\*\*\*\*\*") when a user types the password. To specify a password parameter, place a single asterisk ("\*") after the name of the field, like this:

```
"Password*" = ""
```

Note that the variable name remains \${Password\*} and you have to refer to it as such in your script. The "\*" is removed before displaying the name, so the above password parameter would appear as "Password" in the Set Probe window.

## Dropdown Fields

You can create input parameter fields that present a dropdown menu from which the user can choose from a number of choices.

To create a dropdown field, use this syntax:

```
"Test[Equal,NotEqual]" = "NotEqual" //Default value is  
NotEqual
```



The values between brackets define the choices available to the user. The value on right of the statement is the initial value of the dropdown field.

You can use this parameter in expressions. The full variable is `${Test[Equal,NotEqual]}`, and it returns the current value of the dropdown as selected by the user. To display the value of a dropdown in the Status window, you must use the full variable definition.

```
\4\Dropdown:\0\  ${Test[Equal,NotEqual]}\0\
<snmp-device-variables>
  alarm: (${Dropdown[Choice1,Choice2,Choice3]} !=
    "Choice2") "It's not Choice2!"
</snmp-device-variables>
```

## Check Box Fields

To create a check box, use this syntax:

```
"Checkbox[true,false]" = "true" //Default value is "true"
```

You can use this parameter in expressions. The full variable is `${Checkbox[Equal,NotEqual]}`, and it returns the current value of the check box as selected by the user

## Parameter Section Example

This is an example parameter section that demonstrates the use of the four types of input fields. The screenshot below shows how each input field type appears.

```
<parameters>
  "Text"                                = "Text Value"
  "Password*"                           = ""
  "Dropdown[Choice1,Choice2,Choice3]"   = "Choice2"
  "Checkbox[true,false]"                = "true"
</parameters>
```

**Parameter Test Probe**

This probe shows the various data types that can be used as parameters to a custom probe.

Text: Text Value

Password: \*\*\*\*\*

Dropdown: Choice2

Checkbox: ☒

SNMP Version: SNMPv1 Port: 161

Community: \*\*\*\*\*

*Probe Parameters*

## The <datasets> section

Use the <datasets> section to define the datasets available for a probe. You can also specify which datasets are recorded automatically by default.

**Note:** The <datasets> section replaces the deprecated <autorecord> section. Documentation for the <autorecord> section is below.

The syntax for a <datasets> section is as follows. All columns except Column 1 should be in quotes.

```
<datasets>
  $variablename, "tag", "unitsOfMeasure", "autorecordFlag",
  "legend"
  ...
</datasets>
```

where:

- **\$variablename** - a variable defined by the probe
- **tag** - a short tag that identifies a particular class of dataset. Use these tags to create a report of like variables, such as CPU%, temperature, etc. See the [Automatically-Recorded Data Values](#) page to view pre-defined tags. Probe writers are free to create their own short tags as long as they don't start with "\_".
- **unitsOfMeasure** - the unit of measure used with the dataset. Choose from the list of Units of Measures below.

- **autorecordFlag** - a boolean flag that specifies whether the dataset should be recorded or not.
- **Legend** - a human-readable text string that appears as the legend label for the dataset. This legend overrides a legend placed in a `<snmp-device-variable>` section.

### Example

```
<datasets>
  $temp, "temp-tag", "degrees C", "false", "The Temperature"
  $atemp, "atemp-tag", "degrees C", "true", "Autorecord
Temperature"
</datasets>
```

## Auto-recording values

Certain data values collected from a device are recorded to the Intermapper Database automatically. You can specify other variables you want to record by default when data for a device is stored.

For all probes, the following data is recorded:

- response time (in msec) - tag: **BiRt**
- short-term packet loss (%) - tag: **RPkL**
- input byte rates for all visible interfaces - tag: **BytR**
- output byte rates for all visible interfaces - tag: **BytT**

In addition to the values listed above, built-in probes automatically record other values. The [Automatically-Recorded Data Values](#) page lists those values for each built-in probe.

For your own probes, you can specify that a dataset should be recorded by setting its `autorecordFlag` value to **true**.

## Units of Measure

Use the following units in the `unitsOfMeasure` column of the `<datasets>` section.

| Symbol  | Description  |
|---------|--------------|
| percent | percent      |
| min     | minutes      |
| sec     | seconds      |
| msec    | milliseconds |

|              |   |
|--------------|---|
| bytes        | bytes   |
| kbytes       | kilobytes   |
| packets      | packets   |
| errors       | errors  |
| discards     | discards  |
| frames/sec   | frames per second   |
| bytes/sec    | bytes per second  |
| bits/sec     | bits per second   |
| mbits/sec    | megabits per second   |
| discards/min | discards per minute   |
| errors/min   | errors per minute   |
| errors/sec   | errors per second   |
| failures/sec | failures per second   |
| retries/sec  | retries per second  |
| packets/sec  | packets per second  |
| requests/sec | requests per second   |
| degrees C    | degrees celsius   |
| degrees F    | degrees fahrenheit  |
| dBm          | the power ratio in decibels of the measured power referenced to one milliwatt |
| miles        | a measure of wireless transmission range, (for how many miles it is useful)   |
| volts        | volts   |

## The <autorecord> section (deprecated)

The <autorecord> section has been replaced by the <datasets> section, which provides a control for auto-recording any dataset. The autorecord section is available for backward compatibility; syntax is provided here for reference.

```
<autorecord>  
  $var1, 'tag1', "Legend 1 :units(xxx) "
```

```
$var2, 'tag2', "Legend 2"
$var3, 'tag3', "Legend 3"
</autorecord>
```

where:

- **\$varX** is a variable defined by the probe
- **tagX** is a short tag that identifies a particular class of dataset. Use these tags to create a report of like variables, such as CPU%, temperature, etc. See the [Automatically-Recorded Data Values](#) page to view pre-defined tags. Probe writers are free to create their own short tags as long as they don't start with "\_".
- **Legend X :units(xxx)** is a human-readable text string that describes the dataset, and shows the customer the kinds of data being collected for a particular device. Specify the units for the dataset using the optional ":units" attribute. This legend overrides a legend placed in a <snmp-device-variable> section. In the <datasets> section, :units(xxx) has been replaced with unitsOfMeasure.

### Example

```
<autorecord>
  $lcpu.busyPer, 'cpupercent', "CPU Percent :units(%)"
  $lcpu.avgBusy1, 'cpupercentavg', "Average CPU Percent
:units(%)"
  $lmem.freeMem, 'freemem', "Available memory :units(bytes)"
</autorecord>
```

## Automatically-Recorded Data Values

The following values were selected to be recorded automatically from built-in probes.

| Probe Name/<br>File Name                                       | Variable Name   | Tag(30)       | Units   | Legend<br>(255)                |
|--|-----------------|---------------|---------|--------------------------------|
| Miscellaneous/Legacy/Cisco (v2c)<br>com.dartware.snmpv2c.cisco | \$lcpu.busyPer  | cpupercent    | percent | CPU Percent Busy               |
|  | \$lcpu.avgBusy1 | cpupercentavg | percent | Average CPU Percent over 1 min |

|  |                     |               |         |                                |
|--|---------------------|---------------|---------|--------------------------------|
|  | \$lcpu.avgBusy5     | cpupercentage | percent | Average CPU Percent over 5 min |
|  | \$lmem.freeMem      | freemem       | bytes   | Available Memory               |
| <b>Miscellaneous/TCP Check</b><br>com.dartware.snmp.tcpc<br>heck                         | \$tcpCurrEstab      | numconns      |         | Number of TCP Connections      |
| <b>Network Devices/Cisco/Cisco - IP SLA Jitter</b><br>com.dartware.snmp.cisco-ip-sla.txt | \$cpmCPUTotal1min   | cpupercentage | percent | Average CPU Percent            |
|  | \$AvgJitter         | jitteravg     | msec    | Average Jitter Value           |
|  | \$AvgLatency        | latencymsec   | msec    | Average Latency                |
|  | \$PercentPacketLoss | pktloss       | percent | Jitter Test Packet Loss        |
| <b>Network Devices/Cisco/Cisco - Old CPU MIB</b><br>com.dartware.snmp.cisco              | \$lcpu.busyPer      | cpupercentage | percent | CPU Percent Busy               |
|  | \$lcpu.avgBusy1     | cpupercentage | percent | Avg. CPU Percent over 1 min    |

|  |                        |               |         |                             |
|--|------------------------|---------------|---------|-----------------------------|
|  | \$lcpu.avgBusy5        | cpupercentage | percent | Avg. CPU Percent over 5 min |
|  | \$lmem.freeMem         | freemem       | bytes   | Available Memory            |
| <b>Network Devices/Cisco/Cisco - Process and Memory Pool</b><br>com.dartware.snmp.cisco.newmib | \$lcpu.busyPer         | cpupercentage | percent | CPU Percent Busy            |
|  | \$lcpu.avgBusy1        | cpupercentage | percent | Avg. CPU Percent over 1 min |
|  | \$lcpu.avgBusy5        | cpupercentage | percent | Avg. CPU Percent over 5 min |
|  | \$ciscoMemoryPoolFree1 | freemem       | bytes   | Available Memory #1         |
|  | \$ciscoMemoryPoolFree2 | freemem       | bytes   | Available Memory #2         |
| <b>Network Devices/UPS/APC UPS - AP961x</b><br>com.dartware.ups.apc-ap961x.txt                 | \$leftCharge           | pctcharge     | percent | Percent Charge              |

|   |                  |                  |               |                                     |
|---|------------------|------------------|---------------|-------------------------------------|
|   | \$batMin         | batttimele<br>ft | min           | Time<br>left on<br>battery          |
|   | \$inVolt         | involts          | volts         | Input<br>Voltage                    |
|   | \$batTempC       | temperatu<br>re  | degree<br>s C | Battery<br>Temper<br>ature<br>(°C)  |
| <b>Network<br/>Devices/UPS/APC UPS</b><br>com.dartware.ups.apc.txt                      | \$leftCharge     | pctcharge        | percen<br>t   | Percent<br>Charge                   |
|   | \$batMin         | batttimele<br>ft | min           | Time<br>left on<br>battery          |
|   | \$inVolt         | involts          | volts         | Input<br>Voltage                    |
|   | \$batTempC       | temperatu<br>re  | degree<br>s C | Battery<br>Temper<br>ature<br>(°C)  |
| <b>Network<br/>Devices/UPS/BestPowe<br/>r UPS</b><br>com.dartware.ups.bestpo<br>wer.txt | \$cTimeOnBattery | batttimele<br>ft | min           | Time<br>Left on<br>Battery<br>(min) |
|   | \$cInputVoltage  | involts          | volts         | Input<br>Voltage                    |
|   | \$cIntTempC      | temperatu<br>re  | degree<br>s C | Internal<br>Temper<br>ature (C)     |
| <b>Network<br/>Devices/UPS/Exide UPS</b><br>shef.ac.uk.ups.exide.txt                    | \$LeftCharge     | pctcharge        | percen<br>t   | Battery<br>Charge<br>Left           |
|   | \$LeftMin        | batttimele<br>ft | min           | Time<br>Left on<br>Battery          |



|  |                |              |           |                          |
|--|----------------|--------------|-----------|--------------------------|
|  | \$in1Volt      | involts      | volts     | Input 1 Voltage          |
| <b>Network Devices/UPS/Liebert UPS - OpenComms</b><br>com.dartware.ups.liebert-opencomms.txt | \$LeftCharge   | pctcharge    | percent   | Percent Charge           |
|  | \$LeftMin      | batttimeleft | min       | Time Left on Battery     |
|  | \$in1Volt      | involts      | volts     | Input 1 Voltage          |
|  | \$batteryTempC | temperature  | degrees C | Battery Temperature (°C) |
| <b>Network Devices/UPS/Standard UPS (RFC1628)</b><br>com.dartware.ups.standard.txt           | \$LeftCharge   | pctcharge    | percent   | Percent Charge           |
|  | \$LeftMin      | batttimeleft | min       | Time Left on Battery     |
|  | \$in1Volt      | involts      | volts     | Input 1 Voltage          |
|  | \$batTempC     | temperature  | degrees C | Battery Temperature      |
| <b>Network Devices/UPS/Tripplite UPS</b><br>com.dartware.ups.tripplite.txt                   | \$LeftCharge   | pctcharge    | percent   | Percent Charge           |

|   |                                |                  |               |                                      |
|---|--------------------------------|------------------|---------------|--------------------------------------|
|   | \$LeftMin                      | batttimele<br>ft | min           | Time<br>Left on<br>Battery           |
|   | \$in1Volt                      | involts          | volts         | Input 1<br>Voltage                   |
|   | \$envTempC                     | temperatu<br>re  | degree<br>s C | Ambient<br>Temper<br>ature<br>(°C)   |
|   | \$envHumid                     | humidity         | percen<br>t   | Ambient<br>Humidit<br>y              |
| <b>Network<br/>Devices/UPS/Victron<br/>UPS</b><br>de.medianet.freinet.ups.vi<br>ctron.txt           | \$batt.rem                     | batttimele<br>ft | min           | Battery<br>Time<br>Remaini<br>ng     |
|   | \$input.volt1                  | involts          | volts         | Input<br>Voltage<br>Phase 1          |
| <b>Servers-<br/>Proprietary/Apple/OS X<br/>Server/AFP</b><br>com.dartware.tcp.osxserv<br>er.afp.txt | \$currentConnectio<br>ns       | connectio<br>ns  |               | Connect<br>ions                      |
|   | \$currentThroughp<br>ut        | throughp<br>ut   | bytes/s<br>ec | Through<br>put                       |
| <b>Servers-<br/>Proprietary/Apple/OS X<br/>Server/FTP</b><br>com.dartware.tcp.osxserv<br>er.ftp.txt | \$realConnectionC<br>ount      | authconns        |               | Authent<br>icated<br>Connect<br>ions |
|   | \$anonymousConn<br>ectionCount | anonconn<br>s    |               | Anonym<br>ous<br>Connect<br>ions     |

|  |                                |                      |                  |                          |
|--|--------------------------------|----------------------|------------------|--------------------------|
| <b>Servers-Proprietary/Apple/OS X Server/Info</b><br>com.dartware.tcp.osxserv<br>er.info.txt   | \$cpu                          | cpuperce<br>nt       | percen<br>t      | CPU<br>Usage             |
| <b>Servers-Proprietary/Apple/OS X Server/NAT</b><br>com.dartware.tcp.osxserv<br>er.nat.txt     | \$activeTCP                    | tcpconns             |                  | TCP<br>Links             |
|  | \$activeUDP                    | udpconns             |                  | UDP<br>Links             |
|  | \$activeICMP                   | icmpconn<br>s        |                  | ICMP<br>Links            |
| <b>Servers-Proprietary/Apple/OS X Server/Print</b><br>com.dartware.tcp.osxserv<br>er.print.txt | \$currentQueues                | queues               |                  | Current<br>Queues        |
|  | \$currentJobs                  | numjobs              |                  | Spooled<br>Jobs          |
| <b>Servers-Proprietary/Apple/OS X Server/QTSS</b><br>com.dartware.tcp.osxserv<br>er.qtss.txt   | \$currentConnectio<br>ns       | connectio<br>ns      |                  | Connect<br>ions          |
|  | \$currentThroughp<br>ut        | throughp<br>ut       | bytes/s<br>ec    | Through<br>put           |
| <b>Servers-Proprietary/Apple/OS X Server/Web</b><br>com.dartware.tcp.osxserv<br>er.web.txt     | \$currentRequests<br>By10      | requestrat<br>e      | request<br>s/sec | Request<br>Rate          |
|  | \$cacheCurrentReq<br>uestsBy10 | requestrat<br>ecache | request<br>s/sec | Cache<br>Request<br>Rate |

|  |                          |                 |           |                              |
|--|--------------------------|-----------------|-----------|------------------------------|
|  | \$currentThroughput      | throughput      | bytes/sec | Throughput                   |
|  | \$cacheCurrentThroughput | throughputcache | bytes/sec | Cache Throughput             |
| <b>Servers-Proprietary/Barracuda/Barracuda HTTP</b><br>com.dartware.tcp.barracuda.http.txt   | \$in_queue_size          | inqueue         |           | Input Queue                  |
|  | \$out_queue_size         | outqueue        |           | Output Queue                 |
|  | \$avg_latency            | latencysec      | sec       | Average Message Latency      |
| <b>Servers-Proprietary/Barracuda/Barracuda HTTPS</b><br>com.dartware.tcp.barracuda.https.txt | \$in_queue_size          | inqueue         |           | Input Queue                  |
|  | \$out_queue_size         | outqueue        |           | Output Queue                 |
|  | \$avg_latency            | latencysec      | sec       | Average Message Latency      |
| <b>Servers-Proprietary/Microsoft/DHCP Lease Check</b><br>com.dartware.snmp.dhcpcheck.txt     | \$noAddFree              | dhcpcfree       |           | Number of DHCP Leases Free   |
|  | \$noAddInUse             | dhcpcinuse      |           | Number of DHCP Leases In Use |

|  |                   |                   |             |  |
|--|-------------------|-------------------|-------------|--|
|  | \$noPending       | dhcpend<br>ing    |             | Number<br>of<br>Pending<br>Offers          |
| <b>Servers-<br/>Standard/Custom TCP</b><br>com.dartware.tcp.custom   | \$_connect        | conntime          | msec        | Time to<br>establis<br>h<br>connect<br>ion |
|  | \$_active         | connactiv<br>e    | msec        | Time<br>spent<br>connect<br>ed to<br>host  |
| <b>Servers-Standard/Host<br/>Resources</b><br>com.dartware.snmp.hrmi<br>b                                    | \$_CPUUtilization | cpuperce<br>ntavg | percen<br>t | Average<br>CPU<br>Percent                  |
| <b>Servers-Standard/HTTP<br/>&amp; HTTPS/HTTP (Follow<br/>Redirects)</b><br>com.dartware.tcp.http.fol<br>low | \$_connect        | conntime          | msec        | Time to<br>establis<br>h<br>connect<br>ion |
|  | \$_active         | connactiv<br>e    | msec        | Time<br>spent<br>connect<br>ed to<br>host  |
| <b>Servers-Standard/HTTP<br/>&amp; HTTPS/HTTP (Post)</b><br>com.dartware.tcp.http.cg<br>i.post               | \$_connect        | conntime          | msec        | Time to<br>establis<br>h<br>connect<br>ion |
|  | \$_active         | connactiv<br>e    | msec        | Time<br>spent<br>connect<br>ed to<br>host  |

|  |            |            |      |                              |
|--|------------|------------|------|------------------------------|
| <b>Servers-Standard/HTTP &amp; HTTPS/HTTP (Proxy)</b><br>com.dartware.tcp.http.proxy               | \$_connect | conntime   | msec | Time to establish connection |
|  | \$_active  | connactive | msec | Time spent connected to host |
| <b>Servers-Standard/HTTP &amp; HTTPS/HTTP (Redirect)</b><br>com.dartware.tcp.http.redirect         | \$_connect | conntime   | msec | Time to establish connection |
|  | \$_active  | connactive | msec | Time spent connected to host |
| <b>Servers-Standard/HTTP &amp; HTTPS/HTTP</b><br>com.dartware.tcp.http                             | \$_connect | conntime   | msec | Time to establish connection |
|  | \$_active  | connactive | msec | Time spent connected to host |
| <b>Servers-Standard/HTTP &amp; HTTPS/HTTPS (Follow Redirects)</b><br>com.dartware.tcp.https.follow | \$_connect | conntime   | msec | Time to establish connection |

|  |            |            |         |                              |
|--|------------|------------|---------|------------------------------|
|  | \$_active  | connactive | msec    | Time spent connected to host |
| <b>Servers-Standard/HTTP &amp; HTTPS/HTTPS (Post)</b><br>com.dartware.tcp.https.cgi.post   | \$_connect | conntime   | msec    | Time to establish connection |
|  | \$_active  | connactive | msec    | Time spent connected to host |
| <b>Servers-Standard/HTTP &amp; HTTPS/HTTPS (SSLv3)</b><br>com.dartware.tcp.https.notls.txt | \$_connect | conntime   | msec    | Time to establish connection |
|  | \$_active  | connactive | msec    | Time spent connected to host |
| <b>Servers-Standard/HTTP &amp; HTTPS/HTTPS</b><br>com.dartware.tcp.https                   | \$_connect | conntime   | msec    | Time to establish connection |
|  | \$_active  | connactive | msec    | Time spent connected to host |
| <b>SNMP/Comparison</b><br>com.dartware.snmp.oidcomparison.txt                              | \$theOID   | \$Tag      | \$Units | \$Legend                     |

|  |          |       |         |          |
|--|----------|-------|---------|----------|
| <b>SNMP/High Threshold</b><br>com.dartware.snmp.oidhigh.txt                | \$theOID | \$Tag | \$Units | \$Legend |
| <b>SNMP/Low Threshold</b><br>com.dartware.snmp.oidlow.txt                  | \$theOID | \$Tag | \$Units | \$Legend |
| <b>SNMP/Range Threshold</b><br>com.dartware.snmp.oidrange.txt              | \$theOID | \$Tag | \$Units | \$Legend |
| <b>SNMP/Single OID Viewer</b><br>com.dartware.snmp.oidsingle.txt           | \$theOID | \$Tag | \$Units | \$Legend |
| <b>SNMP/String Comparison</b><br>com.dartware.snmp.oidstringcomparison.txt | \$theOID | \$Tag | \$Units | \$Legend |



# Probe Status Windows

When you create a custom probe, you can override the default contents of Status windows. How you do this depends on the type of probe, as follows:

- The `<snmp-device-display>` section - for SNMP probes
- The `<snmp-device-display>` section - for SNMP trap probes
- The `<script-output>` section - for TCP probes
- The `<command-display>` section - for Command-line probes

All of these sections can be formatted using IMML, [Intermapper's Markup Language](#). See the `<snmp-device-display>` example below.

## Controlling the Status window in SNMP Probes with `<snmp-device-display>`

Use the optional `<snmp-device-display>` section to describe the text that appears in a custom SNMP probe's Status window. Probe variables are replaced with their values in the Status window's text.

The default font for the Status window's text is a mono-spaced font, so alignment of text is straightforward. You can change the appearance of the text in the Status window using IMML, [Intermapper's Markup Language](#).

Here is a sample `<snmp-device-display>` section. Note that the variables are replaced with the values retrieved from the device, and that formatting is controlled by IMML.

```
<snmp-device-display>
  \B5\Custom SNMP Probe\0P\
  \4\ipForwDatagrams:\0\ ${ipForwDatagrams} datagrams/sec
  \4\ipInHdrErrors:\0\   ${ipInHdrErrors} errors/minute
  \4\tcpCurrEstab:\0\    ${tcpCurrEstab} connections
</snmp-device-display>
```

## Controlling the Status window in TCP Probes with `<script-output>`

Use the optional `<script-output>` section to describe the text that appears in a TCP-based custom probe's Status window. The data in this section appears in the Status

window when you click and hold the device on the map.

## Controlling the Status window in Command-Line Probes with <command-display>

Use the optional <command-display> section to describe the text that appears in a command-line-based custom probe's Status window. The data in this section appears in the Status window when you click and hold the device on the map.

The format of this section is the same as the <snmp-device-display> section described above.

# IMML - Intermapper Markup Language

You can apply text styles to a probe's description text or to the content in a Status window using IMML, Intermapper's markup language. IMML consists of formatting commands bracketed by backslash ("\") characters. There may be many markup commands between a pair of \...\ characters. The [Example Probe File](#) shows a sample description section.

*Historical note:* Prior to Intermapper 4.0, the markup characters were « and » (&le; and &ge;). Intermapper still accepts these characters, although we encourage everyone to use the \...\ in new probe files as they're easier to type and will pass unchanged through all mail systems.

## How Markup Tags Are Applied

- A markup commands applies to all text that follows it.
- Subsequent markup tags may add to or counteract a previous set of markup tags.

## Markup Tag Summary

| Tag      | Action  |
|----------|---|
| <b>M</b> | Set the font to a mono-spaced font.   |
| <b>G</b> | Set the font to Geneva or other proportional-spaced font).  |
| <b>+</b> | Increase the font size by one. Multiples ("++") increase the font size by the corresponding amount. |
| <b>-</b> | Decrease the font size by one. Multiples are allowed.   |
| <b>B</b> | Set following text in bold.   |
| <b>I</b> | Set following text in italic.   |
| <b>P</b> | Set following text to plain. This undoes all other stylings   |
| <b>U</b> | Set following text to underlined. See <b>Creating a link</b> below for making hyperlinks.           |
| <b>!</b> | Turn off a format that is on.   |

**digit** Set text color to one of the following:

0: Black

4: Light blue

1: Red

5: Green

2: Blue

6: Orange

3: Gray

7: Yellow

## Examples

The following description text is rendered as shown:

```
\b\Bold \i\Bold Italic  
\!b\Italic \p\Plain
```

**Bold *Bold Italic Italic* Plain**

```
\M1++\Big red monospace\p\
```

**Big red monospace**

```
\2U\http://www.example.com  
\p0\
```

<http://www.example.com>

```
\2U=http://www.example.com  
\Text Link\p0\
```

[Text Link](http://www.example.com)

## Creating a link

The last two examples above shows the script code required to create a link. In both example, "\2U\" means "set the color to blue, underline the text."

### Special Cases:

- If, as in the first of the two link examples above, the only text between the opening and closing tags is a URL (e.g., <http://www.example.com>), Intermapper treats it as a link to that page.
- If, as in the last link example above, the underline tag contains "[URL]", the text following the backslash ("Text Link" in the example) appears as blue and underlined.
- In both cases, clicking the text opens that page in a browser.

# Probe Comments

Comments in Intermapper probe files are quite similar to those in HTML. The comments may be interspersed anywhere in a probe file.

Note that HTML comments have a complicated syntax that can be simplified by following this rule:

Begin a comment with "`<!--`", end it with "`-->`", and do not use "`--`" within the comment.

Use this rule with Intermapper as well.

Example:

```
<!--  
  This is a probe comment.  
  It spans several lines.  
  It contains no double-hyphens.  
-->
```

## One-line Comments

You can also use the comment indicator "`--`" at the beginning of a line. The remainder of the line is ignored.

Example:

```
-- This line is a comment
```

# Built-in Probe Variables and Macros

Here is a list of built-in variables available in custom probes and notifiers. Note that certain variables are available only in certain contexts. The variables are listed by context.

- [Command-line Probe Variables](#)
- [SNMP Probe Variables](#)
- [TCP Probe Variables](#)
- [Command Line Notifier Variables](#)
- [The Chartable macro](#)
- [The Eval Macro](#)
- [The Scalable10 and Scalable2 Macros](#)

## Command Line Probe Variables

The following variables are available in the specified sections of Command-line probes.  
(probe-type=cmd-line)

### The <command-line> and <command-exit> sections

The following variables are available in <command-line> and <command-exit> sections of command line probes

| Variable Name               | Variable Description   |
|-----------------------------|--|
| <code>\${address}</code>    | The network address of the device.   |
| <code>\${devicename}</code> | The device name of the device.<br><b>Note:</b> In some cases, the device name may resolve to the device's IP address.                |
| <code>\${port}</code>       | The network port number that is being monitored.   |
| <code>\${exit_code}</code>  | The exit code of the command-line probe.<br><br><b>Note:</b> The <code>\${exit_code}</code> variable is used in <command-exit> only. |

|                                   |   |
|-----------------------------------|---|
| <b><code>\${cscript}</code></b>   | Evaluates to the full path to the Windows cscript.exe utility; it also automatically adds /nologo as a command-line option.<br><br><b>Note:</b> This variable is only available in Windows. |
| <b><code>\${python}</code></b>    | Evaluates to the full path to the python interpreter installed as part of Intermapper DataCenter; it also automatically adds any necessary command-line options for normal operation.       |
| <b><code>\${community}</code></b> | The community string for the device.  |
| <b><code>\${mapname}</code></b>   | The name of the map containing the device being probed.   |
| <b><code>\${mapid}</code></b>     | The internal ID of the map containing the device being probed.  |

## The <command-display> section

The following variables are available in the <command-display> section of command line probes. (probe-type=cmd-line)

| Variable Name  | Variable Description   |
|--|--|
| <b><code>\${devicename}</code></b>   | The device's name taken from first line of the label.  |
| <b><code>\${deviceaddress}</code></b>  | The network address of the device.   |
| <b><code><a href="#">\${eval:}</a></code></b>  | The eval macro.  |
| <b><code><a href="#">\${chartable[:fmt]:expr}</a></code></b>   | Evaluate <code>expr</code> and format the result as a chartable value  |
| <b><code><a href="#">\${scalable2:fmt:expr}</a></code><br/><b><code><a href="#">\${scalable10:fmt:expr}</a></code></b></b> | Scales large numbers into smaller units for better readability. The values are chartable.  |
| <b><code><a href="#">\${^stdout}</a></code></b>  | Any output written to the standard output of a command-line script. See below for additional information about the effect of <code>\${^stdout}</code> on the <code>reason</code> string. |
| <b><code>\${nagios_output}</code></b>  | Parses a Nagios plugin's output for display.   |

## SNMP Probe variables

A variable name consists of letters, digits and an underscore, and must begin with a letter. Variable names are not case-sensitive. A variable name can be referred to in the probe as `$VariableName` or `${VariableName}`. Use the bracketed form for variables and parameters that have one or more spaces in the name.

The variables listed below are available in SNMP Probes. (probe-type = customsnmp)

In the `<snmp-device-display>` section of a probe file, an occurrence of a variable name is replaced with its value, rounded to the nearest integer.

For example, if a calculation variable has the value of 3.14159265, using it in the `<display-output>` section results in the value of "3"; if the variable had the value 4.75 it is displayed as "5".

This value is chartable; clicking it makes a new chart, and dragging it adds it to an existing chart. If you need to display a non-integer value for the variable, use [the `\$\(chartable\)` macro](#) described below.

## The `<snmp-device-display>` section

| Variable Name   | Variable Description  |
|---|---|
| <code>\$(devicename)</code>   | The device's name taken from first line of the label.   |
| <code>\$(deviceaddress)</code>  | The network address of the device.  |
| <code>\$(imserveraddress)</code>  | The network address of the Intermapper Server.  |
| <code>\$(alarmpointlist)</code>   | The list of alarm points.   |
| <a href="#">\$(eval:expr)</a>   | The eval macro.   |
| <a href="#">\$(chartable[:fmt]:expr)</a>  | The chartable macro.  |
| <a href="#">\$(scalable2:fmt:expr)</a><br><a href="#">\$(scalable10:fmt:expr)</a> | Scales large numbers into smaller units for better readability. The values are chartable.   |
| <code>\$([variablename]:legend)</code>  | The variable's legend as specified in the <code>&lt;snmp-device-variables&gt;</code> section For more information, see <a href="#">SNMP Probe Variables</a> . |

## The `<snmp-device-properties>` section

| Variable Name            | Variable Description       |
|--------------------------|----------------------------|
| <code>\$(ifIndex)</code> | The interface index.       |
| <code>\$(ifType)</code>  | The interface type.        |
| <code>\$(ifDescr)</code> | The interface description. |
| <code>\$(ifAlias)</code> | The interface's alias      |



## The OID column of the <snmp-device-variables> section

| Variable Name                      | Variable Description  |
|------------------------------------|---|
| <code>\${SpecificTrap}</code>      | Trap Field: specific-trap (SNMP v1; generic-trap is enterpriseSpecific)   |
| <code>\${GenericTrap}</code>       | Trap Field: generic-trap (SNMP v1)  |
| <code>\${TimeStamp}</code>         | Trap Field: trap timestamp (SNMP v1, v2c)   |
| <code>\${Enterprise}</code>        | Trap Field: enterprise (SNMP v1)  |
| <code>\${CommunityString}</code>   | Trap Field: community (SNMP v1, v2c)  |
| <code>\${TrapOID}</code>           | Trap Field: trap OID (SNMPv2c, v3)  |
| <code>\${SnmVersion}</code>        | Trap Field: trap version  |
| <code>\${SenderAddress}</code>     | Trap Field: trap sender's address   |
| <code>\${AgentAddress}</code>      | Trap Field: trap agent's address (if different from sender)   |
| <code>\${VarbindCount}</code>      | Trap Field: count of varbind variables<br><br><b>Note:</b> the next three macros do not use a colon<br><code>\${VarbindValue8}</code> returns the value of the eighth Varbind item. |
| <code>\${VarbindOID[NNN]}</code>   | Trap Field: <i>NNNth</i> varbind OID  |
| <code>\${VarbindValue[NNN]}</code> | Trap Field: <i>NNNth</i> varbind Value  |
| <code>\${VarbindType[NNN]}</code>  | Trap Field: <i>NNNth</i> varbind Type   |

## TCP Probe Variables

These variables are available in TCP probes. (probe-type = tcp-script)

### The <script> section

| Variable Name | Variable Description |
|---------------|----------------------|
|---------------|----------------------|

|                                   |  |
|-----------------------------------|--|
| <code>\${_remoteaddress}</code>   | The network address of the remote end of the connection.   |
| <code>\${_remoteport}</code>      | The network port number of the remote end of the connection.   |
| <code>\${_localaddress}</code>    | The network address of the local end of the connection.  |
| <code>\${_localport}</code>       | The network port number of the local end of the connection.  |
| <code>\${_gmttime}</code>         | The current time in RFC 822 format.  |
| <code>\${_version}</code>         | The version number of the Intermapper program.   |
| <code>\${_line:len}</code>        | The text of the last line received, truncated to the given length.   |
| <code>\${_idletimeout}</code>     | The idle timeout for the probe in seconds.   |
| <code>\${_stringtomatch}</code>   | The string we attempted to match in the last EXPT or MTCH command.   |
| <code>\${_base64:str}</code>      | Encode the given argument into base64.   |
| <code>\${_cvspassword:str}</code> | Encode the given argument using the cvs password algorithm.  |
| <code>\${_md5:str}</code>         | The MD5 hash of the given argument, in hexadecimal.  |
| <code>\${_idleline}</code>        | The line number of the script where we were before the idle handler was invoked.   |
| <code>\${_secsconnected}</code>   | The number of seconds the probe spent connected to the other end. May be 0 if we were disconnected immediately or connection failed. |
| <code>\${_length:str}</code>      | The length of the given argument in bytes.   |
| <code>\${_float:num}</code>       | The argument "pretty-printed" as a floating point number via printf %g.  |
| <code>\${_hmac:key:msg}</code>    | The HMAC-MD5 of the message, using the given key.  |
| <code>\${_urlencode:str}</code>   | Encodes the specified string for use in URLs.  |

## The <script-output> section

| Variable Name   | Variable Description  |
|---|---|
| <code>\${devicename}</code>   | The device's name taken from first line of the label.                                     |
| <code>\${deviceaddress}</code>  | The network address of the device.  |
| <code><a href="#">\${eval:}</a></code>  | The eval macro.   |
| <code><a href="#">\${scalable2:fmt:expr}</a></code><br><code><a href="#">\${scalable10:fmt:expr}</a></code> | Scales large numbers into smaller units for better readability. The values are chartable. |

## Variables passed to Command-line Notifiers

The following variables are available for passing to a command line notifier. These values allow you to pass messages or URLs as command-line arguments in formats that are platform-friendly.

| Variable Name                     | Variable Description   |
|-----------------------------------|--|
| <code>\${message}</code>          | The notifier's message text. (On Windows, each double-quote " is escaped by \".)   |
| <code>\${stripped_message}</code> | The notifier's message text with quotes ( ' and " ) removed and newlines (\r and \n) replaced by space.                      |
| <code>\${escaped_message}</code>  | The notifier's message text escaped for url syntax (e.g. %20 for space, etc.)  |
| <code>\${urlescape:str}</code>    | Escapes a string specified in <b>str</b> for use in a URL. Any macros included in <b>str</b> are expanded prior to escaping. |

## Macros

Intermapper supports several macros that can control and manipulate the way variables are displayed, as well as their use in charts.

### The `${chartable}` macro

Use the `${chartable}` macro to evaluate **expr** and to format the result as a chartable value.

## Usage

```
${chartable [:min][:max][:fmt]:expr}
```

In the output section of a probe file, the `${chartable: ...}` macro creates an underlined value that can be clicked to add it to a chart. The macro also controls the field width and number of decimal places. There are two parameters:

- **min/max** - Use the **min** and **max** parameters to enter a range the chart uses for display of the data.

**NOTE:** In order to be parsed correctly, the **min** and **max** parameters must be immediately preceded by a plus (+) or minus (-).

- **fmt** - A formatting string that indicates the number and placement of the digits near the decimal point, and the variable to be formatted. The formatting string can be either a mask composed with the '#' symbol or a quoted printf specifier like those accepted by the [sprintf](#) function.
- **expr** - A variable or an expression (but not a macro). Intermapper evaluates the expression and displays the result according to the formatting string.

## Examples

If the variable `$pi` is set to 3.14159265,

```
${chartable: #.## : $pi }:          --> 3.14
${chartable: #.##### : $pi }:      --> 3.1415927
${chartable: "%3d" : $pi }:         --> 3 (with 2 leading
spaces)
${chartable: "%9.7f" : $pi}:         --> 3.1415927
${chartable: "%11.7f" : $pi}:        --> 3.1415927 (also with
2 leading spaces)
${chartable: #.##### : $pi*100}:    --> 314.1592650
${chartable: "%9.7f" : $pi*1000}:    --> 3141.5926500
${chartable: "%11.7f" : $pi*10000}: --> 31415.9265000 (no
leading spaces)
```

If the variable `$speed` can be greater than 4Gb,

```
${chartable: +0:+10E9 : "%d" : $speed }: --> decimal integer
${chartable: +0:+10E9 : "%e" : $speed }: --> exponential
notation
```

If the variable `$value` can be a positive or negative value of more than 4Gb,

```
${chartable: -10E9:+10E9 : "%d" : $value }: --> decimal
integer
${chartable: -10E9:+10E9 : "%e" : $value }: --> exponential
notation
```

## The `#{eval}` macro

Use the `#{eval}` macro to compute a value in the output of a script. It is available in the following contexts:

- The `<command-display>` section of command-line probes
- The `<snmp-device-display>` section of SNMP probes
- The `<script-output>` section of TCP probes

## Usage

The syntax for the `#{eval}` macro is as follows:

```
#{eval:[expr]}
```

The expression can use any operators or functions defined in [Probe Calculations](#), allowing you to perform variable assignments, arithmetic calculations, relational and logical comparisons, as well as use built-in functions to perform bitwise, rounding and mathematical operations. You can also perform operations on strings using `sprintf` formatting and regular expressions.

## Examples

Here are some examples that use the Eval macro:

```
Arithmetic: #{eval:${test} := (4-1)*(2+1)/(9/3)}
<!-- result = 3, also assigns result to ${test} -->

Modulo: #{eval:${test}%2}
<!-- result = 1, uses the ${test} variable -->

String assign: #{eval:${yes}:="Yes"} #{eval:${no}:="No"}
Numeric assign: #{eval:${test}:=5} (
Conditional: #{eval: $test==5 ? ($response := "Yes") :
($response := "No")}
<!-- result="Yes", because ${test} variable = 5
      result also assigned to ${response} variable, output on
the next line -->
${response}

subid(): #{eval:subid("1.3.6.1.2.1.4.20.1.1.10.10.2.20", -4,
4)}
<!-- result="10.10.2.20" -->

Regular Expression: #{eval: "test123" =~ "^te([st]*) ([0-
```

```
9]*)"; "${1} ${2}"}  
<!-- result = "st 123"-->
```

## The `${variablename:legend}` macro

In the `<snmp-device-display>` section of a probe file, the `${variablename:legend}` macro is replaced with the legend field defined for that variable in the `<snmp-device-variables>` section. For example, given this definition:

```
<snmp-device-variables>  
    ipForwDatagrams, 1.3.6.1.2.1.4.6.0, PER-SECOND, "Forwarded  
datagrams"  
</snmp-device-variables>
```

The following entry in the `<snmp-device-display>` section shows "Forwarded datagrams".

```
${ipForwDatagrams:legend}
```

## The `${scalable10}` and `${scalable2}` macros

Use the `${scalable10}` and `${scalable2}` macros to display numbers in the appropriate scaled units. The values are always between 1.0 and 1000, are chartable, and are scaled by a factor of 1000 or 1024:

- The `${scalable10}` macro scales by a factor 1000 (for msec, Mbps, etc.).
- The `${scalable2}` macro scales by a factor of 1024 (for KBytes, GBytes, Terabytes, etc.)

Both macros display the appropriate scale. They use the same syntax as the `${chartable}` macro.

## Usage

`${scalable10[:fmt]:expr}`

`${scalable2[:fmt]:expr}`

## Example

```
${scalable10: ### : 12304 }bytes => 12.30 kbytes  
${scalable10:"%3.2d" : 12304 }bytes => 12.30 kbytes
```

```
${scalable2: ### : 12304 }bytes => 12.02 kbytes  
${scalable2: "%3.2d" : 12304 }bytes => 12.02 kbytes
```

The following examples use the `scalable10` macro.

| char   | scale | factor | short   | for example  | val   | units | example       |
|--------|-------|--------|---------|--------------|-------|-------|---------------|
| output |       |        |         |              |       |       |               |
| -----  | ----- | -----  | -----   | -----        | ----- | ----- | -----         |
| -      |       |        |         |              |       |       |               |
| k      | *     | 1000   | kilo    | 12304        |       | bytes | 12.30 kbytes  |
| M      | *     | 1e6    | Mega    | 3421814      |       | bytes | 3.42 Mbytes   |
| G      | *     | 1e9    | Giga    | 125032100300 |       | bytes | 125.03 Gbytes |
| T      | *     | 1e12   | Tera    | 1.23 x 10^12 |       | bytes | 1.23 Tbytes   |
| none   | *     | 1      | nothing | 123          |       | bytes | 123.00 bytes  |
| m      | /     | 1000   | milli   | 0.02835      |       | sec   | 28.35 msec    |
| u      | /     | 1e-6   | micro   | 0.00047658   |       | sec   | 476.58 usec   |
| n      | /     | 1e-9   | nano    | 0.0000000032 |       | sec   | 3.20 nsec     |
| p      | /     | 1e-12  | pico    | 1.0 x 10^-10 |       | sec   | 100.00 psec   |

## The `${^stdout}` variable and the Reason string

In command-line probes, a specially formatted output string is used to define variables and their values. Normally, *anything else written to standard output is used as the reason string for the probe.*

If `${^stdout}` exists in the command-display section of a command-line probe, then anything written to standard output by the probe script is assigned as the value of the `${^stdout}` variable. This allows the script to programmatically define all or part of the contents of the lower part of the status window.

Using `${^stdout}` means that the reason string does not get defined. To compensate, and to allow the definition of a meaningful reason string, a convention was defined. If the specially-formatted output string mentioned above defines a variable named `reason`, then its value is assigned to the reason string used in the status window.

For example, output from the WMI Top Processes probe might look like this:

```
\{ProcessTime0:=100.0,CPU:=1.0,reason:="CPU utilization is
below 60%"}
\B5\WMI Top Processes\0P\
  \B4\CPU Utilization:\P0\    $CPU %
    \B4\wmiprvse(3924)\P0\    $ProcessTime0 %
```

# Using Persistent Variables

SNMP and Command-line probes can make use of persistent variables. A persistent variable is one that retains its value between polls.

## SNMP Probe Example

The example below demonstrates the use of persistent variables in an SNMP probe.

```
<!--
SNMP probe with persistent variables
(com.dartware.snmp.persistent)
Custom Probe for Intermapper (http://www.intermapper.com)
Please feel free to use this as a base for further
development.
Original version 24 November 2003 by reb.
Updated 29 July 2005 -
Updated 28 Oct 2005 - include display_name so it displays
properly in Intermapper 4.4 -reb
21 Apr 2006 - Changed to test conversion of $variables to a
condition string.
20 Sep 2011 - Updated to talk about variable persistence -reb
18 Oct 2011 - Minor editing polish -reb
-->

<header>
  "type"           = "custom-snmp"
  "package"        = "com.dartware"
  "probe_name"     = "snmp.persistent"
  "human_name"     = "SNMP Persistent Variables"
  "version"        = "1.1"
  "address_type"   = "IP,AT"
  "port_number"    = "161"
  "display_name"   = "Miscellaneous/Test/SNMP Persistent
Variables"
</header>

<snmp-device-properties>
  -- none required
</snmp-device-properties>

-- The <description> contains text that will be displayed in
the Set  Probe window.
-- Describe the probe as much as necessary so that people
will understand what it does and how it works.
<description>
\GB\SNMP Probe with Persistent Variables \P\
Sometimes probes need to compare variables from previous
```



```

invocations
o the current values. Intermapper SNMP probes can retain
variables from one invocation to the next.
This is done by setting a variable in this invocation to
preserve the value of the current calculation. This becomes
the "old variable" in the next invocation. The steps are:
- Read the new (current) value into a variable ("XYZ");
- Do the computations with it;
- Save that value in a separate variable ("oldXYZ");
- Re-run the probe. The oldXYZ will still contain its
previous value.
\b\Example Probe\p\
This probe reads ifInOctets for a specified interface and
computes the difference between it and the previous
ifInOctets. It also reads the sysUpTime.0, and computes the
time delta between the probe executions. From these values,
the probe computes the traffic rate. For comparison, the
probe also looks at ifInOctets using Intermapper's standard
PER-SECOND calculations.
</description>

-- Parameters are user-settable values that the probe uses
for its comparisons.
-- Specify the default values here. The customer can change
them and they will be retained for each device.
<parameters>
"Interface" = "24"
</parameters>

-- SNMP values to be retrieved from the device, and
-- Specify the variable name, its OID, a format (usually
DEFAULT) and a short description.
-- CALCULATION variables are computed from other values
already retrieved from the device.
<snmp-device-variables>
actInOctets, ifInOctets.$Interface, PER-SECOND, "actual
inOctets"
inOctets, ifInOctets.$Interface, INTEGER, "current
value"
deltaBytes, $inOctets-$oldInOctets, CALCULATION, ""
curSysUpTime, sysUpTime.0, INTEGER, "current sysuptime"
deltaTime, $curSysUpTime - $oldSysUpTime, CALCULATION, ""
-- Now update the oldXXXX variables with current values for
next time
-- NB: prevInOctets is only needed for display - it is not
used in
-- the calculations above
prevInOctets, $oldInOctets, CALCULATION, "from last time"
oldInOctets, $inOctets, CALCULATION, "old inOctets for

```

```
next time"
oldSysUpTime, $curSysUpTime, CALCULATION, "old sysuptime for
next time"
</snmp-device-variables>

-- Specify rules for setting the device into Alarm or Warning
state
<snmp-device-thresholds>
</snmp-device-thresholds>

-- The <snmp-device-display> section specifies the text that
will be appended
-- to the device's Status Window.
<snmp-device-display>
\B5\Persistent SNMP Variables on Interface=$interface\0P\
\4\Old Octets:\0\  $prevInOctets \3g\bytes\0mp\
\4\Cur Octets:\0\  $inOctets \3g\bytes\0mp\
\4\Delta's:\0\  $deltaBytes \3g\bytes\0mp\ in $deltaTime
\3g\centi-seconds \0mp\
\4\Computed Rate:\0\  ${eval:sprintf("%d",($deltaBytes) /
$deltaTime * 100) } \3g\bytes/sec\0mp\
\4\Actual Rate:\0\  ${scalable10: #.### : $actInOctets}
\3g\bytes/sec, using built-in PER-SECOND type\0mp\
</snmp-device-display>
```

## Command Line Probe Example

The example below demonstrates the use of persistent variables in a command-line probe.

```
<!--
Command-line Return (com.dartware.tool.persistent.txt)
Copyright© HelpSystems, LLC. All rights reserved.

Test of persistent variables by round-tripping values: pass
them into script, get results back, post to status window.
20 Sep 2011 -reb
18 Oct 2011 Minor editing -reb
-->

<header>
  type = "cmd-line"
  package = "com.dartware"
  probe_name = "tool.persistent"
  human_name = "Command Line Persistent Variables"
  version = "1.1"
  address_type = "IP"
  display_name = "Miscellaneous/Test/Command Line
Persistent Variables"
```

```

</header>

<description>
\GB\Command Line Persistent Variables\p\
This is an example of passing persistent variables into a
command-line probe. The attached Python script takes the
variables, updates them, and returns them to be used for the
next iteration.
The script below processes two variables: \b\${SearchString}\p\
and \b\${numericParam}\p\ . The script appends an "a" to
${SearchString}, and adds 1 to the ${numericParam} and returns
both values. (This is a useless script, created just to
demonstrate use of persistent variables.)
Both variables are uninitialized when the probe is executed
the very first time. The script can detect this startup
condition because an uninitialized variable is passed to the
script as the variable's name.
For example, the variable named ${SearchString} will be passed
as a string - "${SearchString}". The script can detect this
value - it's the same name that will be used to return the
new result for the variable - and treat the variable as
uninitialized, by assigning a sensible default value.
The Python script tests the passed-in values to see if they
match the expected name and initializes them accordingly.
\i\Note:\p\ This device's address should be set to
\i\localhost\p\ .
\i\Note:\p\ These variables could be initialized by setting
them in the <parameter> section, but this exposes a lot of
the script's internal variables to the customer: this is
generally not a good design.
</description>

<parameters>
-- no human-editable parameters
</parameters>

<command-line>
-- Unix/OSX: Empty path forces the Intermapper
Settings:Tools directory
path=""
cmd="${PYTHON} persistent.py ${numericParam}"
arg=' '
input="${SearchString}"
</command-line>

<command-exit>
-- These are the exit codes used by Nagios plugins
down:  ${EXIT_CODE}=4

```

```
critical:  ${EXIT_CODE}=3
alarm:    ${EXIT_CODE}=2
warn:     ${EXIT_CODE}=1
okay:     ${EXIT_CODE}=0
</command-exit>

<command-display>
\b5\ Current value of SearchString and numericParam\p0\
  Search String: $SearchString
  Number: $numericParam
</command-display>

<tool:persistent.py>
#!/usr/local/imdc/core/python/bin/imdc -OO
# The line above is required to run as an IMDC plugin
# persistent.py

# Round-trip the passed-in variables, update them, and return
them
# 20 Sep 2011 -reb
import os
import sys
import getopt

try:
    opts, args = getopt.getopt(sys.argv[1:], "")

except getopt.GetoptError, err:
    searchString = "getopt error %d" % (err)

if (args[0] == "$numericParam"): # check to see if the
argument is the name of the parameter
    number = 0      # if so, set it to a good initial value
else:
    number = int(args[0]) # otherwise, convert the string to
an integer

# Read the stdin file which contains the search String
f = sys.stdin      # open stdin
searchString = f.readline().strip() # get the line & remove
leading & trailing whitespace

if (searchString == "$SearchString"): # check to see if the
value is the name of the parameter
    searchString = ""      # if so, set it to a good initial
value

retcode=0
searchString = searchString + "a" # add another "a" to the
```

```
end of the string
number = number+1      # increment the number as well
retstring = "Hunky Dory!"

print "\{ $SearchString := '%s', $numericParam := '%d' } %s"
% (searchString, number, retstring)
sys.exit(retcode)
</tool>
```

# SNMP Probes

```
type="custom-snmp"
```

Using SNMP probes, you can monitor certain MIB variables that aren't tested by Intermapper's built-in probes. These MIB variables might include the CPU utilization of a router, temperature inside the equipment, switch closures, etc.

Like other probes, SNMP probes are invoked and return the status and condition string for the device being tested. Here's a summary of the operational flow of an SNMP probe:

1. Intermapper polls the device for the values (called *probe variables*) specified in the probe file as well as the device's built-in MIB variables (usually byte and packet rates for interfaces).
2. Intermapper also polls each interface for probe variables as needed.
3. Intermapper then evaluates a series of expressions in the probe file, comparing the probe variables to thresholds.
4. If a comparison is triggered (generally, if the probe variable is above or below the given threshold), then Intermapper sets the device status as specified in the probe, if it is "worse" than the device's current status.
5. When the user clicks and holds on a device, Intermapper processes the relevant *display* section to produce the text for the Status window.

## Common Sections of an SNMP probe

SNMP probes follow the same general format as other probe files.

- The [<header>](#) section of a command-line probe specifies the probe type, name, and a number of other properties fundamental to the operation of the probe.
- The [<description>](#) section specifies the help text that appears in the Set Probe window. Format the description using IMML, [Intermapper's Markup language](#).
- The [<parameters>](#) section defines the fields presented to the user in the Set Probe window.

## Sections Specific to SNMP Probes

```
<header>
  type = "custom-snmp"
</header>
```

**Note:** See [SNMP Trap Probes](#) for information on creating probes that handle SNMP traps.

Each SNMP probe also has:

- [An <snmp-device-variables> section](#) - Specifies which MIB variables to collect from the device
- [An <snmp-device-thresholds> section](#) - Specifies how those variables are to be tested against thresholds to determine the device's status
- [An <snmp-device-display> section](#) - Specifies what information about the device and its links should be displayed in the Status window
- [The <snmp-device-properties> section](#) - Specifies certain aspects of the SNMP queries sent to the device.
- [The <snmp-device-variables-ondemand> Section](#) - Many devices store information in SNMP tables. Intermapper can retrieve this tabular information and display the data "on demand".

# The <snmp-device-variables> Section

Use the <snmp-device-variables> section to specify the values you want to retrieve using a particular SNMP OID. These values, called *probe variables*, can then be compared to thresholds to create alarms, warnings, etc.

Each line of the <snmp-device-variables> section defines a particular variable to be retrieved. The definition is composed of four comma-separated attributes:

```
[Variable-name], [OID], [Type], [Chart-legend]
```

## Sample <snmp-device-variables> Section

```
<snmp-device-variables>
--Variable-name  OID --- TYPE ---- CHART LEGEND -----
ipForwDatagrams, 1.3.6.1.2.1.4.6.0, PER-SECOND, "Forwarded
datagrams"
ipInHdrErrors,   1.3.6.1.2.1.4.4.0, PER-MINUTE, "IP
received header err"
tcpCurrEstab,    1.3.6.1.2.1.6.9.0, DEFAULT,      "Number of
TCP conn's"
sysDescr,        1.3.6.1.2.1.1.1.0, DEFAULT
</snmp-device-variables>
```

**Note:** The OIDs above have a trailing ".0" to specify their full OID.

## Status Window Text - The <snmp-device-display> Section

Use the <snmp-device-display> to control the way information gathered during polling appears in the Status window. Create a <snmp-device-display> section with the items to be displayed. For more information, see [Customized Status Windows](#).

Intermapper retrieves MIB variables from a device and then tests them against thresholds. The <snmp-device-variables> section defines the OIDs of MIB variables that are to be retrieved. These values are called *probe variables* and can then be compared to thresholds to create alarms, warnings, etc.

Each line of the <snmp-device-variables> section defines a particular variable to be retrieved. The definition is composed of four comma-separated attributes:



```
[VariableName], [OID], [Type], [Chart Legend]
```

The definitions of these attributes are:

- **VariableName** is the name used to represent the particular MIB value in this probe. For more information, see the [Built-in Variable Reference](#) topic.
- **OID** is the SNMP Object ID for the particular probe variable. The OID can be expressed as a string of dotted numbers or as an OID name, if the corresponding MIB has been imported into Intermapper. An OID can also be an expression, if the type is "CALCULATION" ([see note below](#)).
- **Type**- specifies how Intermapper displays a value. The type may be one of the following:
  - **Default** - Intermapper deduces an applicable type from the SNMP type of the variable and displays it according to the "Format for DEFAULT types" table below.
  - **Integer** - values are coerced to a numeric value. If you have a string value returned "78Fred", the value as INTEGER is 78.
  - **Integer64** - values are coerced to a numeric value (up to 64-bits). If you have a string value returned "78Fred", the value as INTEGER is 78.
  - **Hexadecimal** - If the value is a number, it is displayed as a hex number, preceded by "0x" (0xFFFFFFFF). Otherwise, it is represented as a series of hex characters separated by spaces (44 61 72 77 69 6E 20 52 69 63 68 61 72). *This type is not chartable.*
  - **Hexnumber** - converts a string of hexadecimal digits into a number. For example, a string value of "FE" is converted to the number 254.
  - **Total-value** - displays the actual value of a counter or gauge, not a computed rate value. This is always an unsigned number.
  - **Total64-value** - displays the actual value of a counter or gauge, not a computed rate value. This is always an unsigned number (up to 64-bits).
  - **Per-second** and **Per-minute** - force Intermapper to compute a rate for the particular variable by computing the difference between two successive samples and dividing by the elapsed time.
  - **String** - sets a variable to the text string that corresponds to this OID's enumerated type, as defined in the MIB. (see [Enumerated Values](#) below) *This type is not chartable.*
  - **Calculation** - sets the variable to the result of the calculation shown in the OID field.
  - **TrapVariable** - sets a variable based on the value received from an SNMP trap. A complete discussion of Trap Variables is available in [About Custom SNMP Trap Probes](#)
  - **IPADDRESS** - Intermapper displays a 4-byte octet string as an IPv4 address and a 16-byte octet string as an IPv6 address.

**Format for DEFAULT types** - All SNMP variables have an inherent type

---

(one of the choices in the Type column below.) If a probe variable is declared as DEFAULT, Intermapper displays it according to this table:

| Type                 | Displayed as:   |
|----------------------|---|
| Counter32, Counter64 | Per-Second  |
| Unsigned32 (Gauge)   | Total-Value   |
| Integer              | Integer   |
| OctetString          | String (if 1st digit printable)<br>Hexadecimal (if 1st digit not printable) |
| Object ID            | String  |
| IPAddress            | String  |
| TimeTicks            | String  |

- **Chart-legend** is an optional field that provides a text label to be placed on any strip charts that incorporate this variable. Chart legends may contain embedded variable names in the form \$VariableName.

#### Notes:

- Calculation variables have a slightly different form, as described below.
- See [Probe Calculations](#) for a description of the functions and operators that are available in expressions.
- See [Using Persistent Variables](#) for information on how to save variable values between device polls.
- A scalar's OID must end in ".0" according to the SNMP specifications. See [SNMP OIDs](#) for a description of allowable formats for OIDs.
- See [On-Demand SNMP Tables](#) for a description of how your probe can display tabular information from a MIB.
- When Intermapper retrieves a value, by default it issues an SNMP Get-Next-Request for the previous OID, unless the pduType is set to "get-request" (see [Probe Properties](#).)

Here is a sample <snmp-device-variables> section.

```
<snmp-device-variables>
  --Variable-name  OID ---          TYPE ----      CHART
LEGEND -----
  sysDescr,        1.3.6.1.2.1.1.1.0,  DEFAULT,
  sysLocation,     sysLocation.0,      DEFAULT,
  ipInHdrErrors,   1.3.6.1.2.1.4.4.0,  PER-MINUTE,  "IP
received header err"
```

```
    ifInOctets1,      ifInOctets.1,      DEFAULT,  
"bytes/sec received on interface 1"  
    ifInOctets2,      ifInOctets.$if,      DEFAULT,  
"bytes/sec received on interface ${if}"  
    TempF,            ($TempC * 1.8)+32, CALCULATION, "Degrees  
F"  
</snmp-device-variables>
```

- **\$sysDescr** is set by retrieving the OID 1.3.6.1.2.1.1.1.0 and using that value. It will be displayed as the default format, that is a string.
- **\$sysLocation** is set by querying the OID sysLocation.0 (which is equivalent to the numeric 1.3.6.1.2.1.1.6.0). It, too, will be displayed as a string. Note that you can use a human-readable SNMP variable name instead of its numeric OID.
- **\$ipInHdrErrors** is set by querying the OID 1.3.6.1.2.1.4.4.0, but will be displayed as a number of errors \*per minute\*.
- **\$ifInOctets1** is set by querying the OID ifInOctets.1 (1.3.6.1.2.1.2.2.1.10.1). Note that the final digit is "1", indicating that it is reading row 1 of the values in the table. It will be displayed as a number of octets (byte) per second, since Intermapper's default display format for a counter is "per-second".

The next two examples are controlled by variables that have been set elsewhere, perhaps set manually in the <parameters> section of the probe.

- **\$ifInOctets2** is set by first evaluating the variable `$if`, then substituting that value into the OID. If `$if` were set to "1", then `$ifInOctets` would retrieve `ifInOctets.1`, and would result in the same value as `$ifInOctets1`. Note that `$if` is also used in the variable's legend.
- **\$TempF** is a "calculation" variable. It is set by evaluating the expression `($TempC * 1.8)+32` where `$TempC` was set elsewhere.

## SNMP Scalar and Table Values

SNMP has two kinds of values: *table* and *scalar* values.

- A **table** value is an element of a table: the variable name (e.g. `ifInOctets`) is the name of a column; the final digit (or digits) is the index for the element; it defines the table row containing the element. Thus `ifInOctets.1` is the full OID for the value in the first row of the `ifInOctets` column.
- A **scalar** value is "one of a kind" - there is only a single value, so you must specify a ".0" after the name to indicate that it's the only "row". For example, `sysDescr` can be represented as `1.3.6.1.2.1.1.1.0` or `sysDescr.0`. Both OIDs end in ".0".

## Using Variables in OIDs and Legends

You can use SNMP variables in OIDs and legends. The example below uses `$if` as the OID index, and displays it in the legend.

```
ifInOctets2, ifInOctets.$if, DEFAULT, "bytes/sec received on  
interface ${if}"
```

## Calculation Variables

A *Calculation* type variable receives the result of an arithmetic expression. After all variables have been polled, Intermapper calculates the expression, and sets the value of its variable to the result. In the example above:

```
TempF, ($TempC * 1.8)+32, CALCULATION, "Degrees F"
```

The variable "TempF" is set to the value of the expression `(10 * sin(0.01 * time()))`. This gives a sine wave that makes an attractive chartable value. Use "\$SineValue" to refer to the variable elsewhere in the probe.

## Built-in Variables

Intermapper provides a number of built-in variables, detailed in the [Built-in Variable Reference](#) topic.

## Macros

Intermapper also supports several macros that can help control the output of variables, as well as their use in charts.

- `${chartable[:fmt]: expr}`
- `${variablename:legend}`
- `${eval: expr}`
- `${scalable10}` and `${scalable2}`

These are detailed in the Built-in Variable Reference topic's [Macros section](#).

## Enumerated Values

Many MIBs use an integer to represent one of several states. For example, `ifOperStatus` (1.3.6.1.2.1.2.2.1.8.x) is defined in MIB-II as:

```
INTEGER { up(1), down(2), testing(3) }
```

This means that the value 1 represents the "up" condition; 2 represents "down"; and 3 represents "testing".

The type you use when you define the variable affects the result:

- If you define a variable to retrieve this value as INTEGER or DEFAULT, the probe displays the value as a number.
- If you define it as a STRING, the probe uses the MIB to find the string representation, and sets the variable (in this case) to the value "up", "down", or "testing".

-- If the MIB has been imported, the string is displayed in the output if the variable is declared as STRING.

```
variable1, ifOperStatus.3, STRING, ""
```

-- The integer value is always used in the output if the variable is declared as DEFAULT or INTEGER.

```
variable2, ifOperStatus.3, DEFAULT, ""  
variable3, ifOperStatus.3, INTEGER, ""
```

If the OID or MIB name isn't defined (because the corresponding MIB hasn't been imported or because of a typo), the probe displays the integer value.

## Alternatives to Enumerated Values

If no MIB file is available, you can create a calculation variable to select a string based on the numeric value returned.

### Example: Two choices

```
-- If you have two choices, use a conditional expression:  
xxxx ? yyyy : zzzz  
  
-- It can be read as:  
if xxxx is true then  
    return yyyy  
otherwise  
    return zzzz  
  
-- The variable looks like this:  
xxxxStr, ($xxxx == 0 ? "yyyy" : "zzzz"), CALCULATION,  
"replacement string for $xxxx"
```

### Example: Three or more choices

```
-- Chain the expression:  
aaaa ? bbbb : cccc ? dddd : eeee ? ffff : gggg
```

```
-- Can be read as:
if aaaa is true then
    return bbbb
else if cccc is true
    return dddd
else if eeee is true
    return ffff
else return gggg

-- Generally, aaaa, cccc, and eeee test to see if a single
variable is equal to 1, 2, 3, etc.

-- The calculation variable then looks like this:
aaaaStr, ($aaaa==0 ? "bbbb" : $aaaa==1 ? "dddd" : $aaaa==2
? "ffff" : "gggg"), CALCULATION, "replacement string for
aaaa"
```

# The <snmp-device-thresholds> Section

Use the <snmp-device-thresholds> section to specify the comparisons that should be made between probe variables and other values.

Each line in the threshold section contains a *status*, a *comparison*, and an optional *condition string* for probe variables. If the comparison is *triggered*, (if the expression comparing the probe variable to a constant or other variable is true) then the device is changed to the corresponding status (if that exceeds its current status.)

A threshold can be one of the following (they are case-sensitive), and should be presented in this order:

- down
- critical
- alarm
- warning
- okay

## Sample <snmp-device-threshold> Section

```
<snmp-device-thresholds>
  down: ${ifOperStatus}    = 0 "Device Down"
  critical: ${ipInHdrErrors} > 15 "ipInHdrErrors
critical"
    alarm: ${ipForwDatagrams} > 10 "ipForwarded datagrams
too high"
    alarm: ${tcpCurrEstab}   >= 1
    alarm: ${ipInHdrErrors} > 10 "ipInHdrErrors too
high"
  warning: ${ipForwDatagrams} > 5
  warning: ${ipForwDatagrams} <= 2
  warning: ${ipInHdrErrors} > 5
  okay: 1 = 1 "Everything is OK"
</snmp-device-thresholds>
```

## Creating Comparisons

As implied above, comparisons are evaluated *in order from top to bottom until a comparison is triggered (result is true)*. It is important to put the Critical comparisons first, followed by Alarm, Warning, and OK.

At that point, if the associated status is more severe than the device's current status, the device now uses its status and condition. No further comparisons are made once one has triggered.

When a comparison is triggered, it is written to the log file as well as being added to the bottom of the device's Status window. If the condition string is present, it is displayed in addition to the comparison string.

## Numeric Comparisons

The following numeric comparison operators are legal:

>, >=, <, <=, =, and !=.

## String Matches

By default, Intermapper performs numeric comparisons.

**To compare values as strings:**

- Enclose one or both of the operands in double-quotes ("). For example, the comparison

```
warning: ${sysContact} != "Fred Flintstone"
```

performs a string comparison because the name is enclosed in quotes.

- Use the =~ and !~ operators to provide partial string matches. They perform "contains" and "doesn't contain" comparisons, respectively.



# The <snmp-device-properties> Section

The <snmp-device-properties> section specifies certain aspects of the SNMP queries sent to the device. Like other sections, it is closed with a </snmp-device-properties> tag. For example:

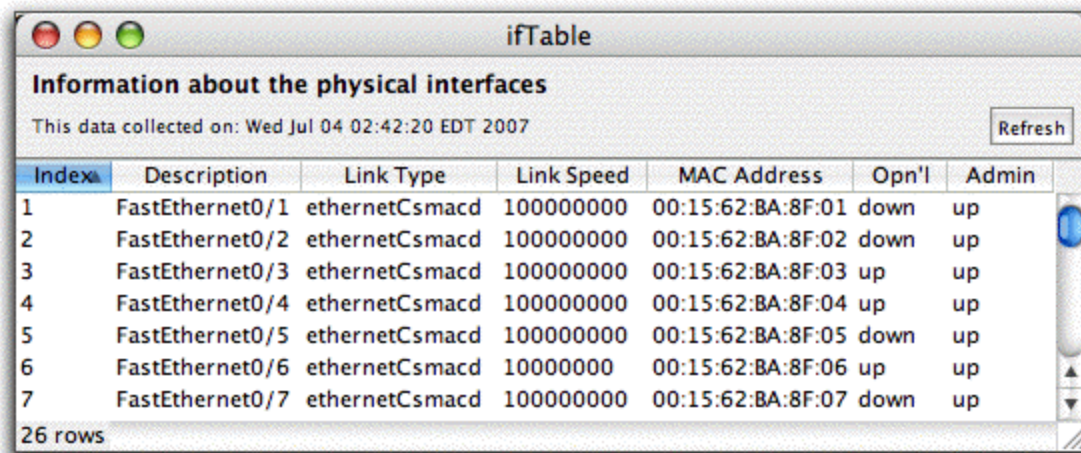
```
<snmp-device-properties>
  nomib2          = "true"
  pdutype         = "get-request"
  apcups          = "false"
  maxvars         = "10"
  interface_numbered = ($ifIndex == 2 or $ifDescr =~ "en2")
  interface_visible  = ($ifIndex == 2 or $ifDescr =~ "en2")
</snmp-device-properties>
```

The properties that may be set include:

- **nomib2="true"** -- Intermapper does not query the sysUptime MIB-2 variable.
- **pdutype="get-request"** -- Intermapper uses SNMP Get-Request, instead of Get-Next-Request queries.
- **apcups="false"** -- If apcups is false, Intermapper will not query the APC-UPS MIB even for devices that auto-detect as one.
- **maxvars="10"** -- maxvars controls the maximum number of variables to put in each SNMP request. If a custom probe requires more variables than maxvars, Intermapper sends multiple queries containing up to maxvars variables.
- **interface\_visible = <expression>** - specifies a "filter expression" for use in determining which interfaces are made visible. By default, Intermapper makes the numbered interfaces visible. Setting this property allows you to make certain *unnumbered* interfaces visible if they match the expression that can use \$ifIndex, \$ifDescr, \$ifType, or \$ifAlias variables.  
**Note:** This property does not allow you to make numbered interfaces hidden.
- **interface\_numbered = <expression>** - specifies a "search expression" for use in determining which interface is made numbered. By default, the ipAddrTable specifies which interface is numbered. This property allows the probe file to override that choice.

# The <snmp-device-variables-ondemand> Section

Many devices store information in SNMP tables. Intermapper can retrieve this tabular information and display the data "on demand", that is, when requested by user action. When you view a table, Intermapper retrieves the information from the device immediately and displays it in a separate window. The information in an on-demand window is not part of the regular polling cycle, nor is it refreshed until you specifically request it. The image below shows a sample on-demand window.



The screenshot shows a window titled "ifTable" with the subtitle "Information about the physical interfaces". Below the subtitle, it says "This data collected on: Wed Jul 04 02:42:20 EDT 2007" and there is a "Refresh" button. The table has 7 columns: Index, Description, Link Type, Link Speed, MAC Address, Opn'l, and Admin. The data is as follows:

| Index | Description     | Link Type      | Link Speed | MAC Address       | Opn'l | Admin |
|-------|-----------------|----------------|------------|-------------------|-------|-------|
| 1     | FastEthernet0/1 | ethernetCsmacd | 1000000000 | 00:15:62:BA:8F:01 | down  | up    |
| 2     | FastEthernet0/2 | ethernetCsmacd | 1000000000 | 00:15:62:BA:8F:02 | down  | up    |
| 3     | FastEthernet0/3 | ethernetCsmacd | 1000000000 | 00:15:62:BA:8F:03 | up    | up    |
| 4     | FastEthernet0/4 | ethernetCsmacd | 1000000000 | 00:15:62:BA:8F:04 | up    | up    |
| 5     | FastEthernet0/5 | ethernetCsmacd | 1000000000 | 00:15:62:BA:8F:05 | down  | up    |
| 6     | FastEthernet0/6 | ethernetCsmacd | 1000000000 | 00:15:62:BA:8F:06 | up    | up    |
| 7     | FastEthernet0/7 | ethernetCsmacd | 1000000000 | 00:15:62:BA:8F:07 | down  | up    |

At the bottom left of the table, it says "26 rows".

On-demand tables are useful for digging down into a device when you suspect there might be a problem. You can create on-demand tables to view a routing table, ARP table, or other statistics that are kept within tables.

## Background on SNMP Tables

The SNMP protocol provides access to two types of variables:

- **Scalar variables** contain single values such as strings (that could represent system description or firmware version), integers (number of interfaces), counters (number of errors), gauges (CPU temperature and memory utilization), etc.
- **Table variables** hold information about similar entities within a device. These entities could be interfaces in a router or switch, users associated with a wireless access point, virtual machines in a server, etc. Each entity's information is represented by a row, whose columns are variables (which are themselves scalars) that hold information about the entity. A row is often called an "entry" in a MIB; each column is specified by an OID prefix plus a unique index that specifies a particular row.

For example, MIB-II defines a table named "ifTable" that gives information about a device's interfaces. An outline of ifTable looks like this:

```
+ ifTable
+ ifEntry [ifIndex]
- ifIndex      "Interface Index"
- ifDescr      "Description"
- ifType        "Link type"
- ifSpeed       "Link speed"
- ifPhysAddress "MAC Address"
- ifOperStatus "Operational status"
- ifAdminStatus "Administrative status"
- ... and so on...
```

Here's how to interpret this information. The *ifTable* is composed of a sequence of *ifEntries* which form the rows of the table. Each row (each *ifEntry*) has a number of variables: we show only some of them, starting with ifIndex and ending with ifAdminStatus. These variables become the columns of each row.

The image above shows the window of an on-demand table for ifTable. The columns match the variables mentioned above. The window also shows the number of rows in the table (at lower left), the time the data was retrieved, and the Refresh button to retrieve current data.

## Table Indexes

Each row of an SNMP table has a unique index. The index for ifTable is the "interface index", that loosely represents the port number of the interface. Individual values are represented by the column name followed by its index. For example:

```
ifSpeed.3 (or the OID 1.3.6.1.2.1.2.2.1.3)
```

would represent the ifSpeed for row 3 of the table. The "column name" is ifSpeed (1.3.6.1.2.1.2.2.1), and the index is the ".3".

## Table Syntax

An on-demand table in a custom SNMP probe mirrors the outline above. Its definition consists of a sequence of lines of comma-separated values defining the variables of one or more tables:

```
<snmp-device-variables-ondemand>
  ifTable, .1, TABLE,
  "Information about the physical interfaces"
  ifTable/ifIndex, 1.3.6.1.2.1.2.2.1.1, DEFAULT,
  "Interface Index" <!-- using OID for column -->
  ifTable/ifDescr, 1.3.6.1.2.1.2.2.1.2, DEFAULT,
```

```

"Description"      <!-- using OID for column -->
    ifTable/ifType,      1.3.6.1.2.1.2.2.1.3, STRING,
"Link Type  "      <!-- using OID for column -->
    ifTable/ifSpeed,    1.3.6.1.2.1.2.2.1.5, DEFAULT,
"Link Speed"       <!-- using OID for column -->
    ifTable/ifPhysAddress, ifPhysAddress,      HEXADECIMAL,
"MAC Address"      <!-- using column name from MIB -->
    ifTable/ifOperStatus, ifOperStatus,        STRING,
"Opn'l"           <!-- using column name from MIB -->
    ifTable/ifAdminStatus, ifAdminStatus,      DEFAULT,
"Admin"           <!-- using column name from MIB -->
</snmp-device-variables-ondemand>

```

This example shows an on-demand table for ifTable.

**Note:** The `<snmp-device-variables-ondemand>` section must contain fewer than 50 queries.

The remainder of the table is composed of comma-separated lines describing each variable.

- The first line creates a table. The first field is the table's name that can be used to represent the table elsewhere in the probe file. The second field should be ".1". The third field ("TABLE") indicates that this is a new table. The fourth field is a human-readable description that is displayed in the on-demand window.
- The remaining lines follow the format of `<snmp-device-variables>`. See their page for more information. The names in the first column contain the table name, a "/", and the name for the column.
- The next four lines define variables (ifIndex, ifDescr, ifType and ifSpeed) that are to be columns of the table. They are defined using the numeric OID that represents the column for those values.
- The final three lines define ifPhysAddress, ifOperStatus, and ifAdminStatus. They are defined using the name of the column from the MIB. This is entirely equivalent to writing out the full numeric OID.

The tables described here are available as the **SNMP/Table Viewer** probe that's built into Intermapper. In addition, the probe file is available on the Table Viewer page of this manual.

## Augmenting Tables

Certain MIB's define a table that "augments" another table. This simply means that the augmenting table uses the same index variables as another table. Since the index variables are the same, you can visualize this as adding columns to an existing table.

For example, in the IF-MIB, the ifXTable augments ifTable, providing a number of useful additions.

Intermapper's table syntax easily supports mixing columns from one or more tables that share the same table definition:

```
<snmp-device-variables-ondemand>
  ifXTable, .1, TABLE,
"Extended ifTable"
  ifXTable/ifIndex, IF-MIB::ifIndex, DEFAULT,
"Interface index"
  ifXTable/ifDescr, IF-MIB::ifDescr, DEFAULT,
"Description"
  ifXTable/ifName, IF-MIB::ifName, DEFAULT,
"Name" <!-- ifXTable -->
  ifXTable/ifAlias, IF-MIB::ifAlias, DEFAULT,
"Alias" <!-- ifXTable -->
  ifXTable/ifType, IF-MIB::ifType, STRING,
"Link Type "
  ifXTable/ifSpeed, IF-MIB::ifSpeed, DEFAULT,
"Link Speed"
  ifXTable/ifHighSpeed, IF-MIB::ifHighSpeed, DEFAULT,
"Mbit/sec"
  ifXTable/ifPhysAddress, IF-MIB::ifPhysAddress, HEXADECIMAL,
"MAC Address "
  ifXTable/ifOperStatus, IF-MIB::ifOperStatus, STRING,
"Opn'l"
  ifXTable/ifAdminStatus, IF-MIB::ifAdminStatus, DEFAULT,
"Admin"
</snmp-device-variables-ondemand>
```

In the example, ifName and ifAlias come from the ifXTable while the others are part of ifTable. Yet they all can be shown in the same on-demand window.

## Index-Derived Variables

Certain SNMP equipment uses the value of one of more columns as part of the row index. In many cases, the column itself is not accessible, and thus cannot be queried directly.

Intermapper has a facility that allows you to derive the values of these columns from the index itself, even from columns that are not accessible. The notation `oid[a:b]` means to fetch the OID `oid` and compute the value from the index:

```
oid[a:b] - remove the subid's for "oid" then start with the
a'th subid and collect b subids.
oid[a:] - remove the subid's for "oid" then start with the
a'th subid and collect the remaining subids
```

Here is an example of retrieving the four columns of ipNetToMediaTable. Note that the table has been given the name "ARPTTable", although the OID is ipNetToMediaEntry.

```
<snmp-device-variables-ondemand>
  ARPTable,
  TABLE,      "Map from IP addresses to physical addresses."
  ARPTable/ipNetToMediaIfIndex,      ipNetToMediaType[0:1],
  DEFAULT,     "Interface index"
  ARPTable/ipNetToMediaNetAddress,    ipNetToMediaType[1:4],
  DEFAULT,     "IP Address"
  ARPTable/ipNetToMediaPhysAddress,   ipNetToMediaPhysAddress,
  HEXADECIMAL, "MAC Address"
  ARPTable/ipNetToMediaType,          ipNetToMediaType,
  STRING,     "Type"
</snmp-device-variables-ondemand>
```

The `ipNetToMediaTable` is defined to use two index values:

- `ipNetToMediaIndex` (the row number of the interface)
- `ipNetToMediaNetAddress` (the IP address of the device)

The full OID used to retrieve a value from the table is its prefix (for example, `ipNetToMediaType` is 1.3.6.1.2.1.4.22.1.4) followed by a single subid for `ipNetToMediaIndex` followed by the four subid's of `ipNetToMediaNetAddress`.

When Intermapper displays the table, it retrieves `ipNetToMediaType`, removes the prefix, then starts at position 0 of the remainder and uses one subid for the `ipNetToMediaIfIndex`, and then starts at position 1 and takes the next four subid's for the value of `ipNetToMediaNetAddress`.

## Calculations within On-demand Tables

Intermapper provides the ability to have calculations in on-demand tables. This is useful for making calculations from values within the same row of the table. The calculations may also use constant values as well as parameters to the probe. In the example below:

- 1) declares the column "ifIndex"
- 2) calculates the value of (column "ifIndex" plus 1) times 2
- 3) uses the previous column to get the original ifIndex back
- 4) and 5) display the current value of ifInOctets and ifOutOctets
- 6) the calculated ratio between these two columns
- 7) and 8) show a circular reference which fails gracefully - Circular1 and Circular2 refer to each other and simply display "-" as a value.

```
<snmp-device-variables-ondemand>
  ifTableTest,
  TABLE
  ifTableTest/ifIndex,          IF-MIB::ifIndex,
  DEFAULT      <!-- #1 -->
  ifTableTest/ifIndexPlus1Times2, ($ifIndex + 1)*2,
```

```
CALCULATION <!-- #2 -->
    ifTableTest/ifIndexBack,          $ifIndexPlus1Times2/2-1,
CALCULATION <!-- #3 -->
    ifTableTest/TInOctets,            IF-MIB::ifInOctets,
DEFAULT <!-- #4 -->
    ifTableTest/TOutOctets,           IF-MIB::ifOutOctets,
DEFAULT <!-- #5 -->
    ifTableTest/ifRatio,              $TInOctets/$TOutOctets,
CALCULATION <!-- #6 -->
    ifTableTest/Circular1,            $Circular2 - 1,
CALCULATION <!-- #7 -->
    ifTableTest/Circular2,            $Circular1 + 1,
CALCULATION <!-- #8 -->
</snmp-device-variables-ondemand>
```

## Displaying On-demand Tables

After you define a TABLE variable in the "ondemand" section of the probe, you can specify that status window should display a link to the ondemand window. To do this, add the variable name to the <snmp-device-display> section of the probe:

```
<snmp-device-display>
...
$ARPTable
</snmp-device-display>
```

The status window displays the table name as a hyperlink. Clicking on the hyperlink opens the on-demand table window shown at the top of the page.

If you want to replace the default table name displayed with your own text, you can specify the "alternate text" in the expanded variable form:

```
<snmp-device-display>
...
${ARPTable:View the entire ARP Table}
</snmp-device-display>
```

## Limitations

- On-demand variables must be in table form with a "/" in the variable name.
- You cannot query tables in the regular <snmp-device-variables> section.
- You cannot reference on-demand tables defined in other probes.
- You cannot specify non-accessible MIB variables by their symbolic OID. Instead, use the "derived values" syntax to determine the correct index-derived OID expression.

- You must declare no more than 50 variables in the <snmp-device-variables-ondemand> section or the query will not work.



# The <snmp-device-display> Section

## Controlling the Status window in SNMP Probes with <snmp-device-display>

Use the optional <snmp-device-display> section to describe the text that appears in a custom SNMP probe's Status window. Probe variables are replaced with their values in the Status window's text.

The default font for the Status window's text is a mono-spaced font, so alignment of text is straightforward. You can change the appearance of the text in the Status window using IMML, [Intermapper's Markup Language](#).

Here is a sample <snmp-device-display> section. Note that the variables are replaced with the values retrieved from the device, and that formatting is controlled by IMML.

```
<snmp-device-display>
  \B5\Custom SNMP Probe\0P\
  \4\ipForwDatagrams:\0\  ${ipForwDatagrams} datagrams/sec
  \4\ipInHdrErrors:\0\    ${ipInHdrErrors} errors/minute
  \4\tcpCurrEstab:\0\     ${tcpCurrEstab} connections
</snmp-device-display>
```

## Using Disclosure Widgets

A disclosure widget, also called a disclosure control, is a user interface element that allows the user to expand or collapse text in a window.

The basic syntax of a disclosure control area is as follows:

```
\#hide:[disclosureblock name]\ Title for Disclosure Block
\#begin:[disclosureblock name]\
  First line of disclosureblock
  Second line of disclosureblock
  Third line of disclosureblock
  Fourth line of disclosureblock
\#end:[disclosureblock name]\
```

Use the #hide and #show to specify the default state of the block.

You can also nest disclosure controls, as demonstrated below.

```
<!--
  Testing Disclosure Widgets(com.dartware.reb.expander.txt)
  Probe for Intermapper (http://www.intermapper.com)
  Please feel free to use this as a base for further
  development.

  Original version - 6 Aug 2010 -reb
-->

<header>
  "type" = "custom-snmp"
  "package" = "com.dartware"
  "probe_name" = "reb.expander_control"
  "human_name" = "Test Expander Control"
  "version" = "1.0"
  "address_type" = "IP,AT"
  "port_number" = "161"
  display_name = "Miscellaneous/Test/Test Expander Controls"
</header>

<snmp-device-properties>
  -- none required
</snmp-device-properties>

<description>
  \GB\Testing Disclosure Widgets\P\

  This probes is for testing out the disclosure widget
  ("expander control") feature
  in Status Windows.
</description>

<parameters>
  -- none
</parameters>

<snmp-device-variables>
  -- none
</snmp-device-variables>

<snmp-device-thresholds>
  -- none
</snmp-device-thresholds>

<snmp-device-display>
  -- The <snmp-device-display> section specifies the text
```

```
that will be appended
-- to the device's Status Window.
\B5\Displaying Expander_ Controls\OP\

\#hide:expander_1\ Title for Expander_1 (initially hidden)
\#begin:expander_1\
  First line of Expander_1
  Second line of Expander_1
  Third line of Expander_1
  Fourth line of Expander_1
\#end:expander_1\
\#show:expander_2\ Title for Expander_2 (initially shown)
\#begin:expander_2\
  First line of Expander_2
  Second line of Expander_2
  Third line of Expander_2
  Fourth line of Expander_2
  \#hide:expander_3\ Expander_3 nested within Expander_2
\#begin:expander_3\
  First line of Expander_3
  Second line of Expander_3
\#end:expander_3\
\#end:expander_2\
\#hide:expander_4\ Title for Expander_4 (controlled by
Expander_1) \#begin:expander_1\
  First line of Expander_4
  Second line of Expander_4
  Third line of Expander_4
  Fourth line of Expander_4
\#end:expander_1\

</snmp-device-display>
```

# The <snmp-device-alarmpoints> Section

Intermapper can monitor multiple conditions within a single device (e.g., a single piece of hardware) and give separate, independent notifications for each. For example, it can send notifications for a high temperature alarm independent of a low-memory condition in the same device.

Each of these conditions is called an "alarm point". Intermapper's custom SNMP probe facility allows you to define multiple alarm points for a device, along with their thresholds and the notifications to be sent.

**Note:** Alarm Point probes are typically customized to a particular purpose, which specifies both the conditions under which alerts are sent and the notifiers to which they are sent. To send an alert using a particular notifier, you must:

- Edit the <snmp-device-alarmpoints> section of probe containing the alarm points as described in [Alarm Point Format](#), and
- Enter the name of the notifier in the <snmp-device-notifiers> section as described in [Alarm Point Notifiers](#).

In probes that contain them, all Alarm Points are sent to the "Default Sounds" notifier by default.

Intermapper tracks the state of each alarm point separately. Alarms on one point will not affect the status, logging, or notifications of any other alarm point. However, the visual appearance of a device will reflect the most serious condition of any of its contained alarm points.

Alarm points have the following five severities. Each severity is assigned a color for quick visual identification.

| Severity | Color        | Description                                       |
|----------|--------------|---|
| Clear    | Green        | Nothing exceptional to report                     |
| Minor    | Yellow       | Device has departed from its normal "clear" state |
| Major    | Orange       | Device's operation is significantly affected      |
| Critical | Solid Red    | Device's operation is seriously degraded          |
| Down     | Blinking red | Device is unresponsive, actual state is not known |

Every time an alarm point changes from one severity to another:

- The the new condition is logged to the log file.
- A notification is sent using the existing Intermapper notifiers, including sounds, e-mail, paging modem or SNPP, or running scripts.

## Alarm Points - What the User Sees

Intermapper displays devices with alarm points much the way it shows "regular" devices. A device's icon is colored according to the most serious condition of all its alarm points.

Note that these colors correspond closely with Intermapper's OK/Warning/Alarm/Down coloring. The Critical state is new, and gets a solid red color to indicate that it's "worse" than the orange Alarm or Major severity.

A device's icon takes on the color of its most serious alarm point. For example, a device with two alarm points, one in Critical and one in Minor severity is colored a solid red.

## Acknowledgments

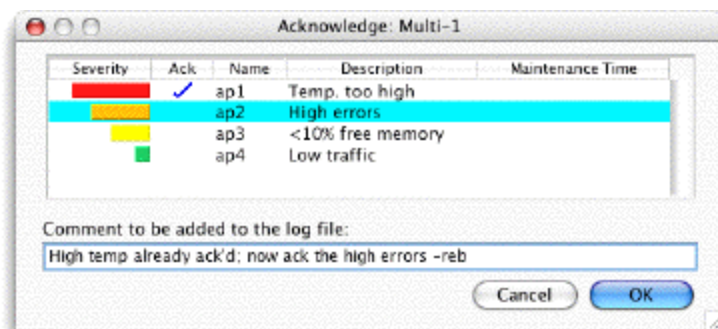
Acknowledging an alarm lets the operator indicate that they know about a problem and that they are working on it. An acknowledgment blocks further notifications for that alarm, and colors the icon blue to show that, although the problem remains, someone has taken responsibility for it.

The blue-acknowledged color makes it easy to see new problems at a glance. When all icons are green (working properly) or blue (in alarm, but being worked on) any new alarm appears as a yellow, orange, or red icon.

Alarm points can be acknowledged independently. That is, acknowledging one alarm point does not affect the state of other alarm points. Acknowledging an alarm point leaves the device's color set to its most serious un-acknowledged alarm point. When all alarm points have been acknowledged, the device icon turns blue.

The Acknowledge window for devices with alarm points will look much like the current Acknowledge window, with these differences:

- When acknowledging one device, the Acknowledge window will display a list of the alarm points, sorted in order of severity.
- The operator may select one, many, or all the alarm points of a device and acknowledge them.



- Selecting multiple devices and acknowledging them at once acknowledges each alarm point of each device in that one action.
- The Acknowledge window also contains a text field that is used to enter a comment about who is acknowledging the alarm, and why.

## Notifications

Alarm points can use the same notification settings as the device, or they can have independent notifications. That is, each alarm point's set of notifications can be separate from any others, and each transition to a new severity can have its own notification. Notifications for alarm points follow the current Intermapper scheme of sending the notification to an identity. Each identity is configured to use a single notification method (sound, e-mail, modem paging, SNPP, running a script, etc.) to send the desired message. Alarm point notifications can have independent repeats, delays, and counts, as well. They are defined in the probe file as described in the [Alarm Point Notifier List](#) section.

## Log File Messages

Intermapper writes messages to the Event Log file for individual alarm point actions. The entries will be written on a change of severity, for notifications, acknowledgements, or for maintenance mode changes. The lines will have tab-delimited fields in this order:

- **Date-time** Date and time the entry was made into the log file
- **Severity** A four or five-character severity of the event (clear, minor, major, crit, unkn)
- **Identity** The identity of the alarm point, with the map, device, and the alarm point names separated by colons. (e.g., MapName:DeviceName:PointName)
- **Explanatory-text** The condition string or result-description of the alarm point

## Configuring Alarm Points

Alarm points are configured in a Custom SNMP Probe. The details are contained in the [Alarm Point Format](#) section of the manual.

# Alarm Point File Format

Alarm points are defined in the <snmp-device-alarmpoints> section that contains several lines of the format:

```
name: severity (condition-to-test) Condition-String [ =>
Notifier-list ]
```

Here is an example:

```
<snmp-device-alarmpoints>

-- Name:      Severity (Condition-to-Test)          Condition-
String => Notifier-List
  SiteTemp: critical ($Temp > $CriticalHighTemp) "VERY_HIGH_
TEMP" => PageFred
  SiteTemp:   major ($Temp > $MajorHighTemp)      "HIGH_TEMP"
            => PageFred

</snmp-device-alarmpoints>
```

The fields of each entry are:

- **Name** is the name of the alarm point. If multiple lines in this section contain the same Name, then they will be treated as the different thresholds for the same alarm point.
- **Severity** is one of 'critical', 'major', 'minor' or 'clear'. It defines the resulting severity of the given point test.
- **Condition-to-test** is an expression that evaluates to a boolean result, e.g. "(\$Temp > \$CriticalHighTemp)" You can use variables from the <snmp-device-variables> section or the <parameters> section in your expression. See the Probe Calculations topic's [Expression Syntax section](#) for details about valid expressions. In the example, \$Temp is a variable read from the SNMP device; \$CriticalHighTemp and \$MajorHighTemp are parameters set by the user.
- **Condition-String** is a string that describes the resulting status if the Condition-to-test evaluates to true.
- **Notifier-list** is an optional, comma-delimited list of notifier names. The notifier names listed here are mapped to actual "Intermapper Notifier Names" via the <snmp-device-notifiers> section.

When Intermapper evaluates alarm point expressions, it scans the list for a particular alarm point, and sets its status based on the first expression that "triggers". If no expression triggers, then Intermapper sets the alarm point severity to "Clear"

# Macros

Intermapper supports several macros that show information about an alarm point:

- `${alarmpointname}` shows the alarmpoint's severity as a five-character string. The strings are colored to match the severity. To use this facility, enter the alarmpoint name enclosed in `${...}`. For example, to show the SiteTemp alarm point's severity (above), enter:

```
${SiteTemp}
```

and it would generate the strings "CRIT ", "MAJOR", "MINOR", or "CLEAR", with the appropriate color.

- `${alarmpointname:condition}` shows the alarmpoint's condition string, as defined in the `<snmp-device-alarmpoints>` section. For example, to show the SiteTemp alarm point's condition above, type:

```
${SiteTemp:condition}
```

and it would generate the string "VERY\_HIGH\_TEMP". This string may contain any markup as described on the [Probe File Description](#) page.

## AlarmPoint Facilities

Intermapper provides several alarm point facilities:

### Underscore Feature

Use the Underscore feature to control whether an alarm point is cleared permanently or temporarily when you reset the AlarmPointList.

A device's AlarmPointList contains the following important information:

- alarm points that are **currently in alarm** (i.e., not in Clear state)
- alarm points that were **recently in alarm**, but are now Clear.

This minimizes the clutter in the AlarmPointList, so that it only contains relevant and interesting information. (All the other alarm points are assumed to be clear, and therefore can be ignored.)

The "recently in alarm" qualifier deserves explanation. There is a link in the device's Status Window that allows you to "reset the alarm point list" and remove the cleared alarm points

Alarm point names that begin with an underscore ("\_") are treated in this way (hence the name of the facility.) For example: `_SWO_PROC_SWO_PROC` is an underscore alarm point, whereas `SWO_PROC_SWO_PROC` is treated as a normal alarm point.



## State Transitions

- **Startup** When a map is opened, devices with alarm points are in the Unknown (grey) state, and their AlarmPointList is empty.
- **Normal Operation:** As Intermapper receives information about a device's state, either from a poll or a received trap, it sets its icon color accordingly. If this information sets a new non-underscore alarm point (one whose state is currently not known), it is added to the AlarmPointList, in the proper color/severity. If it is for a new underscore alarm point, it is added to the AlarmPointList only if the severity is not Clear. If this new information updates an existing alarm point, then that point's severity will be updated in the AlarmPointList.
- **Resetting the AlarmPointList:** When the user clicks an AlarmPointList's **Reset** link in the status window, all the Clear alarm points are removed from the device's AlarmPointList. All alarm points that are in alarm (not Clear) remain in the list.

**Note:** The **Reset** link removes all "clear" alarm points, but the effect is permanent only for "underscore" alarm points. non-underscore alarm points are cleared temporarily; the cleared non-underscore alarm points reappear in the status page in the next refresh (unless the condition that resulted in "clear" severity has changed).

## Resetting to Neutral Alarm State

Intermapper can receive a trap or some other command that sets a device into the "Startup" state described above. This is useful for the process of re-synchronizing Intermapper's notion of a device's state with its actual state. A single trap could indicate, "I don't know the state of a device" (perhaps because it had gone down, and now came back up), and Intermapper reflects that lack of certainty by clearing the alarm point list and turning the device grey.

Intermapper provides a "reset" severity that resets the device to its "power on" state. That is, the device and all its alarm points are set to the Unknown state. Such an alarm point is never listed in the AlarmPointList.

Usage:

```
DeviceResetRule: reset ( reset-condition ) condition-string  
=> Notifier-List
```

If reset-condition is true and there are alarms to be cleared, an event log entry will be created for the action. If the condition-string is not empty, a notification will be sent to the Notifier-List.

## Facilities to speed up rule evaluation

We will implement a "Break" severity that will abort the processing of the remaining rules of the probe if ever its expression is true. Such an alarm point will never be listed in the AlarmPointList.

Usage:

```
BreakRule: break ( break-condition ) condition-string =>
Notifier-List
```

In a break rule, Notifier-List is not used. If the condition-string is not empty, an event log message will be created everytime this rule fires (this is done for debugging purposes).

## Sample probe

Below a sample probe that uses all the features described above. To test the probe, use the net-snmp *snmptrap* program to send traps to the device. For example, to set \$trapVar to 5, use the following command line:

```
snmptrap.exe -v2c -c community computername ''
1.3.6.1.4.1.11898.2.1 1.3.6.1.4.1.11898.2.1.18.1.18 i 5
```

In the sample probe below, three trap variables are used in three separate alarm points. The first group of alarm points (\_trapVarAP) is an "underscore" alarm point, the state is not shown unless the alarm point reaches a non-clear state (minor, major, critical). Note that setting the \$trapVar variable to 4 never brings the alarm point state to critical since there is a break rule right before the rule that brings the state to critical state:

```
<header>
  type           =      "custom-snmp"
  package        =      "com.dartware"
  probe_name     =      "snmp.testalarmpoint"
  human_name     =      "Alarm point test example"
  version        =      "0.1"
  address_type   =      "IP,AT"
  flags          =      "SNMPv2c"
  port_number    =      "161"
</header>

<description>
  ...
</description>

<snmp-device-variables>
  trapVar,      1.3.6.1.4.1.11898.2.1.18.1.18,
```

```
TRAPVARIABLE, "trap variable 1"
    trapVar2,      1.3.6.1.4.1.11898.2.1.18.1.19,
TRAPVARIABLE, "trap variable 2"
    trapVar3,      1.3.6.1.4.1.11898.2.1.18.1.20,
TRAPVARIABLE, "trap variable 3"
</snmp-device-variables>

<snmp-device-notifiers>
    NotifySomeone: "NotifyFred:0:0:0"
</snmp-device-notifiers>

<snmp-device-alarmpoints>
    -- sample underscore alarm point
    -- the three reset alarm point have the same effect:
    resetting the device state to initial state

    _trapVarAP: clear ($trapVar == "2") "trapVar - clear" =>
NotifySomeone
    _trapVarAP: reset ($trapVar == "3") "trapVar - reset" =>
NotifySomeone
    _trapVarAP: minor (${trapVar} == "4") "trapVar - minor"
=> NotifySomeone
    _trapVarAP: critical (${trapVar} == "5") "trapVar -
major" => NotifySomeone
    _trapVarAP: break ($trapVar == "6") "trapVar - break" =>
NotifySomeone
    _trapVarAP: critical (${trapVar} >= "6") "trapVar -
critical" => NotifySomeone

    -- other, normal alarm points
    trapVarAP2: clear ($trapVar2 == "2") "trapVar2 - clear"
=> NotifySomeone
    trapVarAP2: reset ($trapVar2 == "3") "trapVar2 - reset"
=> NotifySomeone
    trapVarAP2: break ($trapVar2 == "4") "trapVar2 - break"
=> NotifySomeone
    trapVarAP2: critical (${trapVar2} >= "4") "trapvar2 -
critical" => NotifySomeone

    trapVarAP3: clear ($trapVar3 == "2") "trapVar3 alarm" =>
NotifySomeone
    trapVarAP3: reset ($trapVar3 == "3") "trapVar3 alarm" =>
NotifySomeone
    trapVarAP3: break ($trapVar3 == "4") "trapVar4 alarm" =>
NotifySomeone
    trapVarAP3: critical (${trapVar3} >= "4") "trapvar alarm"
=> NotifySomeone
</snmp-device-alarmpoints>
```

```
<snmp-device-display>
  \B5\Trap variable values\0P\
  \4\trapvar:\0\ $trapVar, ${_trapVarAP:condition}\0P\
  \4\trapvar2:\0\ $trapVar2, ${trapVarAP2:condition} \0P\
  \4\trapvar3:\0\ $trapVar3, ${trapVarAP3:condition} \0P\

  $alarmpointlist
</snmp-device-display>
```

## Alarm Point Notifier List

The `<snmp-device-notifiers>` section contains several lines of the format:

```
NotifierName: "notifier-rule" [ , "notifier-rule" ]
```

where the *NotifierName* is an identifier that can be used in the notifier-list section of the alarm-points section, and the *notifier-rule* is a quoted specification for a notification rule.

A *notifier-rule* contains the name of the actual Intermapper notifier and the notification delay, repeat and count, using the format below (the quote characters are required):

```
"name:delay:repeat:count"
```

Delay and repeat are specified in minutes. If values for delay and repeat are omitted, the value is zero. The count is the number of additional times the notification is repeated. If repeat is zero, the count will be ignored because there is no repeat. If repeat is non-zero and the count is omitted, the count is infinite (repeat forever).

```
<snmp-device-alarmpoints>
  -- Name:      Severity (Condition-to-Test)      Condition-
String => Notifier-List
  SiteTemp: critical ($Temp > $CriticalHighTemp) "VERY_
HIGH_TEMP" => PageFred
  SiteTemp:  major ($Temp > $MajorHighTemp)      "HIGH_
TEMP"      => PageFred
</snmp-device-alarmpoints>

<snmp-device-notifiers>
  PageFred: "Fred via Pager:0:0:0"
</snmp-device-notifiers>
```

In this example, either of the SiteTemp alarms triggers the "PageFred" notifier. Looking farther down in the <snmp-device-notifiers> section, we see that "PageFred" sends the notification to the "Fred via Pager" (which is defined in the **Notification list**.)

# Probe Calculations

Intermapper can compute values from data retrieved from devices, including SNMP MIB variables, round-trip time, packet loss, availability, etc. The results of these computations can be compared to thresholds to set device status and indicate problems.

## Expression Syntax

Intermapper's Expression Syntax has the following features:

- Supports arithmetic expressions using +, -, \*, /, %, and unary minus.
- Supports the use of parentheses to group sub-expressions for calculation first.
- Stores all intermediate and final results as double-precision floating point numbers.
- Supports relational operators <, >, <=, >=, =, <>, ==, !=. The value for TRUE is "1.0". The value for FALSE is "0.0".
- Supports short-circuit logical operators 'and', 'or', 'not' as well as &&, ||, !.
- Supports variables and functions from a provided symbol table. Variables may use \$var syntax or \${var} syntax. Persistent variables retain values between polls. For more information, see [Using Persistent Variables](#).
- Supports built-in functions for bitwise operations, rounding, and other common mathematical functions.
- Supports embedded string comparisons and simple regular expression tests. A variable in double-quotes will be treated as a string. All double-quoted strings are interpolated for variables in a Perl-like fashion. The use of + as the concatenation operator is supported. See below for [an example that uses Regular Expressions](#).

The set of capabilities are derived from C, Perl, Excel, and expr(1).

## Reserved keywords

- and
- or
- not

## Precedence Table (Least to Most)

1. Assignment: :=
2. Conditional Expression: ?:
3. Logical Or: 'or', ||
4. Logical And: 'and', &&

5. Equality Tests: ==, =, !=,
6. Relational Tests: <, >, <=, >=
7. Addition, Subtraction, Concatenation: +,-
8. Multiplication, Division, Modulo: \*, /, %
9. String Matching: =~, !~
10. Unary: -, !, 'not'

## Built-in Numeric Functions

- `abs( x )` - Absolute value of 'x'.
- `round(x),round(x,y)` - Round 'x' to nearest integer.
- `trunc( x )` - Remove all digits after the decimal point; e.g. `trunc( 3.987 ) = 3`.
- `min( x1, x2, ... , xn )` - Minimum value of x1, x2, ..., xn.
- `max( x1, x2, ... , xn )` - Maximum value of x1, x2, ..., xn.
- `bitand( x, y )` - Bitwise 'and' of 'x' and 'y'.
- `bitor( x, y )` - Bitwise 'or' of 'x' and 'y'.
- `bitlshift( x, y )` - Bits of 'x' shifted left by 'y' bits.
- `bitrshift( x, y )` - Bits of 'x' shifted right by 'y' bits.
- `bitxor( x, y )` - Bitwise exclusive-or of 'x' and 'y'.
- `sin( x )` - sine of 'x' where 'x' is in radians.
- `cos( x )` - cosine of 'x' where 'x' is in radians.
- `tan( x )` - tangent of 'x' where 'x' is in radians.
- `pi()` - value of PI (e.g. 3.14159...)
- `pow( x, y )` - 'x' to the power of 'y'.
- `sqrt( x )` - square root of 'x'.
- `exp( x )` - e to the power of 'x', where e is the base of the natural logarithms.
- `log( x )` - natural logarithm of 'x'.
- `log( x, y )` - logarithm of 'x' to base 'y', e.g. `log( 100, 10 ) = 2`
- `time()` - Time in seconds since 1 January 1970 UTC.

## Built-in String Functions

- `defined(str)` - Takes a string argument, and returns a non-zero value (1) if the variable name specified in the input string is defined.
- `strfind( strToBeSearched STRING, substrToFind STRING )` - Case sensitive match returns the position of the first matching substring.
- `strifind( strToBeSearched STRING, substrToFind STRING )` - Case insensitive match returns the position of the first matching substring.

- [strlen](#)(str) - Returns the length in bytes of the string 'str' or the combined length of all string arguments.
- [sprintf](#)( fmt, ...) - Returns formatted string using format specifier 'fmt'. Format specifier 'fmt' contains format codes that begin with '%'.  
 • [strftime](#)( fmt, [secs]) - Returns formatted date/time string using format specifier 'fmt'.
- [strptime](#)( str, fmt ) - Returns the number of seconds since UTC 1970 represented by the given date/time string, as interpreted using the specified format code.
- [subid](#)(oid, start, length) - Gets the specified length sub-OIDs from a given OID string, starting from index start (the index starts from 0).
- [substr](#)( str, offset, len )
- [unpack](#)( binary str, fmt )
- Regular Expressions - See below for [an example that uses Regular Expressions](#).

## Function Descriptions

### defined

FUNCTION defined(variable:STRING):INTEGER;

Returns a non-zero value (1) if the variable name specified in the input string is defined (it has already been assigned a value).

**Note:** This function takes a string argument. Note the usage below.

Example:

```
defined("var2") == 1 ? "$var2 is defined" : "$var2 is
undefined"
```

### round

FUNCTION round(x:DOUBLE, y:INTEGER):DOUBLE;

FUNCTION round(x:DOUBLE):INTEGER;

Rounds a given double value (*x*) to the nearest integer or to the given number of decimal places (*y*).

Examples:

```
round(8.6) --> 9
round(3.14159, 3) = 3.142
```



## strfind

FUNCTION strfind( strToBeSearched:STRING, substrToFind:STRING ):INTEGER

Case-sensitive string match returns an integer representing the position of the first occurrence of a substring in the string. If the substring is not found, the function returns a value of -1.

Example:

```
strfind( "Ethernet Interface", "int")  
  
returns -1 (did not find the substring)
```

## strifind

FUNCTION strifind( strToBeSearched:STRING, substrToFind:STRING ): INTEGER

Case-insensitive string match returns an integer representing the position of the first occurrence of a substring in the string. If the substring is not found, the function returns a value of -1.

Example:

```
strifind( "Ethernet Interface", "int")  
  
returns 9 (found the substring at position 9)
```

## strlen

FUNCTION strlen(str[, ...]:STRING):INTEGER

Returns the length of the string *str* in bytes.

Returns the combined length of all string arguments in bytes.

Examples:

```
strlen( "HelpSystems" ) --> 11  
strlen( "HelpSystems", "2000" ) --> 15
```

## sprintf

FUNCTION sprintf( fmt:STRING, ... ):STRING

Returns formatted string using format specifier *fmt*. Format specifier *fmt* contains format codes that begin with '%'. The following format codes are supported:

- c - Formats numeric argument as ascii character
- d - Formats numeric argument as decimal integer
- o - Formats numeric argument as octal integer
- x - Formats numeric argument as hexadecimal number (lower case)
- X - Formats numeric argument as hexadcimal number (upper case)
- u - Formats numeric argument as decimal integer (unsigned)
- s - Formats argument as an ascii string (NUL terminated)
- a - Formats argument as a hexadecimal string with bytes separated by ':'
- f - Formats numeric argument as floating point (fixed precision)
- e - Formats numeric argument as floating point (scientific notation)
- g - Formats numeric argument as floating point (easy to read)
- % - Prints a percent sign

The general specification for a format code is:

```
% [-] [<width>] [. <precision> ] <code>
```

## String Formatting

For string data using %s, the width specifies the minimum width of the output field, and the precision specifies the number of characters to output. If the number of output characters is less than the minimum field width, the output is padded with spaces.

Example:

```
sprintf( "%12s", "HelpSystems" )
Results in " HelpSystems"
sprintf( "%s", "HelpSystems" )
Results in "HelpSystems"
```

The default alignment is to the right; so padding is added to the beginning of the string. To left align the output of %s, you need to include a '-' immediately following the '%':

```
sprintf( "%-12s", "HelpSystems" )
Results in "HelpSystems "
sprintf( "%-10.4s", "HelpSystems" )
Results in "Help      "
```

## Integer Formatting

Integers format similar to strings, except the <precision> field specifies the maximum field width, and this is enforced by padding with 0's if necessary.

```
sprintf( "%5d", 12 )
Results in "    12"
```

```
sprintf( "%-5d", 12 )
    Results in "12  "
sprintf( "%6.5d", 12 )
    Results in " 00012"
sprintf( "%-2X", 15 )
    Results in "F  "
sprintf( "%-2.2x", 15 )
    Results in "0f"
```

## Floating Point Formatting

The floating point format codes use the <precision> field to specify the number of decimal places following the decimal point. %f uses the format '[-]ddd.ddd', and %e uses the format '[-]d.ddde+-dd'.

```
sprintf( "%f", 1/2 )
    Results in "0.500000"
sprintf( "%5.3f", 1/2 )
    Results in "0.500"
sprintf( "%e", 1/2 )
    Results in "5.000000e-01"
sprintf( "%g", 1/2 )
    Results in "0.5"
```

## Address Formatting

The %a format code outputs a string in hexadecimal.

```
sprintf( "%a", "\x01\x02\x03" )
    Results in "01:02:03"
sprintf( "%a", "HelpSystems" )
    Results in "48:65:6C:70:53:79:73:74:65:6D:73"
```

## strftime

FUNCTION strftime( fmt [, time] )

Returns formatted date/time string using format specifier 'fmt'. Format specifier 'fmt' contains format codes that begin with '%'. If a time argument is provided, it must be in seconds since UTC 1970. If no time argument is provided, it defaults to the current time. The following format codes are supported on all platforms:

- a - Abbreviated weekday name
- A - Full weekday name
- b - Abbreviated month name
- B - Full month name

- c - Date and time formatted something like "Tue Feb 06 10:25:22 2007"
- d - Day of month (01-31)
- H - Hour number (00-23)
- I - Hour number (01-12)
- j - Day of the year number (001-366)
- m - Month number (01-12)
- M - Minute number (00-59)
- p - "AM" or "PM"
- S - Second number (00-61)
- s - returns the number of seconds since the Epoch, 1970-01-01 00:00:00 +0000 (UTC).
- U - Week of the year number (00-53). First Sunday is day 1 of week 1.
- w - Weekday number (0-6). Sunday is 0.
- W - Week of the year number (00-53). First Monday is day 1 or week 1.
- x - Date.
- X - Time.
- y - Two-digit year number (00-99)
- Y - Year with century (e.g. 2007)
- z - returns the +hhmm or -hhmm numeric timezone (that is, the hour and minute offset from UTC) for the InterMapper server.
- % - Prints a percent sign (when preceded by a %)

The `strptime` function is implemented using the identically named function in the underlying system. Other format codes may work, but these are not portable.

```
strptime( "%c")
Results in "Tue Feb  6 11:19:24 2007"
strptime( "%Y-%m-%d", 1170778895)
Results in "2007-02-06"
```

## strptime

FUNCTION `strptime( str , fmt )`

Returns the number of seconds since UTC 1970 represented by the given date/time string, as interpreted using the specified format code. Basically, this function can be used to parse dates.

This function uses the same underlying format codes as `strptime`.

Example:

```
strftime( "%Y", strptime( "1990", "%Y"))  
Results in "1990"
```

## subid

FUNCTION subid(oid, start, length)

Gets the specified length sub-OIDs from a given OID string, starting from index start (the index starts from 0). When the start index is negative, it will be counted from the end of the OID string.

Examples:

```
subid("1.3.6.1.2.1.4.20.1.1.10.10.2.20", 0, 2)    --> "1.3"  
subid("1.3.6.1.2.1.4.20.1.1.10.10.2.20", -4, 4)   -->  
"10.10.2.20"  
subid("1.3.6.1.2.1.4.20.1.1.10.10.2.20", 4, 4)    -->  
"2.1.4.20"  
subid("1.3.6.1.2.1.4.20.1.1.10.10.2.20", -2, 4)   --> "2.20"  
subid("1.3.6", 3, 4)    --> ""  
subid("1.3.6", 2, 4)    --> "6"  
subid("1.3.6", -4, 4)   --> "1.3.6"  
subid("1.3.6", -2, 4)   --> "3.6"
```

## substr

FUNCTION substr(str:STRING, offset:INTEGER):STRING;

FUNCTION substr(str:STRING, offset:INTEGER, length:INTEGER):STRING;

Extract a substring out of *str* and return it. The substring is extracted starting at *offset* characters from the start of the string.

- If *offset* is negative, the substring starts that far from the end of the string instead; *length* indicates the length of the substring to extract.
- If *length* is omitted, everything from *offset* to the end of the string is returned.
- If *length* is negative, the length is calculated to leave that many characters off the end of the string. If neither *offset* nor *length* is supplied, the function will return *str*. (See Perl *substr*).

Examples:

```
substr( "0123456789", 7)      --> "789"  
substr( "0123456789", 4, 2)  --> "45"  
substr( "0123456789", 4, -2) --> "4567"  
substr( "0123456789", -2, 1) --> "8"
```

## unpack

FUNCTION `unpack(str:STRING, format:STRING):VALUE`

Take a string *str* representing a data value and convert it into a scalar value. The format string specifies the type of value to be unpacked. (See perl *unpack*).

- If the input string is shorter than the expected number of bytes to be unpacked, treat the input string as if it is padded with zero bytes at the end.

```
unpack("\1\2\3", ">L")
```

is the same as

```
unpack("\1\2\3\0", ">L")
```

- If the input string is longer, the remaining bytes in the input are ignored.
- If the endian modifier is not supplied, the target platform's byte order (little endian on Windows, big endian on Mac) is used.
- If format specifier is not supplied, the function returns *str*.

| Format specifier | Description  |
|------------------|--|
| c                | signed character value (1 byte)  |
| C                | unsigned character value (1 byte)  |
| l                | signed long value (4 bytes)  |
| L                | unsigned long value (4 bytes)  |
| s                | signed short value (2 bytes)   |
| S                | unsigned short value (2 bytes)   |
| #B               | a base64 string (all bytes)  |
| >                | big-endian modifier  |
| <                | little-endian modifier   |
| H                | decodes the given hexadecimal value and returns an integer (up to 32-bits) |
| #H               | decodes the given hexadecimal value and returns a string                   |

Examples:

```
unpack( "F", "c")                --> 70 (decimal
ASCII value)
unpack( "F", "H")                --> 15 (Hex
converted to decimal value)
unpack( "48656C7053797374656D732C20496E632E", "#H") ->
HelpSystems, Inc.
```

### Notes:

- It is difficult to create examples that input unprintable characters. Refer to [the perl documentation](#) for more information about `unpack()`.
- The `unpack()` function supports one format code in the format string.

# Using Regular Expressions In Custom SNMP Probes

You can use a regular expression to divide a string into separate variables after retrieving it from a device. In the example below, a customer had a piece of equipment that returned the following information in `sysDescr.0`:

```
FW TR6-3.1.4Rt_F213E4, 2.4GHz, 0dBi ext. antenna
```

They created a probe that retrieved `sysDescr.0` and then parsed out those strings with the following commands in the `<snmp-device-variables>` section of the probe:

```
<snmp-device-variables>
  sysDescr, 1.3.6.1.2.1.1.1.0,
            DEFAULT, "system description"
  firmware, "$sysDescr" =~ "^FW ([^,]+), (.+)Hz, (.+)
antenna" ; "${1}", CALCULATION, "Firmware"
  frequency, "${2}",
            CALCULATION, "Frequency"
  antenna, "${3}",
            CALCULATION, "Antenna"
  . . .
</snmp-device-variables>
```

1. Retrieve `sysDescr.0` (OID of 1.3.6.1.2.1.1.1.0) and assign it to the variable `$sysDescr`.

2. Set the value of \$firmware based on the calculation. There are many things going on in this line:
  - The "=~" operator indicates that the \$sysDescr variable should be parsed using the regular expression string that follows.
  - This regular expression breaks the string at the comma characters. The "[^,]" matches any single character that isn't a comma; adding a "+" forms a pattern that matches multiple non-comma characters.
  - Parentheses around a pattern serve to memorize a string. Each pair of paren's matches a string whose value is placed in variables numbered \${1}, \${2}, \${3}, etc.
  - The semicolon followed by "\${1}" indicates that the entire CALCULATION should return the value of \${1} as a string.
  - The variable \$firmware thus gets assigned the value of \${1}
3. Assign the variable \$frequency with the result of the second match (\${2}).
4. Assign the variable \$antenna with the result of the third match (\${3}).

**Note:** It is beyond the scope of this manual to describe the full capabilities of regular expressions. There are a number of tutorials available on the web. One example is the [Perl Regular Expression Tutorial](#).



# Specifying SNMP OIDs in Custom Probes

## Introduction

Intermapper supports two kinds of OID's: Numeric and Symbolic. The Symbolic OIDs become available when a MIB has been imported into Intermapper.

In addition, Intermapper supports three kinds of OID expressions: Get-Next, Trap-Conditional, and Index-Derived.

## Numeric OID's

Numeric OID's contain only numbers separated by periods. Preceding periods are ignored. A trailing period is allowed if there is only one subid.

### Examples:

```
.1  
1.  
1.3.6
```

### Invalid examples:

```
1 (no period)  
1.3.6. (trailing period but with multiple subids)  
1.3.6.blah (not numeric)  
1.3.6.1.2.1.system.sysUpTime.0 (not numeric)
```

Unlike Net-SNMP, Intermapper ascribes no special meaning to OID's that begin with a period; all numeric OID's are considered absolute.

Errors in numeric OID's are reported by the system to the Event Log when the error is in a custom probe. The error message will have the form:

```
Syntax error in OID "1.3.6.1..1.2"
```

## Symbolic OID's

A symbolic OID begins with a letter, after ignoring any preceding periods. Intermapper must be able to locate a MIB file that defines the symbols used. There are three types of symbolic OID's:

1. Simple symbols specify a starting symbol and zero or more trailing subid's.
2. Relative symbols specify a starting symbol and one or more subid symbols.
3. Scoped symbols specify the name of the MIB, the scope operator ::, followed by a simple or relative symbol.

Relative and scoped symbols are handy when a symbol is ambiguous, i.e. the same symbol name is defined differently in two separate MIB files. You should prefer the scoped OID form, when possible.

Symbolic names are case-sensitive.

### Examples:

```
Simple: sysUpTime
        sysUpTime.0
        enterprises.9.2.3.4.5

Relative: system.sysUpTime
          system.sysUpTime.0

Scoped:  SNMPv2-MIB::sysUpTime
        SNMPv2-MIB::system.sysUpTime.0
```

### Invalid Examples:

```
Simple:  sysUpTiime      (misspelled; not found)
        sysupTime      (wrong case)
        sys%pTime      (disallowed character %)
        sysUpTime.0.    (bad; trailing period)

Relative: system.ifIndex (bad; ifIndex isn't under system)

Scoped:  SNMPv2-MIB.sysUpTime (bad; must use :: for scoped
OID)
        IF-MIB::sysUpTime     (bad; wrong MIB module for
sysUpTime)
```

Errors in symbolic OID's are reported by the system to the Event Log when the error is in a custom probe. The error message will have the form:

```
Syntax error in OID "sys%pTime"
```

## OIDs indexed by Strings

Certain MIBs specify tables that are indexed by strings. The net-snmp documentation at <http://www.net-snmp.org/tutorial/tutorial-5/commands/output-options.html>

describes this. A convenient way to enter these OIDs is:

```
NET-SNMP-EXTEND-MIB::nsExtendOutLine."LOG"
```

and a SNMP variable could be created like this:

```
outLine, NET-SNMP-EXTEND-MIB::nsExtendOutLine."LOG", DEFAULT, ""
```

## Limitations of Symbolic OID's

1. Symbolic OID's will only work if the necessary MIB file is loaded into Intermapper. If Intermapper cannot resolve the symbolic OID using a MIB file, this is considered a syntax error in the symbolic OID. At this time, there is no way to bundle a MIB file with a probe as one file; this is a future direction.
2. It may happen that two or more MIB files define the same symbol. When this happens, Intermapper may pick the wrong definition. You can avoid this by using the scoped OID form.

## Get-Next OID Expressions

Intermapper has a special syntax for "get-next" style OID's - attach a + to the end of the OID.

Normally, when you specify a variable to query in a custom SNMP probe, you specify the complete OID, including the instance. For example, you might specify "sysUpTime.0" or "ifInOctets.13". For sysUpTime, the .0 specifies the (only) instance. For ifInOctets, the .13 specifies the value for ifIndex 13.

There are occasions when you want to query a variable using a preceding OID. For example, you might want to query the value of ifInOctets for the first interface, but you can't assume the ifIndex of the first interface is 1. Here's how you would specify the OID:

```
ifInOctets+
```

To retrieve the value of ifInOctets for the interface whose ifIndex follows 13, specify the OID with a plus:

```
ifInOctets.13+
```

The plus sign must immediately follow the OID. Technically, it's not part of the OID, but considered an operator in Intermapper's OID expression language.

Note: Get-Next OID expressions will not work for custom SNMP probes that specify get-request queries.

## Trap-Conditional OID Expressions

Trap-conditional OID expressions allow you to assign a variable only when it occurs in the varbind list of a certain trap. For example, you might want to set the value of your probe's

"sysUpTimeCrashed" variable to value of the "sysUpTime.0" variable included in the varbind list of a "systemCrashed" trap. However, you don't want to set "sysUpTimeCrashed" when you see the sysUpTime.0 value in any other received trap. To restrict the assignment of sysUpTime.0 to only the systemCrashed trap, you need to specify both the systemCrashed trap OID and the sysUpTime.0 OID using the ?: operator. This combination is called a "Trap-Conditional" OID, or "Trap OID" for short.

Examples:

```
systemCrashed?:sysUpTime.0  
  
sysTrapOID?:sysContact  
  
SOMEMIB::sysTrapOID.1?:SMIV2-MIB::sysContact
```

Supported in 4.4, the legacy format for trap OID's is a numeric OID followed by an OID:

```
1.3.6.1.2.1::sysUpTime.0
```

The legacy format does not allow use of a symbolic name for a trap OID; this conflicts with the scoped format above. The use of :: for Trap-conditional OID expressions is deprecated. Please use ?: in the future.

## Index-Derived OID Expressions

When querying tables from SNMP devices, it is often useful to assign the value of a variable from a row's OID index. This technique will work even if the values used to index the row have an access of "not-accessible".

For more information, see [Index-Derived Variables](#) in the <snmp-device-variables-ondemand> Section topic.

# SNMP Probe Example

```
<!--
  Single OID Custom Probe
  (com.dartware.snmp.oidsingle.txt)
  Custom Probe for Intermapper
  (http://www.intermapper.com)
  Please feel free to use this as a base for further
  development.

  10 May 2007 Cloned from High Threshold probe -reb
  3 Jul 2007 Changed probe name to Single OID Viewer -
reb
  4 Sep 2012 Added a datasets section -jpd

  You can read the Developer Guide to learn more about
  Intermapper Probes. It's at:
  http://intermapper.com/go.php?to=intermapper.devguide
-->

<header>
  "type"          = "custom-snmp"
  "package"       = "com.dartware"
  "probe_name"    = "snmp.oidsingle"
  "human_name"    = "SNMP - Single OID Viewer"
  "version"       = "1.4"
  "address_type"  = "IP,AT"
  "port_number"   = "161"
  "display_name"  = "SNMP/Single OID Viewer"
  "flags"         = "Minimal"
</header>

<snmp-device-properties>
  nomib2  = "true"
  pdutype = "get-request"
</snmp-device-properties>

<description>
\GB\Single OID Viewer\P\

This probe retrieves a single SNMP MIB variable and
displays it in the device's Status Window.
```

\ib\Variable\p\ specifies the MIB name or OID for the value to retrieve. If you have imported the MIB for this device, you may enter the symbolic name for this value, otherwise, simply enter its OID here.

\bi\Legend\p\ is a text string used to identify the variable in the status window and any strip charts. If left blank, the variable's name or OID will be used.

\bi\Units\p\ is a text string that will be displayed next to the value in the Status Window. You can use it for the unit of measure (packets/sec, degrees, etc.)

\bi\Tag\p\ is a short text string that identifies a particular class of dataset. Tags will be used to correlate different variables from different probes that describe the same thing, such as CPU% or temperature.

</description>

-- Parameters are user-settable values that the probe uses for its comparisons.  
-- Specify the default values here. The customer can change them and they will be retained for each device.

```
<parameters>
  "Variable"  = "ifNumber.0"
  "Legend"    = ""
  "Units"     = ""
  "Tag"       = "exampletag"
</parameters>
```

-- SNMP values to be retrieved from the device, and  
-- Specify the variable name, its OID, a format (usually DEFAULT) and a short description.  
-- CALCULATION variables are computed from other values already retrieved from the device.

```
<snmp-device-variables>
```

```
    theLegend,  ($Legend!="" ? "$Legend" : "$Variable"),
CALCULATION, "Legend/OID"
    theOID,     $Variable ,      DEFAULT,  "$theLegend"
```

```
</snmp-device-variables>
```

```
-- The <snmp-device-display> section specifies the text  
that will be appended  
-- to the device's Status Window.
```

```
<snmp-device-display>  
\B5\ $theLegend:\0P\ $theOID \3G\Units\mp0\  
</snmp-device-display>
```

```
<datasets>  
$theOID, "$Tag", "$Units", "false", "$Legend"  
</datasets>
```

# SNMP Trap Probes

```
type = "custom-snmp-trap"
```

A *trap* is an unsolicited packet sent from a device to Intermapper (or other SNMP management console). The trap generally contains one or more data values that give information about the device's state.

When a trap arrives, Intermapper first determines which device(s) on the enabled maps should receive information from the trap. Intermapper examines the Agent Address (for relayed traps) or the Source IP address, and passes a copy of the trap packet to *each* device on the maps whose IP address matches. For example, if a device with the IP address is on two maps, or is present twice on the same map, each of those devices will receive a copy of the trap.

Intermapper then parses out all the values from the trap and assigns them to trap variables for use in the remainder of the probe. Intermapper then re-evaluates the expressions in the probe, and sets the device status appropriately. If a particular trap variable is not set by an incoming trap, expressions containing that variable are *not* evaluated. See the [The <snmp-device-variables> Section for Traps](#) section below for details of defining trap variables.

Finally, as a result of receiving the trap, Intermapper re-polls the device that sent the trap. This guarantees that Intermapper has the most up-to-date information about the device's state. If another trap arrives before the final response of this new poll has returned, Intermapper will complete the current poll and initiate another round of polling to get the new state.

**Note:** A trap is sent as a UDP packet. If something on your network is causing packet loss, it is possible to lose a trap packet. Therefore, HelpSystems recommends that you don't rely completely on traps as a means for monitoring the health of a device. There is no substitute for regular polling.

The [Example Trap Probe](#) demonstrates how to retrieve and display the contents of a trap.



# The <snmp-device-variables> Section For Traps

A **Trap Variable** is a variable defined in a custom probe file whose value is set to a value received in a trap. Intermapper has three kinds of Trap Variables, only one of which can be declared in a probe:

- **Packet Trap Variables** - a set of variables automatically set by Intermapper when a trap is received.
- **Positional Trap Variables** - a set of variables automatically set by Intermapper. Use positional Trap Variables to access data from the trap's VarBind list by position in the list.
- **Named Trap Variables** - variables you define by associating an SNMP OID with a name. If the OID exists in the trap's VarBind list, the variable is set to the value in the trap.

A Trap Variable will never be polled: that is, Intermapper never sends an SNMP GetRequest or GetNextRequest to retrieve its value.

## Packet Trap Variables

In addition to the variables in the VarBind List, a probe can set variables based on the fields of the trap packet's header.

- **\$GenericTrap** - The GenericTrap field in the trap (SNMPv1). This field can take on the values:
  - 0 - coldStart;
  - 1 - warmStart;
  - 2 - linkDown;
  - 3 - linkUp;
  - 4 - authenticationFailure;
  - 5 - egpNeighborLoss;
  - 6 - An enterprise-specific value.
- **\$SpecificTrap** - The value of the SpecificTrap field in the trap. If the \$GenericTrap value is 0-5, the \$SpecificTrap is zero (0); otherwise it is a positive 32-bit value specified by the vendor (SNMPv1).
- **\$TimeStamp** - The TimeStamp field in the trap, in hundredths of a second.
- **\$Enterprise** - The value of the SNMPv1 enterprise field (SNMPv1)
- **\$CommunityString** - The value of the CommunityString field in the trap (SNMPv1, SNMPv2c).
- **\$TrapOID** - The value of the TrapOID field in the trap (SNMPv2c, SNMPv3).
- **\$AgentAddress** - The IP address of the SNMP agent that generated the trap.

- **\$SenderAddress** - The IP address of the device that sent the trap. This could be different from the \$AgentAddress when the sender is forwarding traps for the agent.
- **\$SnmpVersion** - Represents the version of the trap. Values can be 0 (v1), 1 (v2c) or 3 (v3).
- **\$VarbindCount** - The number of variables contained in the VarBind list.

## Positional Variables from the Varbind List

You can access values from the VarBind List by *position* using variables of the form:

- **\$VarbindValueN** - The value of the N'th variable in the trap's VarBind List
- **\$VarbindTypeN** - The type of the N'th variable in the trap's VarBind List
- **\$VarbindOIDN** - the OID of the N'th variable in the trap's VarBind List

**Note:** N may be from 1 to 50.

## Named Trap Variables

The only way to set a named Trap Variable value is to receive a trap that contains the OID in its VarBind List, or the set the named variable to the value of a positional variable. The [Probe Variables section](#) of this document describes the file format. Here is an example:

```
<snmp-device-variables>
  InterMapperTimeStamp, 1.3.6.1.4.1.6306.2.1.1.0,
  TRAPVARIABLE, "Timestamp"
</snmp-device-variables>
```

In this example, the variable \$InterMapperTimeStamp is set every time a trap arrives containing the OID 1.3.6.1.4.1.6306.2.1.1.0 in the VarBind List. Trap Variables that don't have values set by an incoming trap are left undefined.

A full [example trap file](#) is available.

Here's how several useful trap variables might be defined

```
<snmp-device-variables>
  genericTrapVar,      $GenericTrap,      TRAPVARIABLE,
    "Generic Trap"
  specificTrapVar,     $SpecificTrap,      TRAPVARIABLE,
    "Specific Trap"
  timeStampVar,        $TimeStamp,         TRAPVARIABLE,
    "Timestamp"
  enterpriseVar,       $Enterprise,        TRAPVARIABLE,
    "Enterprise"
  commStringVar,       $CommunityString,   TRAPVARIABLE,
```

```

    "Community String"
trapOIDVar,          $TrapOID,          TRAPVARIABLE,
    "Trap OID"
agentAdrsVar,        $AgentAddress,      TRAPVARIABLE,
    "Agent Address"
senderAdrsVar,        $SenderAddress,     TRAPVARIABLE,
    "Sender Address"
snmpVersionVar,       $SnmpVersion,      TRAPVARIABLE,
    "SNMP Version"
varbindCountVar,      $VarbindCount,     TRAPVARIABLE,
    "Varbind Count"
-- the first and second values from the Varbind List by
position
    trap_var1, $VarbindValue1, TRAPVARIABLE, "First value"
    trap_var2, $VarbindValue2, TRAPVARIABLE, "Second value"
</snmp-device-variables>

```

**Note:** The TRAPVARIABLE type causes the value to be displayed in the most useful format. You can also use one of following to force the display to a certain format. These variables are equivalent to their non-trapvariable counterparts, whose complete descriptions of the formats are available in [Probe Variables](#):

- TRAPVARIABLE-TOTAL-VALUE
- TRAPVARIABLE-PER-SECOND
- TRAPVARIABLE-PER-MINUTE
- TRAPVARIABLE-STRING\*
- TRAPVARIABLE-INTEGER
- TRAPVARIABLE-HEXADECIMAL\*
- TRAPVARIABLE-HEXNUMBER
- TRAPVARIABLE-DOUBLE

\* STRING and HEXADECIMAL are both strings; they can't be charted.

## Accessing Trap Variables by Position

When accessing VarBind List entries, you can access them either by name or by position. Access by name is much easier to program and understand, but there have been instances where a vendor's traps contained VarBind List entries with the same name: in those cases you must get their values by position. Below are examples of accessing VarBind List entries by name and by position.

Given this trap, Intermapper creates the following Event Log entry:

```

03/23 11:37:34 TRAP IC3 Demo System:Video Stream ENC01
LIVEWAVE-MIB::deviceFaulted (v2c)
{ LIVEWAVE-MIB::deviceUnitID : "5",
  LIVEWAVE-MIB::deviceName : "5 - Video Stream",
  LIVEWAVE-MIB::deviceStatus : "6" }

```

The trap contains these three values: deviceUnitID, deviceName, and deviceStatus. (Intermapper has already imported a LIVEWAVE MIB that defines these OIDs.)

The variables are declared in the variables section:

```
<snmp-device-variables>
  deviceUnitID, LIVEWAVE-MIB::deviceUnitID, TRAPVARIABLE,
  "Device Unit ID"
  deviceName,    LIVEWAVE-MIB::deviceName,    TRAPVARIABLE,
  "Device Name"
  deviceStatus, LIVEWAVE-MIB::deviceStatus, TRAPVARIABLE,
  "Device Status"
</snmp-device-variables>
```

When a trap is received, the probe variables above are set to the values of the trap variables from the VarBind list. One way you could use them is as follows:

```
<snmp-device-thresholds>
  critical: deviceStatus == 3 "Problem with $deviceUnitID
  $deviceName: Device status = $deviceStatus"
  okay:     deviceStatus == 1 "$deviceUnitID $deviceName
  functioning normally."
</snmp-device-thresholds>
```

You can also access the variables by position in the VarBind list:

```
<snmp-device-variables>
  deviceUnitID, LIVEWAVE-MIB::$VarbindValue1, TRAPVARIABLE,
  "Device Unit ID"
  deviceName,    LIVEWAVE-MIB::$VarbindValue2, TRAPVARIABLE,
  "Device Name"
  deviceStatus, LIVEWAVE-MIB::$VarbindValue3, TRAPVARIABLE,
  "Device Status"
</snmp-device-variables>
```

# The <snmp-device-display> Section for Traps

Use the <snmp-device-display> section to format the device's Status window exactly the same way as you do in an SNMP Probe. For more information, see the SNMP Probe's [<snmp-device-display> Section](#) topic.

# Trap Viewing and Logging

The contents of trap message are logged to the Event Log file at the time the trap is received. There are two forms: Short and Verbose. (The format is controlled by the **Verbose Trap Logging** checkbox in the **Server Settings > SNMP** preference pane.)

## Short Trap Format

```
06/08 20:50:29 TRAP TestMap:192.168.2.1 1.3.6.1.4.1.6306
(333)
{ "321", "456" } (via 192.168.1.233)<p>
```

## Verbose Trap Format:

```
06/08 20:50:05 TRAP TestMap:192.168.2.1 1.3.6.1.4.1.6306
(333)
{ 1.3.6.1.4.1.6306.99.1 : "321", 1.3.6.1.4.1.6306.99.2 :
"456" } (via 192.168.1.233)<p>
```

The fields of the trap entry in the log file are defined below, with examples in "[ ... ]":

- **Date and Time** - [ 06/08 20:50:05 TRAP ]  
The date and time followed by the word "TRAP"
- **Map Name and Device ID** - [ TestMap:192.168.2.1 ]  
The map name and device ID, separated by a colon (":")
- **Enterprise OID and Trap Field** - [ 1.3.6.1.4.1.6306 (333) ]  
The Enterprise OID, followed by the specific trap field in paren's
- **VarBind List** - The contents of the VarBind List, enclosed in curly braces, and separated by commas.  
Short format: { "321", "456" } shows only the values sent for each VarBind in quotes.  
or  
Verbose format: { 1.3.6.1.4.1.6306.99.1 : "321", 1.3.6.1.4.1.6306.99.2 : "456" } shows the OID, a colon (":"), and the OID's value in quotes.
- **Address** - [ (via 192.168.1.233) ] The address of the relaying computer, if present.

The verbose format shows all the information that was sent with the trap.

# Example - Trap Viewer Probe

The following example shows the handling of traps.

```
<!--
    SNMP Trap Viewer probe
    (com.dartware.snmp.trapdisplay.txt)
    Probe for Intermapper (http://www.intermapper.com)

    Copyright© HelpSystems, LLC.
    Feel free to use this as the basis for creating new
    probes.

    25 Apr 2005 Original version - reb
    4 May 2005 Changed to "custom-snmp-trap" -reb
        Modified for IM 4.4 header/display items.
    8 May 2007 Added special trap variables to the probe and
    display -reb
    29 May 2007 Changed probe name to "Trap Display", updated
    description -reb
    1 Jun 2007 Changed probe name to "Trap Viewer"; tweaked
    description;
        left canonical name alone -reb
-->

<header>
    "type"                =        "custom-snmp-trap"
    "package"              =        "com.dartware"
    "probe_name"           =        "snmp.trapdisplay"
    "human_name"           =        "Trap Viewer"
    "version"              =        "2.2"
    "address_type"         =        "IP,AT"
    "port_number"          =        "161"
    "display_name"         =        "SNMP/Trap Viewer"
</header>

<description>
\GB\Trap Viewer Probe\P\

This probe listens for trap packets to arrive and displays
the contents of the
trap in the Status Window. It does not actively poll the
device, nor does it
take any action based on the trap contents.

You can view all the variables that have been parsed from the
trap packet in the
device's Status Window. You can also use this as a prototype
```

for making your own  
trap probes.

\B\How the Trap Viewer Probe Works\p\

When a trap arrives, the probe parses the trap to get the values from the trap's header as well as the first ten items in its Varbind List. It assigns all these values to variables that can be used in the probe and displayed in the Status Window.

To see how this probe works, you can configure your equipment to send traps to InterMapper, or use the net-snmp \b\snmptrap\p\ command. Either way, the Status Window will show the values present in any traps that arrive.

For more information on the \b\snmptrap\p\ command, read the net-snmp documentation for the  
[\u2=http://www.net-snmp.org/tutorial/tutorial-4/commands/snmptrap.html](http://www.net-snmp.org/tutorial/tutorial-4/commands/snmptrap.html) and the  
[tutorial\p0\](http://www.net-snmp.org/docs/man/snmpinform.html) and the  
[command\0p\](http://www.net-snmp.org/docs/man/snmpinform.html). The remainder of this note shows how to send a trap with variables from the Dartware MIB:

\i\SNMPv1 Traps\p\

- a) Add a device to a map with the IP address  
 \i\192.168.56.78\p\
- b) Set it to use this probe
- c) Issue the snmptrap command below from the command line (it should all be on one line):

```
snmptrap -v 1 -c commString localhost
1.3.6.1.4.1.6306 192.168.56.78 6 123 4567890
1.3.6.1.4.1.6306.2.1.1.0 s "05/08 23:26:35"
1.3.6.1.4.1.6306.2.1.2.0 s Critical
1.3.6.1.4.1.6306.2.1.3.0 s "Big Router"
1.3.6.1.4.1.6306.2.1.4.0 s "Critical: High Traffic"
1.3.6.1.4.1.6306.2.1.5.0 s "127.0.0.1"
1.3.6.1.4.1.6306.2.1.6.0 s "SNMP Traffic Probe"
```



```
\i\SNMPv2c Traps\p\
```

- a) Add a device to the map with an IP address of  
`\i\localhost\p\`
- b) Set it to use this probe
- c) Issue the `snmptrap` command below from the command line (it should all be on one line)

```
snmptrap -v 2c -c commString localhost
4567890 1.3.6.1.4.1.6306
1.3.6.1.4.1.6306 192.168.56.78 6 123 4567890
1.3.6.1.4.1.6306.2.1.1.0 s "05/08 13:26:35"
1.3.6.1.4.1.6306.2.1.2.0 s Critical
1.3.6.1.4.1.6306.2.1.3.0 s "Big Router"
1.3.6.1.4.1.6306.2.1.4.0 s "Critical: High Traffic"
1.3.6.1.4.1.6306.2.1.5.0 s "127.0.0.1"
1.3.6.1.4.1.6306.2.1.6.0 s "SNMP Traffic Probe"
```

```
</description>
```

```
<!-- Copy/paste these lines into the terminal window for
testing...
```

```
snmptrap -v 1 -c commString localhost 1.3.6.1.4.1.6306
192.168.56.78 6 123
4567890 1.3.6.1.4.1.6306.2.1.1.0 s "05/08 13:26:35"
1.3.6.1.4.1.6306.2.1.2.0 s
Critical 1.3.6.1.4.1.6306.2.1.3.0 s "Big Router"
1.3.6.1.4.1.6306.2.1.4.0 s
"Critical: High Traffic" 1.3.6.1.4.1.6306.2.1.5.0 s
"127.0.0.1"
1.3.6.1.4.1.6306.2.1.6.0 s "SNMP Traffic Probe"
```

```
snmptrap -v 1 -c commString localhost 1.3.6.1.4.1.6306
192.168.56.78 6 123
4567890 1.3.6.1.4.1.6306.2.1.1.0 s "05/08 13:26:35"
1.3.6.1.4.1.6306.2.1.2.0 s
Critical 1.3.6.1.4.1.6306.2.1.3.0 s "Big Router"
1.3.6.1.4.1.6306.2.1.4.0 s
"Critical: High Traffic" 1.3.6.1.4.1.6306.2.1.5.0 s
"127.0.0.1"
1.3.6.1.4.1.6306.2.1.6.0 s "SNMP Traffic Probe"
1.3.6.1.4.1.6306.2.1.7.0 s
"var7" 1.3.6.1.4.1.6306.2.1.8.0 s "var8"
1.3.6.1.4.1.6306.2.1.9.0 s "var9"
1.3.6.1.4.1.6306.2.1.10.0 s "var10" 1.3.6.1.4.1.6306.2.1.11.0
s "var11"
1.3.6.1.4.1.6306.2.1.12.0 s "var12"
```

```
snmptrap -v 2c -c commString localhost 4567890
```

```

1.3.6.1.4.1.6306
1.3.6.1.4.1.6306.2.1.1.0 s "05/08 13:26:35"
1.3.6.1.4.1.6306.2.1.2.0 s Critical
1.3.6.1.4.1.6306.2.1.3.0 s "Big Router"
1.3.6.1.4.1.6306.2.1.4.0 s "Critical:
High Traffic" 1.3.6.1.4.1.6306.2.1.5.0 s "127.0.0.1"
1.3.6.1.4.1.6306.2.1.6.0 s
"SNMP Traffic Probe"
-->

-- The parameters in this probe are unused, but could be used
to
-- set thresholds for various alarms.
<parameters>
    "MinValue" = "10"
    "MaxValue" = "50"
</parameters>

<snmp-device-variables>

    -- TrapVariables are updated when a trap arrives.
    -- This set of variables comes from the Dartware MIB
    -- and would be sent in a trap from another copy of
Intermapper.

    trapTimeStamp,          1.3.6.1.4.1.6306.2.1.1.0, TRAPVARIABLE,
"Timestamp"
    DeviceStatus,          1.3.6.1.4.1.6306.2.1.2.0, TRAPVARIABLE,
"Status"
    DeviceDNS,             1.3.6.1.4.1.6306.2.1.3.0, TRAPVARIABLE,
"DNS Name of Device"
    DeviceCondition,       1.3.6.1.4.1.6306.2.1.4.0, TRAPVARIABLE,
"Condition String"
    TrapSourceAdrs,        1.3.6.1.4.1.6306.2.1.5.0, TRAPVARIABLE,
"Source of trap"
    ProbeType,             1.3.6.1.4.1.6306.2.1.6.0, TRAPVARIABLE,
"Probe that generated trap"

    -- Variables from the trap packet itself

    genericTrapVar,        $GenericTrap,          TRAPVARIABLE,
"Generic Trap"
    specificTrapVar,       $SpecificTrap,          TRAPVARIABLE,
"Specific Trap"
    timeStampVar,          $TimeStamp,             TRAPVARIABLE,
"Timestamp"
    enterpriseVar,         $Enterprise,            TRAPVARIABLE,
"Enterprise"

```

```

    commStringVar,      $CommunityString,  TRAPVARIABLE,
    "Community String"
    trapOIDVar,        $TrapOID,          TRAPVARIABLE,
    "Trap OID"
    agentAdrsVar,      $AgentAddress,      TRAPVARIABLE,
    "Address"
    senderAdrsVar,     $SenderAddress,     TRAPVARIABLE,
    "Sender Address"
    snmpVersionVar,    $SnmpVersion,      TRAPVARIABLE,
    "SNMP Version"
    varbindCountVar,   $VarbindCount,      TRAPVARIABLE,
    "Varbind Count"

    -- Positional names of Varbind List items

    vbVal1,            $VarbindValue1,      TRAPVARIABLE, "Value of
Varbind1"
    vbType1,           $VarbindType1,      TRAPVARIABLE, "Type of
Varbind1"
    vbOID1,            $VarbindOID1,       TRAPVARIABLE, "OID of
Varbind1"
    vbVal2,            $VarbindValue2,      TRAPVARIABLE, "Value of
Varbind2"
    vbType2,           $VarbindType2,      TRAPVARIABLE, "Type of
Varbind2"
    vbOID2,            $VarbindOID2,       TRAPVARIABLE, "OID of
Varbind2"
    vbVal3,            $VarbindValue3,      TRAPVARIABLE, "Value of
Varbind3"
    vbType3,           $VarbindType3,      TRAPVARIABLE, "Type of
Varbind3"
    vbOID3,            $VarbindOID3,       TRAPVARIABLE, "OID of
Varbind3"
    vbVal4,            $VarbindValue4,      TRAPVARIABLE, "Value of
Varbind4"
    vbType4,           $VarbindType4,      TRAPVARIABLE, "Type of
Varbind4"
    vbOID4,            $VarbindOID4,       TRAPVARIABLE, "OID of
Varbind4"
    vbVal5,            $VarbindValue5,      TRAPVARIABLE, "Value of
Varbind5"
    vbType5,           $VarbindType5,      TRAPVARIABLE, "Type of
Varbind5"
    vbOID5,            $VarbindOID5,       TRAPVARIABLE, "OID of
Varbind5"
    vbVal6,            $VarbindValue6,      TRAPVARIABLE, "Value of
Varbind6"
    vbType6,           $VarbindType6,      TRAPVARIABLE, "Type of
Varbind6"

```

```

    vbOID6,          $VarbindOID6,          TRAPVARIABLE, "OID of
Varbind6"
    vbVal7,          $VarbindValue7,        TRAPVARIABLE, "Value of
Varbind7"
    vbType7,         $VarbindType7,         TRAPVARIABLE, "Type of
Varbind7"
    vbOID7,          $VarbindOID7,          TRAPVARIABLE, "OID of
Varbind7"
    vbVal8,          $VarbindValue8,        TRAPVARIABLE, "Value of
Varbind8"
    vbType8,         $VarbindType8,         TRAPVARIABLE, "Type of
Varbind8"
    vbOID8,          $VarbindOID8,          TRAPVARIABLE, "OID of
Varbind8"
    vbVal9,          $VarbindValue9,        TRAPVARIABLE, "Value of
Varbind9"
    vbType9,         $VarbindType9,         TRAPVARIABLE, "Type of
Varbind9"
    vbOID9,          $VarbindOID9,          TRAPVARIABLE, "OID of
Varbind9"
    vbVal10,         $VarbindValue10,       TRAPVARIABLE, "Value of
Varbind10"
    vbType10,        $VarbindType10,        TRAPVARIABLE, "Type of
Varbind10"
    vbOID10,         $VarbindOID10,         TRAPVARIABLE, "OID of
Varbind10"
</snmp-device-variables>

<snmp-device-display>

\B5\Information about the Trap\0P\
  \4\CommunityString:\0\ $commStringVar
  \4\   Timestamp:\0\ $timeStampVar
  \4\   AgentAddress:\0\ $agentAdrsVar
  \4\   SenderAddress:\0\ $senderAdrsVar
  \4\   GenericTrap:\0\ $genericTrapVar \3IG\ (v1 only)
\P0M\
  \4\   SpecificTrap:\0\ $specificTrapVar \3IG\ (v1 only)
\P0M\
  \4\   Enterprise:\0\ $enterpriseVar \3IG\ (v1 only) \P0M\
  \4\   TrapOID:\0\ $trapOIDVar \3IG\ (v2c only) \P0M\
  \4\   SnmpVersion:\0\ $snmpVersionVar \3IG\ (0=SNMPv1;
1=SNMPv2c) \P0M\
  \4\   VarbindCount:\0\ $varbindCountVar \3IG\ (total number
of Varbinds) \P0M\

\B5\Varbind List Items parsed by OID\0P\

```

```
\4\      TimeStamp:\0\    $trapTimeStamp \3IG\ \POM\  
\4\    Device Status:\0\    $deviceStatus \3IG\ \POM\  
\4\      Device DNS:\0\    $deviceDNS \3IG\ \POM\  
\4\Condition String:\0\    $deviceCondition \3IG\ \POM\  
\4\Trap Source Adrs:\0\    $TrapSourceAdrs \3IG\ \POM\  
\4\      Probe Type:\0\    $ProbeType \3IG\ \POM\  
  
\B5\Varbind List Items by Position\0P\ \3IG\ (Varbind Value /  
Varbind Type / Varbind OID) \POM\  
\4\ VarBindList #1:\0\    $vbVal1 / $vbType1 / $vbOID1  
\4\ VarBindList #2:\0\    $vbVal2 / $vbType2 / $vbOID2  
\4\ VarBindList #3:\0\    $vbVal3 / $vbType3 / $vbOID3  
\4\ VarBindList #4:\0\    $vbVal4 / $vbType4 / $vbOID4  
\4\ VarBindList #5:\0\    $vbVal5 / $vbType5 / $vbOID5  
\4\ VarBindList #6:\0\    $vbVal6 / $vbType6 / $vbOID6  
\4\ VarBindList #7:\0\    $vbVal7 / $vbType7 / $vbOID7  
\4\ VarBindList #8:\0\    $vbVal8 / $vbType8 / $vbOID8  
\4\ VarBindList #9:\0\    $vbVal9 / $vbType9 / $vbOID9  
\4\VarBindList #10:\0\    $vbVal10 / $vbType10 / $vbOID10  
</snmp-device-display>
```

# The Dartware MIB

HelpSystems, LLC has registered the Enterprise 6306 for its own SNMP variables. The remainder of this page shows the Dartware MIB in ASN.1 notation.

```
--
*****
-- DARTWARE-MIB for Intermapper and other products
--
-- May 2007
--
-- Copyright© HelpSystems, LLC
-- All rights reserved.
--
*****

DARTWARE-MIB DEFINITIONS ::= BEGIN

    IMPORTS
        MODULE-IDENTITY, OBJECT-TYPE, NOTIFICATION-TYPE,
enterprises
        FROM SNMPv2-SMI
        DisplayString
        FROM SNMPv2-TC;

    dartware MODULE-IDENTITY
        LAST-UPDATED "200507270000Z"
        ORGANIZATION "Dartware, LLC"
        CONTACT-INFO "Dartware, LLC
            Customer Service
            Postal: PO Box 130
            Hanover, NH 03755-0130
            USA
            Tel: +1 603 643-9600
            E-mail: support@dartware.com"

        DESCRIPTION
            "This MIB module defines objects for SNMP traps
sent by
Intermapper."

        REVISION      "200705300000Z"
        DESCRIPTION
```

```
        "Updated descriptions to show timestamp format,
correct strings for intermapperMessage."
```

```
    REVISION      "200512150000Z"
```

```
    DESCRIPTION
```

```
        "Added intermapperDeviceAddress and
intermapperProbeType."
```

```
    REVISION      "200507270000Z"
```

```
    DESCRIPTION
```

```
        "First version of MIB in SMIV2."
```

```
 ::= { enterprises 6306 }
```

```
notify          OBJECT IDENTIFIER ::= { dartware 2 }
```

```
intermapper     OBJECT IDENTIFIER ::= { notify 1 }
```

```
intermapperTimestamp OBJECT-TYPE
```

```
    SYNTAX          DisplayString (SIZE(0..255))
```

```
    MAX-ACCESS      read-only
```

```
    STATUS          current
```

```
    DESCRIPTION
```

```
        "The current date and time, as a string,
in the format 'mm/dd hh:mm:ss'."
```

```
 ::= { intermapper 1 }
```

```
intermapperMessage OBJECT-TYPE
```

```
    SYNTAX          DisplayString (SIZE(0..255))
```

```
    MAX-ACCESS      read-only
```

```
    STATUS          current
```

```
    DESCRIPTION
```

```
        "The type of event - Down, Up, Critical,
Alarm, Warning, OK, or Trap - as a string."
```

```
 ::= { intermapper 2 }
```

```
intermapperDeviceName OBJECT-TYPE
```

```
    SYNTAX          DisplayString (SIZE(0..255))
```

```
    MAX-ACCESS      read-only
```

```
    STATUS          current
```

```
    DESCRIPTION
```

```
        "The (first line of the) label of the
device as shown on a map, as a
string."
```

```

        ::= { intermapper 3 }

intermapperCondition OBJECT-TYPE
    SYNTAX          DisplayString (SIZE(0..255))
    MAX-ACCESS      read-only
    STATUS          current
    DESCRIPTION
        "The condition of the device, as it would
be printed in the log file."
    ::= { intermapper 4 }

intermapperDeviceAddress OBJECT-TYPE
    SYNTAX          DisplayString (SIZE(0..255))
    MAX-ACCESS      read-only
    STATUS          current
    DESCRIPTION
        "The device's network address, as a
string."
    ::= { intermapper 5 }

intermapperProbeType OBJECT-TYPE
    SYNTAX          DisplayString (SIZE(0..255))
    MAX-ACCESS      read-only
    STATUS          current
    DESCRIPTION
        "The device's probe type, as a human-
readable string."
    ::= { intermapper 6 }

-- For SMIV2, map the TRAP-TYPE macro to the
corresponding NOTIFICATION-TYPE macro:
--
--
-- intermapperTrap TRAP-TYPE
--     ENTERPRISE      dartware
--     VARIABLES       { intermapperTimestamp,
intermapperMessage,
--                     intermapperDeviceName,
intermapperCondition }
--     DESCRIPTION
--         "The SNMP trap that is generated by
Intermapper as a notification option."
--     ::= 1

```



```
    intermapperNotifications OBJECT IDENTIFIER ::= {
intermapper 0 }

    intermapperTrap NOTIFICATION-TYPE
        OBJECTS { intermapperTimestamp, intermapperMessage,
                    intermapperDeviceName,
intermapperCondition,
                    intermapperDeviceAddress,
intermapperProbeType }
        STATUS current
        DESCRIPTION
            "The SNMP trap that is generated by Intermapper
as a notification option."
        ::= { intermapperNotifications 1 }

END
```

# TCP Probes

```
type="tcp-script"
```

TCP Probes connect to the specified device and port, then execute a script that sends and receives data from the device. Intermapper examines the responses, and sets the device's status and condition based on the results.

For example, the HTTP probe connects to the specified port, then issues the commands of an HTTP request to send data to the web server, and verifies the received data. If the response is not as expected, the probe sets the device into alarm or warning.

As another example, the [TCP Example](#) shows another TCP-based probe that connects to a device. It then sends the specified string, waits several seconds and checks the response to determine the device state.

The Custom TCP probe is shown in full as an example, and can be used as the basis for making your own probes.

## Common Sections of TCP Probes

Each TCP probe follows the same general format as other probe files.

- The [<header>](#) section of a command-line probe specifies the probe type, name, and a number of other properties fundamental to the operation of the probe.
- The [<description>](#) section specifies the help text that appears in the Set Probe window. Format the description using IMML, [Intermapper's Markup language](#).
- The [<parameters>](#) section defines the fields presented to the user in the Set Probe window.

## Sections Specific to TCP Probes

Each TCP probe also has:

- Use the [<script>](#) section of a TCP probe to define a sequence of commands the probe uses to interact with and query a device, and to interpret the responses from the device. The [<script>](#) section uses the [TCP Probe Scripting Language](#), a sequential language with a [rich set of commands](#).
- The [<script-output>](#) section of a TCP probe file formats the information retrieved from the device and sends it to the device's Status window. Format the script output using IMML, [Intermapper's Markup language](#).

Intermapper's TCP Probes establish a connection to a remote system, exchange commands and receive responses, and then set the status of the device based on those responses.

This note describes how probe writers can use *regular expressions* and *comparisons* to parse out information from the responses.

# The Overall Process

There is an annotated FTP probe in the [Developer Guide](#). This gives an overview of the script language and shows how it connects and logs into a FTP server, how a script can respond to error conditions, and how to set the device's status based on those conditions.

# Regular Expressions

The TCP Script Language uses the MTCH command to compare a response string to expected values. It can also use a regular expression to match on a part of a string. For example:

```
MTCH "A([BCD]+)E"r else goto @NOMATCH
STOR "testval" "${1}"
```

If the incoming line contains ABDE, then the "testval" variable will contain "BD".

In the MTCH regular expression, enclosing something in parentheses turns it into a capturing subgroup. The one or more Bs, Cs or Ds that it's matching will be stored in the \${1} variable. If you have several capturing groups, they get stored in \${2}, \${3}, etc.

For more information, see [the Regular Expressions examples](#) in Probe Calculations.

# Calculations in Scripts

To perform calculations within a TCP script, you should use [the EVAL command](#). Its argument is an expression (in quotes) that will be evaluated. It usually contains an assignment (with the ":=" operator), that sets a variable to the result of the expression. For example:

```
EVAL $celsius := (($fahrenheit - 32) * 5 / 9)
```

will set the variable \$celsius to the temperature that corresponds to the \$fahrenheit variable. The value of the \$celsius variable can be used in subsequent statements.

# Comparisons in Scripts

You can use the EVAL statement to make comparisons between either strings or numeric (either integer or floating point) values. To do this, write an EVAL statement that compares the two values and set the result in a new variable. If the comparison was true, then the resulting variable will be set to 1, otherwise it will be zero.

Here are examples of comparing numeric and string values:

## Comparing Numeric Values

```
EVAL $x := ($val >
50.5)
NBNE #$x #0 @greater

@less:
...
GOTO @ENDIF
@greater:
...
GOTO @ENDIF
@ENDIF:
```

## Comparing String Values

```
EVAL $x := ("dog" >
"cat")
NBNE #$x #0 @dog

@cat:
...
GOTO @ENDIF

@dog:
...
GOTO @ENDIF

@ENDIF:
```

For more information, see The [Eval Macro section](#) of Built-in Custom Probe Variables.

# Simple Comparisons in Scripts

Intermapper TCP Scripts can compare two string or integer numeric values and branch based on the results. The commands below are no longer preferred as the EVAL statement described above is equally simple and more powerful.

The SBNE ("String Branch Not Equal") compares the two *string* values and branches if they are not equal. One or both of the arguments can be variables, expressed as `${variable-name}`.

The NBNE ("Numeric Branch Not Equal") and NBGT ("Numeric Branch Greater Than") compares two *numeric* values, branching on the result. The arguments to these commands are strings and are expected to be within quotes. To convert a string to a numeric value, place a number sign (#) before the parameter. For example:

```
STOR "val1" "100"  
STOR "val2" "50"  
NBGT #${val1} #${val2} @exit
```

In this example, the string `${val1}` will be converted to the numeric value 100, and `${val2}` will be converted to the numeric value 50, and the branch will be taken, because 100 is greater than 50.

**Note:** The NBGT, NBNE and other TCP probe commands expect integer arguments only (with an optional + or -). A script parses up to the first non-digit character. Thus, the value of "50.5" is 50; the remaining digits are ignored. If you wish to compare against a fraction or floating point value, use the EVAL statement described above.

These commands are described in detail in the [TCP Probe Command Reference](#) topic.

# The <script> Section

Use the <script> section of a TCP probe to define a sequence of commands the probe uses to interact with and query a device, and to interpret the responses from the device. The <script> section uses the TCP Probe Scripting Language, (described below) a sequential language with a [rich set of commands](#).

```
<script>
...
</script>
```

## TCP Probe Scripting Language

Use the Intermapper TCP Probe Scripting language to create custom probes. You can use script statements to send data to the device being tested, to examine responses from that device, and to return a status based on the response. To view a TCP Probe script example, see [Example TCP Probe File](#).

- [Script Process Flow](#)
- [Script Command Format](#)
- [String Argument Format](#)
- [String Matching](#)
- [Numeric Argument Format](#)
- [Using Labels for Program Control](#)
- [Using Variables](#)
- [Handling Script Failures](#)
- [Adding Comments](#)

## Script Process Flow

Each probe has a common process flow. It sends data (as a datagram or over a TCP connection) to the device to be tested, then examines any responses. Based on responses, the probe sets the device status (*UP*, *DOWN*, *CRITICAL*, *ALARM*, *WARN*, *OK*). It also sets a *condition string*, which contains a text description of the state.

## Script Command Format

All script command keywords have the following requirements:

- All commands are 4 letters long.
- All commands are case-sensitive.

- All commands MUST be in UPPER CASE.
- There must be white space between a command and each argument. You can include other text (e.g. comments) after the first argument, as long as it is separated by white space from the remaining arguments.

### Example:

The **MTCH** command has the format

```
MTCH "string" #fail
```

The command statement

```
MTCH "blah" else goto #7
```

is treated exactly the same as

```
MTCH "blah" #7
```

When parsing the statement, Intermapper ignores the "else goto" part. This allows you to include comments to make the behavior of the script more obvious. This extraneous text does not have to be in uppercase.

## String Argument Format

Some commands take string arguments.

- String arguments must be enclosed in double-quotes.

### Example:

```
"This is a string"
```

## Special Characters

The following special characters may be included by using a backslash escape code:

|                 |                 |
|-----------------|-----------------|
| <code>\r</code> | Carriage Return |
| <code>\n</code> | Unix Linefeed   |
| <code>\t</code> | Horizontal Tab  |
| <code>\f</code> | Formfeed        |
| <code>\b</code> | Backspace       |
| <code>\v</code> | Vertical Tab    |

|                   |                        |
|-------------------|------------------------|
| <code>\a</code>   | Alert (bell) Character |
| <code>\"</code>   | Double Quote           |
| <code>\\</code>   | Backslash              |
| <code>\ooo</code> | Octal Number           |
| <code>\xhh</code> | Hexadecimal Number     |

**Special Character Example:**

```
"\tThis sentence is preceded by a tab, and followed by a
carriage return and linefeed.\r\n"
```

## String Matching

The MTCH and EXPT commands both specify a string to match. When specifying the string, you can use regular expressions. See Wildcard Matching, below.

## Controlling Case-Sensitivity

- By default, string-matching is case-sensitive.
- Place an 'i' after the final quote if you want the matching to be case-insensitive.

**Examples:**

"fred" matches only "fred".

"fred"i matches "fred", "FRED", or "FrEd".

## Wild-card Character Matching

In some cases, it is convenient to match a more general pattern. You can use simple regular expressions to match patterns and place them into variables.

To use regular expressions in MTCH and EXPT:

- Place an 'r' after the closing quote of the match string to indicate that the contents of the string is a regular expression.
- Place an 'i' after the closing quote of the match string to indicate that the match is case-insensitive.
- An expression inside round brackets (parentheses) creates a match group and places matched text within a numbered variable. The first variable is \${1}, the second is \${2}, and so on.



- A subsequent MTCH or EXPT command resets the variables, so you should make a copy of the contents into another variable after a match. See the example below.

### Simple Example:

"red"r matches "fred", "Fred", "tred", "bred", etc. It does not match "freD" unless you include the "i" after the string.

### More Complex Example:

Given the following returned data:

```
"var1=12 var2=1234.00 var3=45"
```

You match and store each data variable into an Intermapper variable:

```
MTCH m"var1=([0-9]+)"i else goto +1 (skip the next STOR line)

STOR "var1" "${1}"
MTCH m"var2=([0-9]+)"i else goto @BLAH
STOR "var2" "${1}"
MTCH m"var3=([0-9]+)"i else goto @BLAH
STOR "var3" "${1}"
```

Note: True Regex groups and the Alternate operator (|) are not supported.

## Numeric Argument Format

Some commands take numeric arguments.

- Numeric arguments are formed using a # sign followed by digits.

### Example:

```
WAIT #30
```

## Using Numeric Arguments with the GOTO Command

In many cases, numeric arguments are used to specify the script statement number to go to when a failure occurs. A special notation allows you to express these jumps as relative offsets.

- Include a sign ('+' or '-') after the # to express a relative offset from the current statement.

### Example:

```
GOTO #+2
```

## Default Values and Script Termination

- If a command takes a numeric argument, but you do not include it, the default value is 0.
- If you specify 0 as the statement to goto when the script fails, the script is terminated with a *DOWN* condition.

## Using Labels for Program Control

Use a label as script marker to which you can jump from elsewhere in the script.

Labels take the form:

```
@label_name
```

- Labels must be alone on a line.

**Example:**

```
@IDLE
```

## Jumping to a Label

Use the *GOTO* command to jump to a label.

**Example:**

```
WAIT #30 seconds else goto @IDLE
```

## Using Relative Offsets to Transfer Control

You can specify an offset for the *GOTO* command

Specify a offset (in statements) "#+n" or "#-n" to jump forward or backward *n* statements (respectively).

**Example:**

```
MTCH "${WARN Response}" else #+2
```

## Using Variables

You can substitute variables in a script statement before the statement is processed.

- Variables names and their default values can be defined in the <parameter> section of the probe file, or by using the STOR, NADD, or TIME command.
- Variable names are preceded by a dollar sign (\$), and are enclosed by curly braces.
- Variable names are case-insensitive.
- See the the [Built-in Variable Reference](#) topic for detailed information on variable usage.

### Example:

`${Password}` and `${password}` are treated as the same variable

## Built-in Macros

A macro is an expression that modifies an input string to produce another string. The built-in macros are:

---

`${_LINE:<line>}` The first <num> characters of the last line received.

---

`${_BASE64:<param>}` The Base-64 encoding of the string that follows the ":".

---

`${_CVSPASSWORD:<param>}` The value of <param> encoded for use as a password over the CVS pserver protocol.

---

## Handling Script Failures

Certain script commands may fail, either because they are malformed or because an unexpected situation occurs. For example, the script could jump to a non-existent command, it could fail to match a string it expects, or an unexpected disconnection could occur. In each case, the script immediately branches to a failure handler in the script. Each command that can fail takes the statement number of the failure statement as a numeric argument. If this number is omitted, the script will terminate with a "DOWN" status.

### Example:

In the following example, the MTCH command succeeds if the incoming line of data contains "220". If the command fails, the script branches to statement 3.

```
MTCH "220" ELSE #3
```

**Note:** If the script is idle for too long, it may go to a special "idle" handler. See the [WAIT command](#) for more details.

# Adding Comments to your Script

There are two ways to add comments to your script:

- Add text between or after arguments to a script command.
- Add a comment using the InterMapper probe file comment format.

## Adding text within a command line

You can add text between arguments in a command line, as well as adding text after the line.

### Examples:

The following statements all have exactly the same effect:

```
MTCH "331 " #14
MTCH "331 " else #14
MTCH "331 " else goto -1- #14 -- Unexpected or unknown
response to USER command
```

## Adding text in Comment format

Use the HTML comment syntax to add comments to a probe files. Place comments anywhere in a probe file. HTML comment syntax can be simplified by following this rule:

- Begin a comment with "<!--", end it with "-->", and do not use "--" within the comment.

### Example:

```
<!-- This text is treated as a comment, and will be ignored -
->
```

# The <script-output> Section

The <script-output> section of a TCP probe file formats the information retrieved from the device and sends it to the device's Status window. Format the script output using IMML, [Intermapper's Markup language](#).

# Measuring TCP Response Times

You can measure the response time of a device as it is being tested by a TCP probe. Times are measured in milliseconds.

With TCP Probes, Intermapper measures both the *time to establish the connection* and the *time for various portions of an interaction*. These times can be charted and logged.

## Time Measurement Probe Variables

TCP Timers are:

|                                       |                           |  |
|---------------------------------------|---------------------------|--|
| <b>Connection initiation interval</b> | <code>\${_connect}</code> | Records the time required to establish a connection.                                     |
| <b>Connection duration interval</b>   | <code>\${_active}</code>  | Records the duration from the connection request until the end of the end of the script. |

## TCP Script Commands

Intermapper supports two commands for measuring intervals during a script. These are:

|                                |   |
|--------------------------------|---|
| <b>STRT</b>                    | Starts the probe's custom timer.  |
| <b>TIME<br/><i>varname</i></b> | Sets the variable named <code>\${varname}</code> to the milliseconds elapsed since the customtimer was started. |

## The <script-output> Section

Use the optional <script-output> section to display the results of custom TCP probes. The data in this section appears in a Status window when you click and hold on the device. The format of this section is the same as the <snmp-device-display>, described in [Customized Status Windows](#).

Use the `${_connect}` and `${_active}` variables, as well as any variables set with the TIME *varname* command, in the <script-output> section of the Status window.

## A Note on Accuracy

Intermapper uses different techniques to measure the round-trip times of various probes.

- **Pings (ICMP and AppleTalk echoes)** - These are the most accurate timings. Intermapper detects the arrival of the Ping response at the moment it arrives. Thus, it can compute the response times with millisecond accuracy.
- **Other UDP-based and TCP-based probes** - These timings are computed by Intermapper as it does its normal polling. Thus, the measured time can be affected slightly by the such things as the number of devices probed and other various other tasks, as they may affect how long it takes Intermapper to execute a single round of polling.

# Example TCP Probe File

The following is the HelpSystems-provided probe for the Custom TCP script.

```
<!--
Custom TCP (com.dartware.tcp.custom)
Copyright© HelpSystems, LLC.
Please feel free to use this as the basis for new probes.
-->

<header>
  type = "tcp-script"
  package = "com.dartware"
  probe_name = "tcp.custom"
  human_name = "Custom TCP"version = "1.2"
  address_type = "IP"
  port_number = "23"
</header>

<description>
\GB\Custom TCP Probe\P\
This probe lets you send your own string over the TCP
connection and set the
status of the device depending on the response received.
There are six
parameters which control the operation of this probe:
\i\String to send\p\ is the initial string sent over the TCP
connection. This
could be a command which indicates what to test, or a
combination of a command
and a password. The string is sent on its own line,
terminated by a CR-LF.

\i\Seconds to wait\p\ is the number of seconds to wait for a
response. If no
response is received within the specified number of seconds,
the device's status
is set to DOWN.

\i\OK Response\p\ is the substring which should match the
device's "ok
response". If it matches the first line received, the device
is reported to have
a status of OK.

\i\WARN Response\p\ is the substring which should match the
device's warning
response.
```



\i\ALRM Response\p\ is the substring which should match the device's alarm response.

\i\DOWN Response\p\ is the substring which should match the device's down response.

If Intermapper cannot connect to the specified TCP port, the device's status is set to DOWN.

</description>

<parameters>

"String to send" = ""  
"Seconds to wait" = "30"  
"OK Response" = ""  
"WARN Response" = ""  
"ALRM Response" = ""  
"DOWN Response" = ""

</parameters>

<script>

CONN #60 (connect timeout in secs)  
SEND "\${String to send}\r\n"  
WAIT \${Seconds to wait} else goto @IDLE  
EXPT "."r else goto @DISCONNECT  
MTCH "\${OK Response}" else #+2  
DONE OKAY "[Custom] Response was \"\${\_LINE:50}\"."  
MTCH "\${WARN Response}" else #+2  
DONE WARN "[Custom] Response was \"\${\_LINE:50}\"."  
MTCH "\${ALRM Response}" else #+2  
DONE ALRM "[Custom] Response was \"\${\_LINE:50}\"."  
MTCH "\${DOWN Response}" else #+2  
DONE DOWN "[Custom] Response was \"\${\_LINE:50}\"."

@IDLE:

DONE DOWN "[Custom] Did not receive a line of data within  
\${Seconds to wait}  
seconds. [Line \${\_IDLELINE}]"

@DISCONNECT:

DONE DOWN "[Custom] Connection disconnected before a full  
line was received."

</script>

<script-output>

\B5\Custom TCP Information\0P\  
\4\Time to establish connection:\0\ \${\_connect} msec

```
\4\Time spent connected to host:\0\ ${_active} msec  
</script-output>
```

# Command Line Probes

```
type="cmd-line"
```

Intermapper provides the ability to run a *command-line* probe, a script or program (written in perl, C, C++, or your favorite language.) Your program's return value becomes the device's status on the Intermapper map.

## Common Sections of a Command-Line Probe

Each command-line probe follows the same general format as other probe files, sharing these common sections:

- The [<header>](#) section of a command-line probe specifies the probe type, name, and a number of other properties fundamental to the operation of the probe.
- The [<description>](#) section specifies the help text that appears in the Set Probe window. Format the description using IMML, [Intermapper's Markup language](#).
- The [<parameters>](#) section defines the fields presented to the user in the Probe Configuration window.

## Sections Specific to Command-line Probes

Each command-line probe also has:

- The [<command-line>](#) section - defines the command-line, specifying the path to the executable, the command to execute, and any arguments to the command.
- The [<command-exit>](#) section - controls how the device's state is set, based on the results of the command.
- The [<command-display>](#) section - controls what appears in the device's Status window.

Intermapper uses the information in the probe's `<command-line>` section to invoke the program or script and pass arguments to it. Intermapper sets the device's status based on the return code from the program or script. In addition, any data written to the script's standard output file is used as the device's reason string, and appears in the status window. The total amount of data that can be returned by the program, including return code, reason string, and additional values, is 64k.

Intermapper's command-line probes are similar to [Nagios® plugins](#). You can see the [standard set of Nagios plugins](#). Many vendors and individuals have created their own Nagios plugins. You will have to download the Nagios plugins and build/compile them yourself.

If you wish to develop your own command-line probes, we recommend you follow the [developer guidelines for Nagios](#). This will result in probes/plugins that work for both Intermapper and Nagios.

For more information about Intermapper and Nagios Plugins, see the [Nagios Plugins page](#).

See [Command Line Probe Example](#) for a sample shell script and corresponding probe.

## The <tool> section - embedding a companion script

You can also embed script code directly in a probe. This provides an easy way to deliver a command-line probe and a script that it runs in a single probe file, ensuring that the version of the script matches the version of the probe. WMI probes provide a number of good examples of companion scripts. For more information, see [The <tool> Section](#).

## Command Line Script API

When Intermapper invokes a command line program or script, it passes parameters on the command line. Use the `path`, `cmd`, and `arg` properties of the `<command-line>` section to specify the script or other executable to invoke, and any arguments to the command. As the script developer, you are responsible for parsing the arguments.

The script can return three kinds of information to Intermapper:

1. The operating system return code, or *exit code*, is used to indicate the success/failure/severity. This will be handled by the `<command-exit>` section of the probe file.
2. The script can optionally return *additional values*, such as measurements, discovered during execution. It does this by writing to the script's *stdout*. You return these values as a comma-separated list enclosed in "{ ... }" characters. These values can then be handled as variables in the probe's `<command-display>` section. The values themselves are *name-value* pairs in the form:

`<name> := <value>`

3. The script can also return a **reason string** that will be used to explain the device's condition. You specify the reason string by writing to the script's *stdout*. This text should follow the closing "}" of any additional values.

**Example:** The following output from a script sets two values to the probe:

`$rtt` and `$hop`, and sets the device's reason string to *"Round-trip time is very high"*.

```
\{ $rtt := 5, $hop := 2 } Round-trip time is very high
```

You can do a significant amount when writing to stdout, using the `${^stdout}` variable. For more information, see [The `\${^stdout}` variable and the Reason string](#).

## Installing a Command-line Probe

Once you have created your probe, you need to install it before you can test it.

To install and use a command-line probe:

1. If you are using an external script or other executable, create the program, and make it runnable. If it's a Perl, Python, or other script, set the permissions so that it can run from the command line. If it's written in C, C++, or other non-interpreted language, compile the source and then place the resulting binary in an appropriate directory. (See the [path](#) discussion below.)  
**Note:** If you embed a script in the `<tool>` section of the probe, permissions are set by Intermapper when it writes the script to the Tools directory (when you import or reload the probe.)
2. Create a Command-line probe that references the executable program or contains the script in the `<tool>` section.
3. Import or reload the probe (from the Set Probe window) to make it available.

See [Command Line Probe Example](#) for a sample shell script and corresponding probe.

## Passing parameters to a command-line probe

Pass arguments to the command line into the probe by accessing the parameter variables with `${parametername}`. The named arguments can be added to the command line.

For example, use `${Timeout}` for an parameter as follows:

```
<parameters>
  Timeout = "7"
</parameters>
```

The `arg` variable could be set as follows:

```
arg = "-H ${Timeout}"
```

**Note:** Depending on the nature of the parameters you are passing, you may want to pass the parameters through STDIN, as described below.

## Sending Data to STDIN

Using the `ps` command on Unix systems, or using the Task Manager or other utility programs on Windows systems, it is possible to see the command line arguments. This represents a security vulnerability. Use the `input` property of the `<command-line>` section to pass sensitive data to `STDIN`, removing this vulnerability. See [Sending Data to STDIN](#) in *The <command-line> Section* for a detailed example.

# The <command-line> Section

The <command-line> section allows you to specify the information needed to execute the commands for the probe. Use the following variables:

- **path** - specify the path to the executable script/command.
- **cmd** - specify the actual script/command.
- **arg** - specify the arguments to be passed to the script/command.
- **input** - specify information to pass to STDIN to the script/command.

## The Properties of the <command-line> Section

- Use the `path` property to specify the directories in which Intermapper should look for the executable to run as a probe. This is the only path Intermapper will use; the `PATH` environment variable is not used. The `path` property follows the conventions for the `PATH` environment variable on the system hosting Intermapper. The example below is for Unix or Mac OS X. A path for Windows would use "\" instead of "/" and ";" instead of ":".

### Notes:

- If no path is specified, *Intermapper Settings/Tools* is used as the path.
  - On Unix systems, it might be possible to see the command line arguments in the 'ps' listing. This represents a security vulnerability. Use the `input` variable to pass values to stdin, removing this vulnerability. For more information, see [Sending Data to STDIN](#), below.
- Use the `cmd` property to specify the executable you wish to run. In the example below, this is "check\_ping". Note that you need to specify the exact name, including any extensions such as .exe or .cmd. You may also specify arguments as part of the `cmd` property if you'd like.
  - Use the `arg` property to specify arguments to the executable. This may be instead of or in addition to specifying them in the `cmd` property. We could have just as easily written our sample `cmd` property as a command and argument, like this:

```
<command-line>
  path = ""
  cmd  = "check_ping"
  arg  = "-H ${ADDRESS} -w 100,10% -c 1000,90%"
</command-line>
```

- Use the `input` property to pass information to STDIN. See [Sending Data to STDIN](#), below.

Note the use of the "\${ADDRESS}" macro. This is replaced with the address given when the device was created. You can also use the "\${PORT}" macro to indicate the port given when the device was created.

## Sending Data to STDIN

Using the `ps` command on Unix systems, or using the Task Manager or other utility programs on Windows systems, it is possible to see the command line arguments. This represents a security vulnerability. Use the `input` variable to pass sensitive data to stdin, removing this vulnerability.

This mechanism provides a less visible channel for sensitive communication to a probe script. Usernames, passwords, and SSL pass-phrases are likely candidates for this technique.

### Example:

```
<command-line>
  cmd = "executable"
  input = "${User} ${Password}"
</command-line>
```



# The <command-exit> Section

## The <command-exit> Section

The <command-exit> section allows you to specify which results from the command indicate the five Intermapper device states. The states are:

- down
- critical
- alarm
- warning
- okay

For each state, you need to indicate what item Intermapper should examine and what its value should be to result in that state being set. At the moment, the only thing Intermapper can look at is the exit code, which is indicated with `${EXIT_CODE}`. So, in the example below, the line:

```
down: ${EXIT_CODE} = 2
```

means, "To determine if the device is down, examine the exit code from the command; if it is 2, the device is down." If none of the criteria for the states you have defined are true, then the device is set to "unknown".

# The <command-display> Section

The <command-display> section displays variables in the device's Status window using the same form as the output section of other probe types. If the plugin returns a non-integer value, you should use the `#{chartable:...}` macro to display digits to the right of the decimal point. As with other probe types, you format the appearance of the output using IMML, [Intermapper's markup language](#).

See [Command Line Probe Example](#) for a sample shell script and corresponding probe.

# The <tool> Section

Use the <tool> section of a command-line probe to embed a script's code directly into a command-line probe. The <tool> section provides a convenient way to maintain the probe and the script in a single file.

```
<tool:scriptname>
  [script code]
</tool:scriptname>
```

Replace ***scriptname*** with the executable you want to use for the script.

Replace ***[script code]*** with the code of the script.

## What happens when you load a probe with a <tool> section?

When the Intermapper server starts or when you import or reload a probe, if a tool section appears for a probe, Intermapper creates a subdirectory of Tools with the canonical name of the probe and writes the script to that subdirectory, using `scriptname` as a file name.

If a subdirectory of that name already exists, all non-hidden files are deleted before the script is written out. For this reason, you should not edit scripts directly in the subdirectories of the Tools directory, since they are overwritten when probes are reloaded.

For example, given the trivial example of a command-line probe where the canonical name is `com.dartware.cmdline.test`, where the `cmd` clause in the <command-line> section is:

```
cmd="python test.py"
```

or, using the `${PYTHON}` macro:

```
cmd="${PYTHON} test.py"
```

and the tool section is:

```
<tool:test.py>
  # Trivial example
  print "okay"
  raise SystemExit, 0
</tool:test.py>
```

When Intermapper starts or reloads probes, a subdirectory of Tools named `com.dartware.cmdline.test` is created if it doesn't exist, and (in this case) a file

named "test.py" is written into it, containing the text between <tool:test.py> and </tool:test.py>.

The WMI probes provide a number of good examples of this feature.

## Calling external scripts and other executables

While using the <tool> section is recommended, it is optional. You can call an external script or other executable by providing the correct path to it in the `cmd` property of the <command-line> section of the probe. If you provide a paths to multiple directories in the `path` parameter, Intermapper looks in the specified directories for the executable. The <tool> section is appropriate only for scripts, not for compiled programs.

## Python Example

```
<!--
check_connect
(com.dartware.commandline.check_connect.txt)
Copyright© HelpSystems, LLC. All rights reserved.
-->

<header>
  type = "cmd-line"
  package = "com.dartware"
  probe_name = "commandline.check_connect"
  human_name = "Check Connect"
  version = "1.1"
  address_type = "IP"
  display_name = "Miscellaneous/Test/Check Connect"
</header>

<description>
  \GB\Check for connect\p\

  This probe checks to see if you can connect to the given
  address and port.
</description>

<parameters>
  "CHECK_PORT" = "80"
</parameters>

<command-line>
  cmd=${PYTHON}
  arg="check_connect.py ${ADDRESS} ${CHECK_PORT}"
```

```
</command-line>

<command-data>
  -- Currently unused.
</command-data>

<command-exit>
  -- These are the exit codes used by Nagios plugins
  down: ${EXIT_CODE}=4
  critical: ${EXIT_CODE}=3
  alarm: ${EXIT_CODE}=2
  warn: ${EXIT_CODE}=1
  okay: ${EXIT_CODE}=0
</command-exit>

<command-display>
</command-display>

<tool:check_connect.py>
  import sys
  import socket

  # constant return codes for Intermapper
  OKAY = 0
  WARNING = 1
  ALARM = 2
  CRITICAL = 3
  DOWN = 4

  retcode = OKAY
  output = ""

  try:
    host = sys.argv[1] # The remote host
    port = long(sys.argv[2]) # The port
  except:
    print "Usage: check_connect HOST PORT"
    sys.exit(DOWN)

  try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.close()
  except IOError, e:
    retcode = DOWN
    if hasattr(e, 'reason'):
      reason = 'Reason: ' + e.reason
```

```
elif hasattr(e, 'code'):
    reason = str(e.code)
else:
    reason = "unknown"

    output = "Error (" + reason + ") connecting to " + str
(host) + ":" + str(port)

    print output
    sys.exit(retcode)
</tool:check_connect.py>
```

## Cscript Example

```
<!--
Check Web
Copyright© HelpSystems, LLC. All rights reserved.
-->

<header>
    type = "cmd-line"
    package = "com.dartware"
    probe_name = "commandline.check_web"
    human_name = "Check Web"
    version = "1.1"
    address_type = "IP"
    display_name = "Miscellaneous/Test/Check Web"
    visible_in = "Windows"
</header>

<description>
    \GB\Check Web\p\

    Given an address or hostname, attempts to connect to a web
    server.
</description>

<parameters>
</parameters>

<command-line>
    -- Empty path forces the Intermapper Settings:Tools
    directory
    path=
    cmd="${CSCRIPT} check_web.vbs"
```

```
    arg="{address}"
    timeout = ${Timeout (sec)}
</command-line>

<command-data>
    -- Currently unused.
</command-data>

<command-exit>
    down:${EXIT_CODE}=4
    critical:${EXIT_CODE}=3
    alarm:${EXIT_CODE}=2
    warning: ${EXIT_CODE} = 1
    okay:${EXIT_CODE}=0
</command-exit>

<command-display>
    ${^stdout}
</command-display>

<tool:check_web.vbs>
    Dim web
    Set web = Nothing
    Set web = CreateObject("WinHttp.WinHttpRequest.5.1")

    numargs = wscript.arguments.count
    If (numargs < 1) Then
        wscript.Echo "Usage: check_web hostname"
        wscript.quit(4)
    End If

    URL = "http://" + wscript.arguments(0)

    on error resume next
    web.Open "GET", URL, False
    on error resume next
    web.Send
    If err.Number <> 0 Then
        returncode = 4
    Else
        If err.Number = 0 and web.Status = "200" Then
            returncode = 0
        Else
            returncode = 4
        End If
    End If

    If returncode <> 0 Then
```

```
wscript.Echo "Error connecting to " + URL + "."  
Else  
wscript.Echo ""  
End If  
wscript.quit(returncode)  
</tool:check_web.vbs>
```



# Command Line Probe Example

The following shell script is called from the command-line probe:

```
#!/bin/sh
# Expects an address passed in. Passes out the address and a
pretend result.
# Note that we use "\$" instead of just "$" because "$" has
special meaning
# in a shell script.
echo "\{ \$addr := \"$1\", \$result :=1.2345 } Note that
everything after the brace is used as the reason."
```

```
<!--
  Simple Command Line Example (com.dartware.cmd.simple)
  Copyright© Help/Systems, LLC. All rights reserved.
-->

<header>
  type = "cmd-line"
  package = "com.dartware"
  probe_name = "cmd.simple"
  human_name = "Simple Command Line Output Example"
  version = "1.0"
  address_type = "IP"
</header>

<description>
This probe shows how to use the specially-formatted output
from the
simple shell script listed above for display in the command-
display
section, rather than being set to the reason as is usual for
command-line probes.
</description>

<parameters>
</parameters>

<command-line>
  path = ""
  cmd = "simple.sh ${ADDRESS}"
</command-line>

<command-exit>
  down: ${EXIT_CODE} = 2
  alarm: ${EXIT_CODE} = 1
  okay: ${EXIT_CODE} = 0
```

```
</command-exit>

<command-display>
  \B5\Simple Probe Information\0P\
  Output from $addr is $result (${chartable: #.#### :
$result})
</command-display>
```

For more information about Nagios, visit the web site at <http://www.nagios.org>. Nagios® and the Nagios logo are registered trademarks of Ethan Galstad.

# Intermapper Python Plugins

Intermapper DataCenter ships with an embedded Python interpreter. You may use this interpreter to write command-line probe scripts and command-line notifiers. This Python interpreter is a good way to give maximum compatibility across systems. In Intermapper 6.3, the version of Python we ship is 2.6, with optimized system libraries.

An extensive introductory tutorial on Python is available at <http://docs.python.org/tut>

As shipped, this Python interpreter requires the use of optimized and stripped mode (-OO), so the interpreter must be invoked as:

**MacOSX/** /usr/local/imdc/core/python/bin/imdc -OO [script\_  
**Linux/** name]  
**Unix:**

**Windows:** c:\Program  
Files\InterMapper\dwf\core\python\imdc.exe -OO  
[script\_name]

## Notes:

- Use the \${PYTHON} macro as shown below; it automatically determines the platform and expands to the proper path to the interpreter with the -OO argument.
- Use the [The <tool> Section](#) (<tools:sample.py> in the example below) to incorporate the Python script directly into the probe file itself.

## Simple example

A simple sample probe that includes a Python script might look like this. The script automatically gets saved in the InterMapper Settings/Tools directory.

```
<!--
  Command Line Python Sample (com.dartware.python.sample.txt)
  Custom Probe for InterMapper (http://www.intermapper.com)
  Please feel free to use it as a base for further
  development.
  Original version 31 Mar 2004 by Christopher L. Sweeney,
  HelpSystems, LLC.
  Updated 13 Jun 2007 by Stephen P. Ryan, HelpSystems, LLC,
  for Python
  Updated 28 Dec 2007 to update text descriptions and
  include display_name header line -reb
  Updated 3 Jan 2010 to include ${PYTHON} macro -reb
-->

<header>
```

```
type="cmd-line"
package="com.dartware"
probe_name="python.sample"
human_name="Python Sample"
version="1.1"
address_type="IP"
display_name = "Miscellaneous/Test/Python Sample"
</header>

<description>
\GB\Python Sample Command-Line Probe\p\

A sample command line probe which executes a Python script.

The Python script generates and returns a random number which
sets
the device status to one of four values
Down/Alarm/Warning/OK.

</description>

<parameters>
</parameters>

<command-line>
path=""
cmd="${PYTHON} sample.py ${ADDRESS}"
arg=""
</command-line>

<command-exit>
down:${EXIT_CODE}=3
alarm:${EXIT_CODE}=2
warning:${EXIT_CODE}=1
okay:${EXIT_CODE}=0
</command-exit>

<command-display>
</command-display>

<tool:sample.py>
#!/usr/local/imdc/core/python/bin/imdc -OO
# Sample Python script uses InterMapper's Python interpreter

import sys

if (len(sys.argv) < 2):
    print "Usage: %s _address_" % sys.argv[0]
    sys.exit(0)
```

```
addr = sys.argv[1]

# Code to get status from device at address addr
import random
result = random.randrange(4)

print "Pretending we got result %d from device at address %s"
% (result, addr)
sys.exit(result)

</tool:sample.py>
```

# Nagios Plugins

Intermapper's command-line probes are similar to **Nagios® plugins** (<http://www.nagios.org>). You can see the [standard set of Nagios plugins](#). Many vendors and individuals have created their own Nagios plugins, many of which are available in the [development section](#). You will have to download the Nagios plugins and build/compile them yourself.

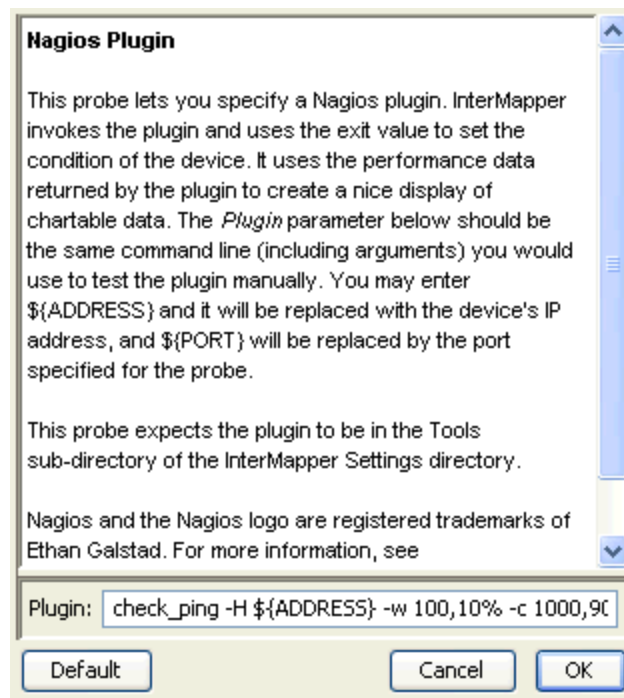
The Nagios Plugin probe lets you specify a Nagios plugin to run, along with any associated parameters. You can use the `${ADDRESS}` and `${PORT}` macros in the command line; Intermapper substitutes the device's IP address and the specified port. Intermapper will invoke the plugin and use the exit value to set the condition of the device to UP/Okay, UP/Alarm, UP/Critical, or DOWN.

Intermapper also interprets the information written by the plugin to stdout and puts it in the Intermapper status window, nicely displaying and making chartable the performance data returned by the probe, and displaying the reason/condition provided.

The Nagios Plugin probe expects the Nagios plugin to be in the Tools sub-directory of the Intermapper Settings directory. Nagios and the Nagios logo are registered trademarks of Ethan Galstad. For more information, see <http://www.nagios.org/>.

## To install and use a Nagios plugin:

1. Download the plugin. Make it executable by following the instructions from the creator.
2. Move the executable file (or a link/alias/shortcut to it) to the *Tools* sub-directory of the *Intermapper Settings* directory.
3. Add a device to the map and set its Probe Type to **Nagios Plugin**.
4. Enter in the plugin file's name and arguments in the Plugin field of the configuration window.
5. You can use the `${ADDRESS}` and `${PORT}` macros in the command line. Intermapper will substitute the device's IP address and the specified port.



## Creating Nagios Probes

If you wish to develop your own Nagios plugins, you should follow the developer guidelines for Nagios (found at <http://nagiosplug.sourceforge.net/developer->

). This will result in probes/plugins that work for both Intermapper and Nagios.

As described in the Nagios Guidelines, a Nagios plugin returns:

- **a POSIX return code** as described in [section 2.4](#) of the Guidelines. Intermapper uses this to determine the device's state.
  - 0 = OK;
  - 1 = Warning (yellow);
  - 2 = Critical (red);
  - 3 = Down.
- **A single output line on STDOUT** with the following format.

<description of the device status>|Perfdata

where:

- <description of the device status> is a short text string. This becomes the Intermapper Condition string, and is described in [section 2.1](#) of the Guidelines. The output string should have the format:

SERVICE STATUS: information text

- | is the "pipe" character to separate the description from the "Perfdata"
- Perfdata (Performance Data) is a series of name/value pairs. These are described in [section 2.6](#) of the Guidelines, but are generally a space-separated list with this form:

'label'=value[UOM];[warn];[crit];[min];[max]

## Example Return String

The Nagios check\_load string returns three values: the load average over 1, 5 and 15 minutes. When the plugin is invoked, it returns a response like this:

```
% ./check_load -w 15,10,5 -c 30,25,20
OK - load average: 0.95, 0.72,
0.64|load1=0.954;15.000;30.000;0;
load5=0.718;10.000;25.000;0; load15=0.635;5.000;20.000;0;
```

Intermapper parses the plugin's response line and uses the `${nagios_output}` macro to produce a status window as shown in the image below.

```
localhost.  
Device Status  
  Name: localhost.  
  DNS Name: localhost.  
  Address: 127.0.0.1  
  Status: UP  
  Probe: Nagios Plugin  
  Up Time: n/a  
  Availability: 100 % (of 2 days, 3 hours, 6 minutes)  
  Packet Loss: 0.0 % (of 0 total attempts)  
  Short-term Packet Loss: 0.0 % (of 0 last attempts) [Reset]  
  Recent Loss: None  
  Response time: 40 msec  
  
  Retention Policy: 24 Hours (-3), Exportable  
  
NAGIOS Probe Performance Data: check_load -w 15,10,5 -c 30,25,20  
load1 : 0.954  
load5 : 0.718  
load15: 0.635  
  
Recent Outages: [Reset]  
08/09 22:18:46: DOWN for 0 seconds  
07/15 03:03:25: DOWN for 20 seconds  
07/10 07:46:07: DOWN for 29 seconds  
Last updated Aug 27, 01:59:02; interval: 30 seconds  
Reason: OK - load average: 0.95, 0.72, 0.63
```

For more information about Nagios, visit the web site at <http://www.nagios.org>. Nagios® and the Nagios logo are registered trademarks of Ethan Galstad.

## Changes for Intermapper 5.0

For those familiar with the older Nagios Template probe, the new Nagios Plugin probe contains the following changes in behavior:

- The Nagios Template probe mapped plugin exit code 2 as down. The Nagios Plugin probe maps plugin exit code 2 as critical, and plugin exit code of 3 as down.
- The Nagios Template probe took anything written to stdout as the "condition" or "reason" for the status. The Nagios Plugin probe detects the presence of performance data ('PERFDATA') ([section 2.6](#) of the Guidelines) in the output, and makes a formatted and chartable display of the data.
- We have not changed the canonical name of the Nagios probe; any device which used the old Nagios Template probe will now automatically use the Nagios Plugin probe. An Intermapper probe will automatically handle a Nagios plugin if it has the following:
  - "flags" = "NAGIOS3" in the <header> section of the probe. See [Probe File Header](#).
  - \${nagios\_output} in the <command-display> section of the probe. See [Built-in Variables](#)



# Nagios Plugin Example

```
<!--
Command Line Nagios Plug-in Example
(com.dartware.nagiosx.template)
Copyright© HelpSystems, LLC. All rights reserved.
-->

<header>
  type           = "cmd-line"
  package        = "com.dartware"
  probe_name     = "nagios.template"
  human_name     = "Nagios Plugin"
  version        = "1.6"
  address_type   = "IP"
  display_name   = "Miscellaneous/Nagios/Nagios Plugin"
  flags          = "NAGIOS3"
</header>

<description>
\GB\Nagios Plugin\p\
This probe lets you specify a Nagios plugin. Intermapper
invokes the plugin and uses the exit value to set the
condition of the device. It uses performance data returned by
the plugin to create a nice display of chartable data. The
\i\Plugin\p\ parameter below should be the same command line
(including arguments) used to test the plugin manually.
\${ADDRESS} is replaced with the device's IP address, and
\${PORT} is replaced by the port specified for the probe.

This probe looks in the Tools sub-directory of the
Intermapper
Settings directory for the plugin.

Nagios and the Nagios logo are registered trademarks of Ethan
Galstad. For more information, see
\U2\http://www.nagios.org\P0\
</description>

<parameters>
  Plugin = "check_ping -H ${ADDRESS} -w 100,10% -c 1000,90%"
</parameters>

<command-line>
-- Empty path forces the Intermapper Settings:Tools directory
  path = ""
  cmd  = ${Plugin}
</command-line>
```

```
<command-exit>
-- These are the exit codes used by Nagios plugins
  down: ${EXIT_CODE}=3
  critical: ${EXIT_CODE}=2
  alarm:    ${EXIT_CODE}=1
  okay:     ${EXIT_CODE}=0
</command-exit>

<command-display>
\B5\NAGIOS Probe Performance Data: ${Plugin}\P0\
${nagios_output}
</command-display>
```

# NOAA Weather Probe Example

It is now easier than ever to build command-line probes. Here is a fun example that retrieves temperature data from the [US NOAA weather feed](#) in a particular city.

How does this probe work?

1. Right-click a device and choose **Select Probe...**
2. Select the **Weather Service-Temp** probe from the Miscellaneous/Test category.
3. Enter the city code for the closest weather station (KLEB is at Lebanon Municipal Airport, about a half mile to the south of us.) The Status window shows the name of the weather station, with a chartable value for the temperature reading.

Under the covers, Intermapper launches a Python program to contact the weather service, retrieve the meteorological conditions for the indicated city, and parses out the XML response to retrieve the temperature. (There's lots more information in the Weather Service feed - the program could easily be extended to display more information.) Here are some of the features of this probe:

- The `${PYTHON}` macro gives the path to the built-in python interpreter of Intermapper DataCenter no matter what platform you're using. For example, the probe can now use

```
cmd = "${PYTHON} program.py"
```

and Intermapper substitutes the proper path to invoke Python, whether on Windows, OSX, or Linux/Unix.

**Note:** To use this macro, the Intermapper DataCenter (IMDC) must be installed. IMDC will be installed automatically with Intermapper 5.2 on Windows and OSX; Linux and Unix systems require a separate install for IMDC.

- You can include the script directly in the text of the probe file. This makes it much easier to write scripts and keep the probe file in sync. To do this, use the `<tool:program-name>` section in your probe file. The example below contains a program named `noaa-weather.py`. When Intermapper loads the probe, it parses out this section and saves it in a folder within the Tools directory of Intermapper Settings. Programs in the `<tools>` section may also save private files in that directory.
- The example probe file uses a couple interesting Python libraries. First is `urllib2` that makes it easy to make queries from web services. It's a few straightforward calls to build a url, issue it, and retrieve the results.
- The probe also uses the `xml.dom.minidom` library to parse out XML data returned from the NOAA web service. This library is particularly well-explained in Chapter 9 of *Dive into Python*.

# The NOAA Temperature Probe

To use this probe, copy the text below, paste it to a text editor, save it to a text file, then use File->Import->Probe... in Intermapper.

```
<!--
Weather Service Temperature - Retrieve the temperature from
the NOAA weather XML (com.dartware.tool.noaa.txt)
Copyright© HelpSystems, LLC.
Please feel free to use this as a base for further
development.
-->

<header>
  type      = "cmd-line"
  package   = "com.dartware"
  probe_name = "tool.noaa"
  human_name = "Weather Service-Temperature"
  version    = "1.2"
  address_type = "IP"
  display_name = "Miscellaneous/Test/Weather Service-Temp"
</header>

<description>
\GB\Retrieve the current temperature\p\

This probe retrieves the current temperature from the NOAA
weather feed. To see the proper city code, visit:

\u4=http://www.weather.gov/xml/current_
obs/\http://www.weather.gov/xml/current_obs/\p0\
</description>

<parameters>
  "Weather Station"    = "KLEB"
</parameters>

<command-line>
  path=""
  cmd="${PYTHON} noaa-weather.py"
  arg="${Weather Station}"
</command-line>

<command-exit>
```

```
-- These are the exit codes used by Nagios plugins
    down:  ${EXIT_CODE}=4
    critical:  ${EXIT_CODE}=3
    alarm:  ${EXIT_CODE}=2
    warn:  ${EXIT_CODE}=1
    okay:  ${EXIT_CODE}=0
</command-exit>

<command-display>
\b5\ Temperature for $loc\p0\
    Temperature: $temp \3g\degrees F\p0\
</command-display>

<tool:noaa-weather.py>

# noaa-weather.py
# Scan the XML results from NOAA's XML feeds
# e.g., http://www.weather.gov/xml/current_obs/KLEB.xml
# for relevant weather-related information.
# 25 Mar 2009 -reb
import os
import re
import sys
import getopt
import urllib
import urllib2
import httplib
from xml.dom import minidom

# httplib.HTTPConnection.debuglevel = 1 # force debugging....

# options are: station

try:
    opts, args = getopt.getopt(sys.argv[1:], "")
except getopt.GetoptError, err:
    searchString = "getopt error %d" % (err)

station = args[0]
userAgent = "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_5; en-us) AppleWebKit/525.18 (KHTML, like Gecko) Version/3.1.2 Safari/525.20.1"
noaaString = "http://www.weather.gov/xml/current_obs/%s.xml"
noaaString = noaaString % (urllib.quote_plus(station))

# print noaaString;
retcode = 4;
```

```
try:
    request = urllib2.Request(noaaString)
    opener = urllib2.build_opener()
    request.add_header('User-Agent', userAgent)
    usock= opener.open(request)
# print buf
except IOError, e:
    if hasattr(e, 'reason'):
        resp = 'We failed to reach a server. '
        reason = 'Reason: ' + 'Wrong host name?' # e.reason[1]
    elif hasattr(e, 'code'):
        resp = 'The server couldn\'t fulfill the request. '
        reason = 'Error code: ' + str(e.code)
    print "\{ $temp := '%s', $loc := 'Unknown' } %s" % (0, resp
+ reason)
    sys.exit(retcode) # make it look down

retcode = 0 # looks like it'll succeed
xmldoc = minidom.parse(usock)
tempList = xmldoc.getElementsByTagName('temp_f')
tempElem = tempList[0]
tempval = tempElem.firstChild.data
loclist = xmldoc.getElementsByTagName('location')
locval = loclist[0].firstChild.data
print "\{ $temp := '%s', $loc := '%s' }%s" % (tempval,
locval, tempval + ' degrees at ' + locval)
sys.exit(retcode)
</tool:noaa-weather.py>
```

### See also

[\\${PYTHON} macro](#) - the full path the to the Python interpreter.

[The <tool> Section](#) - Include a script directly into the probe file.

### Python Documentation:

urllib2 - <http://docs.python.org/library/urllib2.html>

xml.dom.minidom - <http://docs.python.org/library/xml.dom.minidom.html>

**Dive into Python:** A very readable chapter on XML processing in Python

[http://diveintopython.org/xml\\_processing/](http://diveintopython.org/xml_processing/)

# PowerShell\_Probe

A PowerShell probe is essentially a Command Line probe with a PowerShell Script attached to it.

The only difference is that there are two sets of script arguments:

1. The arguments passed to the Windows command line
2. The arguments passed to the script itself

The things you can do with a PowerShell probe are virtually limitless.

See the [PowerShell Probe Example](#) for more information.

# PowerShell Probe Examples

PowerShell probes are command-line probes. They launch PowerShell, then invoke a command.

The two examples below demonstrate two different Intermapper macros:

- **`\${PSREMOTE}** - for less experienced PowerShell users, this macro handles the connection to the remote machine. It creates a credential object and sets up authentication, then executes the specified command on the remote machine.
- **`\${PS}** - for experienced PowerShell users, this macro simply launches PowerShell on the local machine with the specified arguments, and leaves all PowerShell commands up to the developer.

These macros are used in the [<command-line>](#) section of the probe.

Any script must be located in the `Intermapper Settings\Tools` folder. If you include the script in the `<tools>` section, it is installed in the Tools folder when you load the probe.

## Example 1: Installed Software Probe

This probe lists installed applications, updates, or both on the target device. It launches PowerShell with the arguments supplied in `arg`, uses ``${PSREMOTE}` to connect to and authenticate on the remote device, then executes the command specified `input`.

```
<!--
This probe lists installed Applications, Updates, or Both
using PowerShell. Requires PowerShell 2.0 or later and
requires that PS remoting be enabled.

File Name:
com.helpsystems.powershell.remote.installedSoftware.txt
(c) 2015 HelpSystems, Inc.
-->

<header>
    type = "cmd-line"
    package = "com.helpsystems"
    probe_name = "ps.remote.InstalledSoftware"
    human_name = "Installed Software"
    version = "1.0"
    address_type = "IP"
    display_name = "PowerShell/Remote/Installed Software"
    visible_in = "Windows"
    flags = "NTCREDENTIALS"
</header>
```



```

<description>
\GB\List Installed Software\p\

This probe uses PowerShell to provide a listing of installed
software, installed updates, or both. This probe requires
that \b\PowerShell 2.0\p\ or later be installed, and
PowerShell remoting must be enabled and configured to use
this probe. This probe uses the registry, not WMI objects

Intermapper invokes the included ApplicationList.ps1
companion script in Intermapper Settings/Tools.

</description>

<parameters>
    "Type[Software,Update,All]"="Software"
    User=""
    "Password*" = ""
    "Authentication
[Default,Basic,Negotiate,NegotiateWithImplicitCredential,Cre
dssp,Digest,Kerberos]"="Default"
    "Timeout (sec)"="10"
</parameters>

<command-exit>
    down:${EXIT_CODE}=4
    critical:${EXIT_CODE}=3
    alarm:${EXIT_CODE}=2
    warning:${EXIT_CODE}=1
    okay:${EXIT_CODE}=0
</command-exit>

<command-line>
    path=""
    cmd="${PSREMOTE}"
    arg="-ExecutionPolicy RemoteSigned -NoProfile"
    input = "Invoke-Command -FilePath .\ApplicationList.ps1 -
ArgumentList '${Type[Software,Update,All]}'"
    timeout = ${Timeout (sec)}
</command-line>

<command-display>
    ${Type[Software,Update,All]} installed on ${address}
    ${^stdout}
</command-display>

<tool:ApplicationList.ps1>
param([string] $filter)

```

```
$exitCode = 0
$reason = ''

if ($filter -eq 'All')
{
    $software = Get-ItemProperty
HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\
Uninstall\* | Select-Object DisplayName, DisplayVersion,
InstallDate, Publisher | Sort-Object DisplayName
}

elseif ($filter -eq 'Update') #show only windows updates
{
    $software = Get-ItemProperty
HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\
Uninstall\* | Select-Object DisplayName, DisplayVersion,
InstallDate, Publisher | where {$_.DisplayName -match
$filter} | Sort-Object DisplayName
}

elseif ($filter -eq 'Software')
{
    $software = Get-ItemProperty
HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\
Uninstall\* | Select-Object DisplayName, DisplayVersion,
InstallDate, Publisher | where {$_.DisplayName -notmatch
'update'} | Sort-Object DisplayName
}

#get rid of blanks in the input
$cleanedUpList = New-Object System.Collections.ArrayList

foreach($app in $software)
{
    if ($app.DisplayName )
    {
        $cleanedUpList.Add($app) | Out-Null #ArrayList.Add
returns the index of the item added, we don't want this goint
to standard out, confusing Intermapper.
    }
}

#set up the object for return

# Intermapper can't take an array or ArrayList of objects
yet, so convert to a string.
# Also, Powershell will truncate to a default size, unless
the table is formatted using -AutoSize and get around column
```

```

dropping by setting the width of the resulting string.
$stdoutString = $cleanedUpList |format-table -AutoSize | out-
string -width 4096

$result = New-Object PSCustomObject -Property @{
    'stdout'=$stdoutString;
    'ExitCode'=$exitCode;
    'reason'=$reason;
}
write-output $result

</tool:ApplicationList.ps1>

```

## Example 2: Windows Disk Space Probe

This probe checks the amount of disk space on the target device. It uses `${PS}` to launch PowerShell with the arguments supplied in `arg`, then executes the command specified in `input`.

Like the first example, this one connects to a remote device, but does not use `${PSREMOTE}` to handle the connection. This example also passes thresholds to the script so that it can return the correct exit code.

```

<!--

Windows Disk Space Probe
This probe uses a PowerShell script to look up the amount of
disk space on the
target device.
(c) 2015 HelpSystems, Inc.
-->

<header>
    type = "cmd-line"
    package = "com.helpsystems"
    probe_name = "ps.wmi.diskspace"
    human_name = "Non-Remoting (WMI) Disk Space Monitor"
    version = "1.0"
    address_type = "IP"
    display_name = "PowerShell/Disk Space"
    visible_in = "Windows"
    flags = "NTCREDENTIALS"
</header>

<description>

```

```
\GB\Windows Disk Space Monitor\p\
```

This probe uses Powershell to retrieve the disk space available on a drive on the target host. Specifically, it queries the Size and FreeSpace properties of the Win32\_LogicalDisk class, computes percentage free space, and compares it against the Warning and Critical parameters you set. The target host must be running PowerShell with Remoting enabled.

The Drive parameter may be set to "All" to enumerate all Local hard drives on the host. It may also be set to a list of comma-separated drive names (including the colon), which will be listed whether or not they are local hard drives. Zero-sized drives (i.e. an empty cd-rom) will not be listed. The first drive failing the warning or critical criteria will be the one cited in the reason.

The User parameter may be a local user on the target host, or may take the form of "domain\\user" for a domain login. Leave it blank if authentication is not required, such as when the target is the localhost.

Intermapper invokes the WindowsFreeDiskSpace.ps1 companion script which was placed in the Tools folder of the Intermapper Settings folder when this probe was loaded. It uses the exit value to set the condition of the device and the performance data returned by the script to create a nice display of chartable data.

</description>

<parameters>

```
Drive="C:"  
"Warning (%)"="10"  
"Alarm (%)"="5"  
"Critical (%)"="3"  
"Down (%)"="1"  
User=""
```

```

        "Password*" = ""
        "Timeout (sec)"="10"
        "Powershell Version
[notSpecified,2.0,3.0,4.0,5.0]"="notSpecified"
</parameters>

<command-exit>
    down:${EXIT_CODE}=4
    critical:${EXIT_CODE}=3
    alarm:${EXIT_CODE}=2
    warning:${EXIT_CODE}=1
    okay:${EXIT_CODE}=0
</command-exit>

<command-line>
    path=""
    cmd="${PS}"
    arg="-ExecutionPolicy RemoteSigned -NoProfile"
    input = "$c = New-Object
System.Management.Automation.PSCredential -ArgumentList
'${User}', (ConvertTo-SecureString -String '${Password*}' -
AsPlainText -Force) ; Invoke-Command -ScriptBlock { &
'..\WindowsFreeDiskSpace.ps1' -compName '${address}' -cred $c
-drives '${Drive}' -downThr ${Down (%)}} -critThr ${Critical
(%)) -alrmThr ${Alarm (%)}} -warnThr ${Warning (%)}} $"
    timeout = ${Timeout (sec)}
</command-line>

<command-display>

Disk Space Available
    ${^stdout}
</command-display>

<tool:WindowsFreeDiskSpace.ps1>
param([string] $compName,
[System.Management.Automation.PSCredential] $cred, [string]
$drives, [int] $downThr, [int] $critThr, [int] $alrmThr,
[int] $warnThr)

Function Update-ExitCode
{
    param([int] $current_status, [int] $new_status)

    if ($new_status -gt $current_status) { return $new_status
}
    else { return $current_status }
}

```

```
Function Update-Reason
{
    param($disk_name,$disk_threshold)
    return "Disk $disk_name is below $disk_threshold % free."
}

$STATUS = New-Object -TypeName PSObject -Prop(@
{'down'=4;'critical'=3;'alarm'=2;'warning'=1;'ok'=0})

#reason and exit code
[int] $exit_code = $STATUS.ok
[string] $reason = "All disks within acceptable limits"
[string] $debugInfo = ''

$disks = New-Object -TypeName System.Collections.ArrayList

if ($drives -eq "All")
{
    $disks = (Get-WmiObject Win32_LogicalDisk -ComputerName
$compName -Credential $cred -Filter "DriveType='3'" | Select-
Object Size,FreeSpace,DeviceID)
}

else
{
    $diskList = (Get-WmiObject Win32_LogicalDisk -
ComputerName $compName -Credential $cred | Select-Object
Size, Freespace, DeviceId)

    $driveArray = $drives.replace(' ', '').split(',')
    foreach($drive in $driveArray)
    {
        #$debugInfo += "$drive`r`n"

        $found = $false

        foreach($disk in $diskList)
        {
            #$debugInfo += $disk.GetType().FullName

            if ($disk.DeviceID -eq $drive)
            {
                $found = $true
                if ($disk.Size -ne $null)
                {
                    #$debugInfo += " -- adding " +
$disk.DeviceID + " Size: " + $disk.Size + " FreeSpace: " +
$disk.FreeSpace + "`r`n"
                }
            }
        }
    }
}
```

```

        $disks.Add($disk)
    }
    else
    {
        $debugInfo += $disk.DeviceID + " --- No
information ---`r`n"
    }
}

if ($found -ne $true)
{
    $debugInfo += $drive + " --- Not found ---`r`n"
}
}

if ($disks.count -eq 0)
{
    throw "Disks could not be found or parameter error. Check
your probe settings.`r`n" + $debugInfo
}

foreach ($disk in $disks)
{
    #calculate percentage of the disk that is free
    $disk | Add-Member -type NoteProperty -name "PercentFree"
-value ([Math]::round($disk.FreeSpace / $disk.Size * 100))
    $disk.Size = "{0:N1}" -f [Math]::round
(($disk.Size/1GB))
    $disk.FreeSpace = "{0:N1}" -f [Math]::round
(($disk.FreeSpace/1GB))

    # calculate alerts
    if ($disk.PercentFree -le $downThr)
    {
        # $disk | Add-Member -type NoteProperty -name "exit_
code" -value $STATUS.down
        $old_code = $exit_code
        $exit_code = $STATUS.down
        if ($old_code -ne $exit_code)
        {
            $reason = Update-Reason $disk.DeviceID $downThr
        }
    }

    elseif ($disk.PercentFree -le $critThr -and
$disk.PercentFree -gt $downThr )

```

```

{
    # $disk | Add-Member -type NoteProperty -name "exit_
code" -value $STATUS.critical
    $old_code = $exit_code
    $exit_code = Update-ExitCode $exit_code
$STATUS.critical
    if ($old_code -ne $exit_code)
    {
        $reason = Update-Reason $disk.DeviceID $critThr
    }
}

elseif ($disk.PercentFree -le $alarmThr -and
$disk.PercentFree -gt $critThr )
{
    # $disk | Add-Member -type NoteProperty -name "exit_
code" -value $STATUS.alarm
    $old_code = $exit_code
    $exit_code = Update-ExitCode $exit_code $STATUS.alarm
    if ($old_code -ne $exit_code)
    {
        $reason = Update-Reason $disk.DeviceID $alarmThr
    }
}

elseif ($disk.PercentFree -le $warnThr -and
$disk.PercentFree -gt $alarmThr )
{
    # $disk | Add-Member -type NoteProperty -name "exit_
code" -value $STATUS.warning
    $old_code = $exit_code
    $exit_code = Update-ExitCode $exit_code
$STATUS.warning
    if ($old_code -ne $exit_code)
    {
        $reason = Update-Reason $disk.DeviceID $warnThr
    }
}

else
{
    # $disk | Add-Member -type NoteProperty -name "exit_
code" -value $STATUS.ok
    $exit_code = Update-ExitCode $exit_code $STATUS.ok
}
}

#format the output for the probe to display in the status

```



```
window
$stdoutString = ($disks | Format-Table
DeviceID,Size,FreeSpace,PercentFree | out-string)
$stdoutString += $debugInfo

#create the return object that the probe will use for display

$result = New-Object PSCustomObject -Property @{
    'stdout'=$stdoutString;
    'ExitCode'=$exit_code;
    'reason'=$reason;
}
write-output $result

</tool:WindowsFreeDiskSpace.ps1>
```

# Troubleshooting PowerShell Probes

When you run a PowerShell probe, Intermapper launches PowerShell.exe and has it execute a command or script. It passes two sets of input:

- The parameters used to launch PowerShell.
- The command executed once PowerShell is launched.

Intermapper combines these inputs into a single command, which may reference a separate PowerShell script. All scripts (or links to them) must reside in the Intermapper Settings\Tools folder.

Each time a PowerShell probe is chosen, or when its parameters change, two things happen.

1. A connectivity test is run.
2. If the test is successful, the probe runs at the next polling interval.

For the connectivity test, and for each time a PowerShell probe runs, two entries are created in the Debug log:

- One entry shows the input string sent to `stdin`.
- A second entry shows the variables returned by the probe, enclosed in "{...}", followed by the string assigned to `stdout`.

Example Debug Log entries are shown below.

For each entry, the first two sets of numbers are:

- The time
- The IP address of the target device.

## Connectivity Test Command

```
12:56:14 10.65.49.31 : Remoting Disk Space Monitor:
XCmdLine::SendProbe: stdin: 1090 $global:t = 0 ; try { ;
$errorActionPreference = 'Stop' ; $global:t = 1 ; Write-
Output "PSRTest: $global:t" ; $vMaj =
$PSVersionTable.PSVersion.Major ; Write-Output $vMaj ;
$global:t = 2 ; Write-Output "PSRTest: $global:t" ; Test-
WSMan 10.65.49.31 ; $global:t = 3 ; Write-Output
"PSRTest: $global:t" ; $cred = New-Object
System.Management.Automation.PSCredential -ArgumentList
'\Fred Flintstone', (ConvertTo-SecureString -String
'*****' -AsPlainText -Force) ; Connect-WSMan
10.65.49.31 -Authentication Default -Credential $cred ;
$maxConnections = Get-ChildItem -Path
WSMan:/10.65.49.31/Service/MaxConnections ; Disconnect-
```

```
WSMan 10.65.49.31 ; Write-Output $maxConnections ;
$global:t = 4 ; Write-Output "PSRTest: $global:t" ; $sess
= New-PSSession 10.65.49.31 -Authentication Default -
Credential $cred ; $result = New-Object PSCustomObject -
Property @{ 'State'=$sess.State;
'Availability'=$sess.Availability } ; Remove-PSSession -Id
$sess.Id ; Write-Output $result ; } catch { ; throw
"Exception in PSRTest: $global:t $_.Exception.Message" ; }
```

## Connectivity Test Response

```
12:56:19 10.65.49.31 : Remoting Disk Space Monitor:
XCmdLine::PollProbeForPS -- Reason: \{ reason='PowerShell
Remoting Test succeeded; Your probe will run next probe
cycle.'} *** PowerShell Tests *** Running with PowerShell
version 3.0.
Test-WSMan succeeded: received expected response.
Connect-WSMan succeeded: MaxConnections = 300.
New-PSSession succeeded: State = Opened, Availability =
Available.

The PowerShell Remoting Test succeeded. Your probe will run
next probe cycle.
```

## Sending a Command

```
12:56:44 10.65.49.31 : Remoting Disk Space Monitor:
XCmdLine::SendProbe: stdin: 409 $cred = New-Object
System.Management.Automation.PSCredential -ArgumentList
'\Fred Flintstone', (ConvertTo-SecureString -String
'*****' -AsPlainText -Force) ; $sess = New-PSSession
10.65.49.31 -Authentication Default -Credential $cred ; try
{ Invoke-Command -Session $sess -FilePath
.\WindowsFreeDiskSpace.ps1 -ArgumentList localhost, 'C:', D:,
L:', 1, 3, 5, 10 } finally { Remove-PSSession -Id $sess.Id }
```

## Command Response

```
12:56:49 10.65.49.31 : Remoting Disk Space Monitor:
XCmdLine::PollProbeForPS -- Reason: \{
PSComputerName='10.65.49.31',RunspaceId='a8f1f138-7781-
4e77-a185-
18aa6db978c9',PSShowComputerName='true',reason='Disk C: is
below 5 % free.'} DeviceId Size Freespace PercentFree
-----
C:          918.0 43.0          5
D:          13.0  2.0          12
L:          932.0 917.0         98
```

# Troubleshooting Probes

There are a number of different ways to troubleshoot your custom probes.

The most basic troubleshooting is done through the error messages that appear in the device's Status window. Use the [comprehensive list of Error Messages](#) to help you track down errors.

For SNMP probes, you can use [SNMPWalk](#) to view the MIB variables returned from an SNMP device. Use [SNMPWalk with the -O option](#) to redirect the output from the Debug log to an SQLite database.

You can also gain a lot of information by [measuring response times](#) of a device as it is being tested. A number of different timers are available for viewing and charting.

# Errors with Custom Probes

When working with custom probes, you may see results that you don't expect. Here are some common problems.

## "Undefined variable" in Debug log

When processing a probe, Intermapper will not evaluate an expression if it detects a variable that is not defined. A variable will be undefined if it is not in the symbol table. This could happen because:

- there is a typo in the variable name
- the value for the variable was not returned in a SNMP response
- the value was not set earlier in the probe processing

In this case, Intermapper will emit the following message in the Debug log file:

```
Calculation error in rule (probe: com.dartware.example,
expression:
  "($oid 0)": Undefined variable: '$oid'
```

To guard against these error messages (where it may be a legitimate case that a particular variable is undefined) you may use the "defined()" function in the expression:

```
warning: defined("oid") && ($oid > 0) "Warning condition
string"
```

## A device shows a "Reason: No SNMP Response." at the bottom of the status window.

There are several reasons that Intermapper might not be able to retrieve SNMP information from a device. The two most common are

- The device doesn't speak SNMP
- You haven't entered the proper SNMP read-only community string.

*About SNMP*, located in the Troubleshooting section of the [User Guide](#), lists many other reasons.

## When I build a custom probe, the status window shows "[N/A]" for certain values.

This probably means that there is an error with the OID for one of the device variables.

Open the Debug window, and look for entries in this format:

```
12:57:00 router.example.net.: SNMP error status [[query =
28]] noSuchName (2), index = 3
1) 1.3.6.1.2.1.1.3: NULL
2) 1.3.6.1.2.1.1.1: NULL
3) 1.3.6.1.7.1.1.4: NULL
4) 1.3.6.1.2.1.1.6: NULL
```

Note that the first line above shows a "noSuchName" error for index 3. Look at the subsequent lines to find item 3, and check that OID very carefully. In this example, the proper OID should have a "2" in place of the "7" that's there.

## When I build a custom probe, the status window shows "[noSuchName]" for certain values.

This probably means that there is an error with the OID for one of the device variables.

Open the Debug window, and look for entries in this format.

```
13:17:59 OID Error: GetNextRequest from 192.168.1.1 expected
1.3.6.1.2.1.2.2.1.2.10; got 1.3.6.1.2.1.2.2.1.3.1
```

In this case, the desired value is from a non-existent table row. (The OID 1.3.6.1.2.1.2.2.1.2 is the ifDescr for an interface on a device. The index (.10) indicates which row to retrieve. But when Intermapper requested that row, it learned it was not present.) Consequently, Intermapper displays the "noSuchName" value.

# Debugging with the SNMPWalk Command

Intermapper provides a simple SNMPWalk command, available from the Monitor menu, that allows you to perform an SNMPWalk on a specified OID. In some cases this may not be sufficient. You can also execute SNMPWalk as a server command, and include specific arguments as described below.

The Intermapper server implements a simple snmpwalk facility in its debug mode.

```
snmpwalk -v [1|2c|3] -c community -o filename [-e] [-n num-OIDs] -p 161 -r 3 -t 10 IP-address startOID
```

where:

- `-v [1|2c|3]` is the version of SNMP to use: SNMPv1, SNMPv2c, or SNMPv3
- `-c community` indicates the SNMP read-only community string (see note for SNMPv3)
- `-e` if present, means to proceed to the end of the MIB
- `-n num-OIDs` if present, indicates the number of OIDs to display (`-e` and `-n` are mutually exclusive)
- `-o filename` is the name of a SQLite-format file saved in the Intermapper Settings/Temporary directory. For more information see [Using the SNMPWALK -O Option](#).
- `-p` destination port (default is 161)
- `-r` number of retries that Intermapper will attempt if a response doesn't return (default is 3)
- `-t` timeout in seconds that Intermapper waits for a response (default is 10 seconds)
- `IP-address` is the IP address of the device to query
- `startOID` if present as the final argument, indicates the first OID to request

The command will start an SNMP walk on device with the specified IP-Address, starting from the given startOID. The walk will end when the specified number of OIDs has been received. The walk will also end if the OID received from the device does not have the specified start OID as its prefix unless `-e` is specified. If `-e` is specified, the walk will continue until the end of the MIB or the specified maximum OIDs have been received.

*Note:* For SNMPv3, *community* should be in the following format:

```
username:[md5|sha|none]:authpassword:[des|none]:privpassword
```

## Examples

**Example:** SNMP walk of the ifTable of a device with IP address 192.168.1.1 using SNMPv2c with community string public:

```
snmpwalk -v 2c -c public 192.168.1.1 1.3.6.1.2.1.2.2
```

**Example:** SNMP walk of the ifXTable of a device with IP address 10.10.2.20 using SNMPv3 with user name 'user', authentication protocol MD5, authentication password 'auth', privacy protocol DES and privacy password 'priv':

```
snmpwalk -v 3 -c user:md5:auth:des:priv 10.10.2.20  
1.3.6.1.2.1.31.1.1
```

**Example:** SNMP walk of the ifTable of a device with IP address 192.168.1.2 using SNMPv3 with user name 'test', authentication protocol MD5, authentication password 'pass', and no privacy protocol:

```
snmpwalk -v 3 -c test:md5:pass:none: 192.168.1.2  
1.3.6.1.2.1.2.2
```

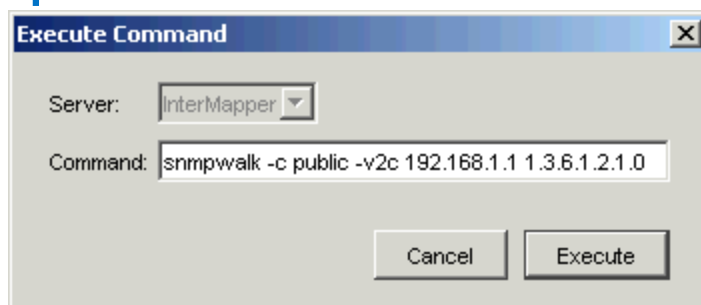
**Example:** Walk starting from the ifTable until the end of the device is reached or until 10,000 OIDs have been received:

```
snmpwalk -v 1 -c public -e -n10000 192.168.1.1  
1.3.6.1.2.1.2.2
```

## Invoking the snmpwalk command

You execute the *snmpwalk* command as a "Server command..." (available from the Help menu's Diagnostics menu) To use this command:

1. Select **Help > Diagnostics > Server Command...** The Server command window appears as shown above.
2. Enter the snmpwalk command, and click Send.
3. The output of the SNMPwalk is written to the **Debug** file, which is at this path: InterMapper Settings : InterMapper Logs : Debug\yyyyymmddhhmm.txt



Note: You can use snmpwalk's -o option to direct the output of snmpwalk to an



SQLite database. For more information, see [Using the SNMPWALK -o Option](#).

4. The output of the SNMPwalk will also appear in the Debug window, as shown below:

```
SNMPWalk 192.168.1.1: prefix 1.3 (maximum number of OIDs:
2000)
-- 9/16/2005 13:04:56
SNMPWalk on 192.168.1.1 started
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.1.1.0 = OctetString:
ExampleOS
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.1.2.0 = OID:
1.3.6.1.4.1.9.1
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.1.3.0 = TimeTicks: 11058776
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.1.4.0 = OctetString:
support@example.com
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.1.5.0 = OctetString:
Example.com Router
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.1.6.0 = OctetString:
http://www.example.com
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.1.7.0 = Integer: 72
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.1.8.0 = TimeTicks: 413
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.1.9.1.2.1 = OID:
1.3.6.1.2.1.1.9.1
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.1.9.1.3.1 = OctetString:
See RFC2580
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.1.9.1.4.1 = TimeTicks: 413
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.2.1.0 = Integer: 2
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.2.2.1.1.1 = Integer: 1
SNMPWalk 192.168.1.1: 1.3.6.1.2.1.2.2.1.1.2 = Integer: 2
...
SNMPWalk 192.168.1.1: Finished (end of MIB reached) --
9/16/2005 13:09:48
```

## snmpwalk stopall command

To stop all SNMPwalks for a particular server, you can enter this command in the Server command... window.

```
snmpwalk stopall
```

## Error Conditions

Intermapper detects the following error conditions:

- When Intermapper walks to the end of the MIB, it displays a "Finished (end of MIB reached)" message
- When Intermapper fails to receive a response after the specified number of retries, it displays a "Finished (No response received)" message
- The snmpwalk expects that the OIDs received are increasing. When Intermapper receives an OID that is out of order, it would terminate the walk with an error message that indicates that a loop is detected in the walk.

## Help from the telnet command line

There is also documentation in the Intermapper's telnet help. Typing 'help snmpwalk' in the telnet window will display a summary of the command.

## Using the SNMPWALK -o Option

Instead of writing the voluminous results of the SNMPWalk to the Debug log, Intermapper can write the results to an SQLite file. To use this feature, you use the **-o** option.

When you use the **-o** option, snmpwalk stores its output into an SQLite database. To use this feature, you specify the name of the database file following the **-o** option.

### Example:

To create an SQLite3 database named "foo" and store the snmpwalk results there, use the following "server command":

```
snmpwalk -v1 -c public -o foo switch 1.3
```

This writes the output to an SQLite database file named "foo", located in Intermapper Settings/Temporary directory. The database file named "foo" may contain multiple snmpwalk's. The file is created if it doesn't exist.

When the **-o** option is used, only these four lines are written to the Debug log:

```
SNMPWalk command received: 'snmpwalk -v1 -c public -o foo -n
10 switch 1.3'
SNMPWalk 192.168.1.36 3: prefix 1.3 (version SNMPv1 ...
SNMPWalk on switch started
SNMPWalk 192.168.1.36 3: Finished (10 OIDs ...
```

## The SNMPWALK Schema

Here is the schema used for the snmpwalk database:

```
CREATE TABLE walks (
  id          INTEGER PRIMARY KEY,
  address     TEXT,
  port       INTEGER,
  startOid    TEXT,
  snmpVersion INTEGER,
  pktTimeout  INTEGER,
  pktRetries  INTEGER,
  maxOids     INTEGER,
  toEnd       INTEGER,
  timeStarted INTEGER,
  timeFinished INTEGER,
  oidCount    INTEGER,
  stopReason  TEXT
);

CREATE TABLE results (
  walk_id    INTEGER,
  name       TEXT,
  oid        TEXT,
  type       INTEGER,
  value      BLOB
);
```

### Table: walks

The walks table stores one row for each snmpwalk command. Each walk will receive a unique id that is used to identify it (id). The other columns are:

address

the address of the snmpwalk target device

port

the udp port number of the snmpwalk target device

startOid

the starting OID specified in the snmpwalk command

snmpVersion

|   |
|---|
| the SNMP version specified  |
| pktTimeout  |
| the packet timeout specified  |
| pktRetries  |
| the packet retry count specified  |
| maxOids   |
| the maximum number of OID's specified   |
| toEnd   |
| a boolean flag indicating that the walk should proceed to the end (i.e. don't limit by startOid). |
| timeStarted   |
| the UTC timestamp when the walk was started   |
| timeFinished  |
| the UTC timestamp when the walk was completed   |
| oidCount  |
| the number of OID rows walked   |
| stopReason  |
| text indicating the reason the walk stopped where it did  |

## Table: results

The results table stores a row for each entry of one snmpwalk, as specified by id in the `walks` table.

|   |
|---|
| walk_id   |
| the id of the walk (references walks.id)                          |
| oid   |
| the text of the OID naming the SNMP variable                      |
| type  |
| the ASN.1 type integer for the SNMP variable                      |
| value   |
| the actual, uninterpreted binary value returned by the SNMP agent |

## Accessing the SQLite Data

On Mac OS X 10.4, you can use the built-in `sqlite3` command to access the data in the SQLite database file:

```
$ sqlite3 foo

sqlite> .mode csv
sqlite> select * from walks;

1,"192.168.1.1",161,1.0,0,10000,3,2000,0,1178801645,11788016
85,0,"No answer received"

2,"192.168.1.2",161,1.3,0,10000,3,2000,0,1178801708,11788018
95,2000,"Finished (2000 OIDs found)"
sqlite> select count(*) from results where walk_id = 1;
0
sqlite> select count(*) from results where walk_id = 2;
2000
sqlite> select * from results where walk_id = 2 order by
oid limit 5;
2,"1.3.6.1.2.1.1.1.0",4,"HP J4813A ProCurve Switch 2524..."

2,"1.3.6.1.2.1.1.2.0",6,"+\006\001\004\001\013\002\003\007\0
13\023"
2,"1.3.6.1.2.1.1.3.0",67,"\035\330J\375"
2,"1.3.6.1.2.1.1.4.0",4,"Bill Fisher"
2,"1.3.6.1.2.1.1.5.0",4,"HP ProCurve Switch 2524"
```

There is a Firefox add-on called "SQLite Manager" that opens and displays SQLite database files. This makes it a cross-platform tool, requiring only Firefox 3.5 or greater, plus a small download.

### To install SQLite Manager:

1. Start Firefox 3.5 or newer - Tools -> Add-ons - Click Get Add-ons
2. Enter "SQLite" into the Search field and press Return.
3. Double-click the "SQLite Manager" item to install it. Restart Firefox when instructed.

### To use SQLite Manager:

1. Tools -> SQLite Manager to open it.
2. Use the Database -> Connect Database (within the window) to open a saved SQLite database
3. Click the results table to view it.

# Reference

Use the Developer Guide reference to get the information you need for a wide variety of topics:

- [Intermapper HTTP API](#) - automate Intermapper operations, data import/export, and more.
- [Retrieving Collected Data](#) - access the Intermapper Database to retrieve collected data.
- [Customizing Web Pages](#) - customize any Intermapper web page to suit your needs.
- [Command-line Options](#) - use the command-line interface to automate or streamline maintenance and monitoring operations.

# Intermapper HTTP API

Intermapper provides an HTTP API for retrieving data from, and sending data to the Intermapper server ("exporting" and "importing", respectively). The API allows an external program to use standard HTTP commands (GET, POST) and a straightforward URL syntax to make these requests.

These are the major features of the HTTP API:

- **File Import/Export** gives access to files of the Intermapper Settings folder.
- **Table Import/Export** gives access to the tables in the same formats that are currently available through the Remote Access Import and Export commands.
- **Acknowledgements** - send Basic Acknowledgements to an Intermapper server using HTTP. This allows you to acknowledge down devices from phones, browsers, or scripts.

Through these API interfaces you can accomplish a number of scripting tasks. Two scripts are provided, allowing you to "clone" the Intermapper Settings directory through a script.

**Note:** To use the HTTP API interface, you must connect as an Intermapper user that is a member of the Intermapper Administrators group, as specified in the Users pane of the Server Settings window.

The features of the HTTP API are described in these topics:

[Importing & Exporting Files](#) - How to access the Intermapper Settings directory's file system.

[Importing & Exporting Tables](#) - How to import or export map data directly.

[Acknowledging Devices](#) - How to acknowledge down devices or interfaces using the HTTP API.

[Scripting Examples](#) - Examples of the "clone" scripts and much more.

# Importing & Exporting Files

Most of the files in the Intermapper Settings folder can be accessed through the HTTP API. These include:

- Custom Icons
- Fonts
- Maps
- MIB Files
- Probes
- Sounds
- Web Pages

These folder contents may be exported, but not imported:

- Extensions
- Certificates folder
- Tools

The contents of the following folders are not currently available:

- Chart Data folder
- Intermapper Logs folder
- Deleted and Disabled Maps folders

All other files are served up as binary files with a MIME type of application/octet-stream. Each file has a corresponding URL to retrieve its contents; each folder in Intermapper Settings also has a URL to retrieve the list of URLs for the files in that folder.

All URLs given below are relative to a URL composed of the Intermapper Server address and the webport, as defined in the server settings. For example, in the following discussion a URL of `/~files` would imply the full URL (either http: or https:):

```
http://imserver_address:webport/~files
```

**Example:** The URL above produces a text listing the URLs for the folders in the Intermapper Settings that can be accessed over HTTP; this is provided for the convenience of scripts that might want to access all files.

A request for following URLs provides a text listing of the URLs for the files within the corresponding folder in the Intermapper Settings folder.

| Intermapper Settings folder | Corresponding URL               |
|-----------------------------|---------------------------------|
| Custom Icons                | <code>/~files/icons</code>      |
| Extensions                  | <code>/~files/extensions</code> |



|           |                  |
|-----------|------------------|
| Fonts     | /~files/fonts    |
| Maps      | /~files/maps     |
| MIB Files | /~files/mibs     |
| Probes    | /~files/probes   |
| Sounds    | /~files/sounds   |
| Tools     | /~files/tools    |
| Web Pages | /~files/webpages |

## HTTP file imports

To import these files over HTTP, issue a POST request to the appropriate URL with the file contents as a payload; the MIME type should be application/octet-stream.

### Icons

You can import icons via HTTP as described above. The URL should take the following form:

```
http://imserver:port/~files/icons/Folder/Filename.type
```

If the file type is a valid image file (jpeg or png), it will be available to use immediately.

Sample curl commands (command-line) to use this facility might look like this (should be all on one line):

```
HTTP: curl --data-binary "@sample.png"  
http://localhost:8080/~files/icons/Default/sample.png  
HTTPS: curl -k --data-binary "@sample.png"  
https://localhost/~files/icons/Default/sample.png
```

**Note:** the `-k` option for HTTPS ignores an unsigned certificate.

## Maps

You can import maps or map data using the HTTP API.

A sample curl command line to import a map file should take this form:

```
$ curl --user admin:Pa55w0rd --data-binary  
@/path/to/local/map_file  
http://imserver:port/~files/maps/map_file
```

# Importing & Exporting Tables

## Table-based Import/Export

The table-based functions of the Intermapper HTTP API match the capabilities of the Import -> Data file... and Export -> Data file... commands of Intermapper Remote Access (available from the File menu). These import and export a number of "tables" of information about the devices being monitored. These tables include:

- Devices
- Interfaces
- Vertices
- Maps
- Notifiers
- Users
- Schema

These tables have detailed descriptions in the *Advanced Data Import/Export* section of the [Intermapper User Guide](#). The URLs for importing and exporting have the following format:

```
http://imserver:port/~export/tablename.format? (options)
```

Supported formats are:

- .tab - save as a tab-delimited text file
- .csv - save as a comma-delimited text file
- .xml - save as an XML format file
- .html- display as HTML directly in the browser

The primary option is "fields=...." The list of valid fields are listed in the "schema" export. For example, this query:

```
http://imserver:port/~export/schema.html
```

provides a list of the supported tables and the fields for each table in an HTML format that you can view right in the browser. Other examples include:

```
http://imserver:port/~export/devices.tab
```

provides a list of all devices on active maps as a tab-delimited file. This URL:

```
http://imserver:port/~export/devices.tab?fields=id,name,macaddress,address
```

provides a list of all devices on active maps, but only includes the ID, Name, MACAddress and Address fields.

## Importing Table-based Data

An external program can also import table information with an HTTP POST operation by including the table data as the payload.

```
http://imserver:port/~import/filename
```

The *filename* in this URL is written to the log file, but is otherwise ignored. It is not used to determine the data to import, nor is it used to specify where the data goes. Intermapper examines the directive line of the attached file to determine what information is imported from the file. It follows the same logic that is used when importing data using the Import->Data File... command available from Intermapper Remote Access's File menu.

A sample curl command line to import map data should take this form:

```
$ curl --user admin:Pa55w0rd --data-binary  
@/path/to/import/file http://imserver:port/~import/file
```

# Acknowledging with HTTP

You can perform a Basic Acknowledgement of a device by issuing a POST to a URL of this format:

```
http://imserver:port/mapid/device/deviceIMID/*acknowledge.cgi?message=URL+encoded+string
```

where:

- *mapid* = the mapid of the corresponding map.
- *deviceIMID* = the IMID of the device you wish to acknowledge.
- *message*: requires a URL-encoded text string

There are several ways of finding these values:

- Look in the web interface at the status window for a device, remove the trailing "device.html" at the end, and replace that with "\*acknowledge.cgi?message=....".
- Review the device table (you can view it using the HTTP API) and get the MapId and IMID.

**Example:** This curl command sends the POST with the proper string to acknowledge the device:

```
curl --user admin:Pa55w0rd  
http://imserver:port/mapid/device/deviceIMID/*acknowledge.cgi?message=URL+encoded+text+string -d "dummy post data"
```

## Notes:

- This command responds with a web page, which makes sense when acknowledging through a browser, but less so when using `curl`. As a result, HTML code is sent to `curl`, which sends it to `stdio`. The returned code can be ignored, logged, or parsed as needed.
- The curl parameter `-d "dummy post data"` causes `curl` to send the command as using the HTML POST method, rather than the GET method.

**Example:** This curl command retrieves the full list of devices and each device's address, MapID and IMID

```
curl --user admin:Pa55w0rd  
http://imserver:port/~export/devices.tab?fields=MapId,IMID,address,name
```

**Example:** You can also use this expression in Python to create the URL to POST:

```
"http://imserver:port/%s/device/%s/*acknowledge.cgi?message=
%s" % (mapId, IMID,urllib.urlencode([('message',
messageStr)]))
```

# HTTP API Scripting Examples

Two command line script examples are included here; these scripts provide good examples of the use of the Intermapper HTTP API:

- **Unix shell script** - for Unix/Linux platforms
- **Windows vbscript** - for Windows platforms

## Notes:

- These scripts are not installed on your Intermapper server.
- If you copy the scripts, *make sure you are copying from the online version* of this guide, available at <http://download.intermapper.com/docs/DevGuide/>. Copying from the PDF version may result in unreliable results, caused by unwanted line breaks.
- Both scripts make a copy of the Intermapper Settings directory.
- Both scripts assume they are invoked *from the destination directory* rather than trying to look it up - this is intended to make testing easier. HelpSystems recommends that you create a dummy Intermapper Settings folder someplace and try the synchronization to that (the destination directory doesn't even require that Intermapper is installed, only the script). The remote server does not need to be the same platform as the destination machine.
- The destination directory should **not** have an active Intermapper running in it, as the script assumes that it can replace files at will.

The following URLs are used to access the folders:

| Intermapper Settings folder | Corresponding URL  |
|-----------------------------|--------------------|
| Custom Icons                | /~files/icons      |
| Extensions                  | /~files/extensions |
| Fonts                       | /~files/fonts      |
| Maps                        | /~files/maps       |
| MIB Files                   | /~files/mibs       |
| Probes                      | /~files/probes     |
| Sounds                      | /~files/sounds     |
| Tools                       | /~files/tools      |
| Web Pages                   | /~files/webpages   |

## Unix shell script

**File location:** The Unix shell script assumes that the current working directory is the "Temporary" folder inside the destination Intermapper Settings.

The script requires bash, curl, tr, grep, sed and awk. to be installed.

### Unix script options

```
clone_im.sh [options]
-r [remote_host_name ]
-t [remote_port ]
-u [remote_user ]
-p [remote_password]

Defaults:
remote_host_name =  "localhost"
remote_port = 8080 (this is the Intermapper web access  port)
remote_user = "admin"
remote_password =  "admin"

Example:
clone_im.sh -r nitro.dartware.com -t 8080 -u IMuser -p
UsErpaSS
```

## Windows vbscript

**File location:** The Windows vbscript assumes that the current working directory is the destination machine's Intermapper Settings folder.

### Windows script options

```
clone_im.vbs
/host:[remote_hostname]
/port:[remote_port]
/user:[remote_user]
/password:[remote_password]

Defaults:
remote_host_name (none, must be specified)
remote_port = 80
user (none, uses auto-login unless specified)
password (none, uses auto-login  unless specified)
secure = false
```

Example:

```
clone_im.vbs /host:nitro.dartware.com /port:8080 /user:IMuser  
/password:UsErpaSS /secure:true
```

## Known bugs

- The Unix version does not yet support a "secure" switch.
- The Unix version should not have default username and password, to use the auto-login if that's available.
- The Unix version should require the current directory to be the top-level of the destination Intermapper Settings directory, not the "Temporary" subdirectory.

## clone\_im.sh

```
#!/bin/bash

# Synchronize Intermapper Settings folder from a remote host
# with Intermapper SDK
#
# Requires curl and gnu awk

remote=localhost
port=8080
user=admin
password=admin
auth=
while getopts 'r:t:u:p:' OPTION ; do
    case $OPTION in
        r)    remote="$OPTARG"
              ;;
        t)    port="$OPTARG"
              ;;
        u)    user="$OPTARG"
              ;;
        p)    password="$OPTARG"
              ;;
        ?)    printf "Usage: %s -r remotehost\n" $(basename $0)
              >&2
              exit 2;;
    esac
done

if [ "$user" ] ; then
    auth="--user $user:$password"
```



```

fi

# This script requires Intermapper to be stopped before
running.
#/etc/init.d/intermapperd stop

# Get list of top-level file directories from Intermapper
topdirs=$(curl $auth -s http://$remote:$port/~files | tr '\r'
'\n')

for dir in $topdirs ; do
    # Get list of files in this directory
    filelist=$(curl $auth -s $dir | tr '\r' '\n')

    webdir=$(basename $dir)
    echo "Processing $webdir..."
    localdir=""

    echo $filelist | grep "does not exist" >& /dev/null
    if [ $? != 0 ] ; then

        # Convert the web path into the corresponding Setting
        folder path
        case $webdir in
            icons) localdir="Custom Icons" ;;
            sounds) localdir="Sounds" ;;
            mibs) localdir="MIB Files" ;;
            probes) localdir="Probes" ;;
            tools) localdir="Tools" ;;
            webpages) localdir="Web Pages";;
            fonts) localdir="Fonts" ;;
            extensions) localdir="Extensions" ;;
            maps) localdir="Maps" ;;
        esac

        for file in $filelist ; do
            # Get this file and move it into the proper location
            curl $auth -s -O $file
            filename=$(basename $file)

            # Decode the URL to find the real filename
            # Modified from http://do.homeunix.org/UrlDecoding.html
            to work with gnu awk
            local_filename=$(echo $filename | \
                sed 's/+//g' | \
                sed 's/\%0[dD]//g' | \
                awk '/%/{while(match($0,/\%[0-9a-fA-F][0-9a-fA-
F]/)){$0=substr($0,1,RSTART-1)sprintf("%c",strtonum

```

```
("0x"substr($0,RSTART+1,2)))substr($0,RSTART+3);}}{print}')
```

```
# This version works with BSD awk
# local_filename=$(echo $filename | \
#     sed 's/+/ /g' | \
#     sed 's/\%0[dD]//g' | \
#     awk '/%/{while(match($0,/\%[0-9a-fA-F][0-9a-fA-F]/)){$0=substr($0,1,RSTART-1)sprintf("%c",0+("0x"substr($0,RSTART+1,2)))substr($0,RSTART+3);}}{print}')
```

```
# Make sure the destination directory exists
local_dirname=$(dirname "$local_filename")
mkdir -p "$localdir/$local_dirname"

echo " " $(basename "$local_filename")
mv $(basename $file) "$localdir/$local_filename"
done
fi
done

# Preferences file is separate, since it's stored in the top
level of the Intermapper Settings directory
curl $auth -s -O http://$remote:$port/~files/Preferences
mv Preferences ../

# restart Intermapper
#/etc/init.d/intermapperd start
```

## clone\_im.vbs

Rem Synchronize with a remote Intermapper server

Option Explicit

```
Dim remoteAddr, remotePort, remoteUser, remotePassword,
secureConnection
Dim urlBase, topLevelDirs, dir, fileList, file, fileName,
fileData
Dim webDir, localDir, localFileName
```

```
'*****
*****
' Parse command line arguments
```

```

If WScript.Arguments.Named.Exists("host") Then
    remoteAddr = WScript.Arguments.Named.Item("host")
Else
    WScript.Echo "Must specify a remote Intermapper host to
synchronize with"
    WScript.Quit
End If

If WScript.Arguments.Named.Exists("port") Then
    remotePort = WScript.Arguments.Named.Item("port")
Else
    remotePort = 80
End If

If WScript.Arguments.Named.Exists("user") Then
    remoteUser = WScript.Arguments.Named.Item("user")
End If

If WScript.Arguments.Named.Exists("password") Then
    remotePassword = WScript.Arguments.Named.Item("password")
End If

if (WScript.Arguments.UnNamed.Count > 0) Then
    secureConnection = (WScript.Arguments.UnNamed.Item(0) =
"secure")
End If

'*****
'*****
' Get list of supported folders
urlBase = MakeURLBase(remoteUser, remotePassword, remoteAddr,
remotePort, secureConnection)

topLevelDirs = Split(FetchURL(urlBase, remoteUser,
remotePassword,true), VbCrLf, -1, vbBinaryCompare)

For Each dir In topLevelDirs
    if dir <> "" Then

        ' Convert the top-level name in the URL to the local

```

```
folder name
webDir = Right(dir, Len(dir) - Len(urlBase) - 1)
Select Case webDir
    Case "icons"
        localDir = "Custom Icons"
    Case "sounds"
        localDir = "Sounds"
    Case "mibs"
        localDir = "MIB Files"
    Case "probes"
        localDir = "Probes"
    Case "tools"
        localDir = "Tools"
    Case "webpages"
        localDir = "Web Pages"
    Case "fonts"
        localDir = "Fonts"
    Case "extensions"
        localDir = "Extensions"
    Case "maps"
        localDir = "Maps"
End Select

' Get the files in this folder
fileList = Split(FetchURL(dir, remoteUser,
remotePassword, true), VbCrLf, -1, vbBinaryCompare)
For Each file In fileList
    If (Len(file) > len(dir)) Then
        localFileName = Right(file, len(file) - len(dir) - 1)

        fileName = localDir & "\" & URLEncode(localFileName)
        fileData = FetchURL(file, remoteUser, remotePassword,
false)
        MakeFolderFor fileName
        SaveBinaryData fileData, fileName
    End If
Next
End If
Next

' Preferences file is not contained in a folder on the
server.
fileData = FetchURL(urlBase & "/Preferences", remoteUser,
remotePassword, false)
SaveBinaryData fileData, "Preferences"

' *****
*****
```

```

'*****
*****

'*****
*****

Function URLDecode(str)
    URLDecode = Replace(Unescape(Replace(str, "+", "
    ")), "%2F", "\")
End Function

Function FetchURL(url, user, password, textonly)
    Dim http, result

    Set http = CreateObject("WinHttp.WinHttpRequest.5.1")
    http.Open "GET", url

    If user <> "" Then
        http.SetCredentials user, password, 0
    End If

    http.Send

    if http.Status = 200 then
        if textonly Then
            result = http.ResponseText
        Else
            result = http.ResponseBody
        End If
    Else
        result = ""
    End If

    Set http = Nothing
    FetchURL = result
End Function

Sub MakeFolderFor(filename)
    'Recursively make folders
    Dim fso, i
    Redim dirstack(0)

    Set fso=CreateObject("Scripting.FileSystemObject")
    dirstack(0) = fso.GetParentFolderName(filename)

    While (dirstack(ubound(dirstack)) <> "" )

```

```
Redim preserve dirstack(ubound(dirstack) + 1)

dirstack(ubound(dirstack)) = fso.GetParentFolderName
(dirstack(ubound(dirstack) - 1))
WEnd

For i = ubound(dirstack)-1 To 0 step -1
    If Not fso.FolderExists(dirstack(i)) Then
        fso.CreateFolder(dirstack(i))
    End If
Next

Set fso=Nothing
End Sub

Function MakeURLBase(user, password, host, port, secure)
    Dim protocol, fullhost
    If secure Then
        protocol = "https://"
    Else
        protocol = "http://"
    End If

    If (port <> "") Then
        fullhost = host & ":" & port
    Else
        fullhost = host
    End If

    MakeURLBase = protocol & fullhost & "/~files"
End Function

Function SaveBinaryData(arrByteArray, strFileName)
    Dim objBinaryStream
    Set objBinaryStream = CreateObject("ADODB.Stream")
    objBinaryStream.Type = 1
    objBinaryStream.Open()
    objBinaryStream.Write(arrByteArray)
    objBinaryStream.SaveToFile strFileName, 2
End Function
```

# Retrieving Collected Data

Intermapper Reports Server is a PostgreSQL database that retrieves data from an Intermapper server and saves it for use by external programs.

Although the Intermapper Reports UI is the easiest way to get data from the database, you can connect to the Intermapper Reports Server database using your own techniques. Several short example reports in Crystal Reports and OpenRPT are available, as well as several example perl scripts. The perl scripts require DBI and DBD::pg.

These scripts have been packaged and zipped and placed on our downloads server, and are available at:

[http://download.intermapper.com/sql/sql\\_examples.tar.gz](http://download.intermapper.com/sql/sql_examples.tar.gz)

Please feel free to share your own, as well.

If you want to create your own queries to retrieve data, see [Creating SQL Queries](#).

## Intermapper Database Schemas

The most up-to-date schema for the Intermapper Database is available at:

[https://\[Your Intermapper Database Server URL\]:8182/~imdatabase/schemaddl.html](https://[Your Intermapper Database Server URL]:8182/~imdatabase/schemaddl.html)

It is also available at:

<http://download.intermapper.com/schema/imdatabaseschema.sql>

# Creating SQL Queries

You can create your own SQL queries to retrieve data from Intermapper Database. The `datasample` tables contain the 5-minute, hourly and daily samples derived from the original data values. The recommended approach for retrieving data is to get it from these tables. For more information, see [Intermapper Database Schemas](#).

You can also query individual data values, but this is much slower than querying the `datasample` tables. In Intermapper 5.4 and earlier, individual data values were stored in the `datapoints` table. They are now stored in the `datastore` table. Existing queries on the `datapoints` table must be rewritten to use the `datastore` table instead. *This applies only if you have written queries in this or a related construct:*

```
SELECT FROM datapoint WHERE dataset_id = 5 AND data_time  
BETWEEN a AND b
```

To retrieve data from the `datastore` table, use the `load_data()` function, described below.

## Using the `load_data()` function

Use this function *only* if you have an existing query on the `datapoint` table, or if you actually need the individual raw values. In the overwhelming majority of cases you should query the `datasample` tables as described above, since they're faster and easier to access.

The syntax for the `load_data()` function is as follows:

```
load_data([dataset id],[datetime start],[datetime end])
```

For example, to retrieve data for `dataset_id = 5` between data\_times `a` and `b`, use this recommended syntax:

```
SELECT data_time, data_value FROM load_data(5, a, b)
```

The explicit column list is not required, but it is recommended. If you use `SELECT *` rather than an explicit column list, the function returns a single column of the built-in *composite-value* type, containing both values. You can still reference the values from this composite data type, but you won't be able to treat it as you would a regular Postgres column.

The `load_data()` function acts as a table source, and accepts the built-in Postgres `infinity` and `-infinity` timestamps:

```
SELECT data_time, data_value  
FROM load_data(1, '2011-11-09 00:00:00', 'infinity')  
ORDER BY data_time
```

You can also use `UNION` to combine sources:

---



```
SELECT 5 as dataset_id, data_time, data_value
FROM load_data(5, '2011-11-09 00:00:00', 'infinity')
UNION SELECT 6 as dataset_id, data_time, data_value
FROM load_data(6, '2011-11-09 00:00:00', 'infinity')
UNION SELECT 14 as dataset_id, data_time, data_value
FROM load_data(14, '2011-11-09 00:00:00', 'infinity')
ORDER BY dataset_id, data_time
```

# Customizing Web Pages

Intermapper comes with a set of default web page layouts, and uses them to generate web pages. Use this section to learn how to customize those pages by modifying the files that Intermapper uses to create the pages delivered by its web server.

Intermapper's built-in web server generates pages based on files in its Web Pages folder. See [Web Pages Folder](#) for more information.

When a web request is received, Intermapper finds a corresponding file (called a *target file*) to use as the response. The target file is then formatted according to information specified a *template file*. The resulting file is returned to the user's web browser.

Intermapper uses the following elements to control the appearance of the web pages returned from its web server:

- [Target files](#) - Contain the main text of the various pages sent by the server
- [Template files](#) - Control the overall format of the web pages
- [Directives](#) - Commands within files to control the formatting of the web pages
- [Quoted links](#) - Make it easy to create links to other pages.
- [Macros](#) - Elements you can insert in your templates and target files to show blocks of useful Intermapper information.
- [Web Pages Folder](#) - Controls which web pages are available to Administrators and Guests.
- [Mime Types](#) - Associates templates or target files with specific MIME types.

**Tip:** The target and template files are simply text files. You may edit them with any text editor. On certain platforms, you must have the correct permissions to edit them.

## Reloading Changed Web Page Files

The changes you make to these files do not take effect until Intermapper reloads them.

**To force Intermapper to reload the Web Page files:**

1. From the Edit menu, choose **Server Settings...**
2. From the Server Configuration category, choose **Web Server**. The Web Server settings panel appears.
3. Stop and then restart the Web Server. The changed web pages are reloaded.

# Target Files

When InterMapper receives a request for a web page, the requested URL is parsed to determine the target of the request. This *target file* contains the text content of the desired page. The target file may contain HTML markup if desired.

In addition to the page's text, the target file can contain these other elements:

- [Directives](#) - Commands that describe or modify the way a page should be displayed.
- [Quoted Links](#) - Provide a quick way to create a link to another page using its name, rather than specifying its full URL. If a string is placed in double-quotes (") and the text matches the title of another InterMapper web page, a link is created.
- [Macros](#) - InterMapper variables that are replaced with text or formatted HTML in the final web page. The macro may be replaced with a static string, a device's name or network address, the contents of another file, or other information. Macros are composed of keywords and optional parameters, and are enclosed in "\${...}".

## Target File Example:

```
#title "This is a test page"
This is some text to be displayed in a
web page. The page's title is "This is a test page", while
the remaining
text is displayed in the "body" of the page. The text may
also contain
plain text, HTML tagged text such as <b>bold</b> and
<i>italic</i>,
and macros, such as the ${date} macro, which displays today's
date.
```

- The first line is a directive that sets the title of the page to be displayed.
- The text between double-quotes is placed in <title>...</title> tags in the resulting web page.
- The remainder of this example is placed in the <body>...</body> section of the resulting page. The macro \${date} will be replaced by the current date when the page is displayed.

## Quoted Links

Note that the text "This is a test page" will be displayed as a link to its own page, since it is a string in quotes that matches the #title of a web page (its own). Note, too, that the text "body" could be a link to a page with a title of "body". It is not an error if no such page exists: in that case, InterMapper displays the quoted string in place. For more information see [Quoted Links](#).

# What Happens When a Target File Is Read?

As the target file is read, Intermapper processes the directives, then the expands the macros and creates the <A

  HREF= . . . > tags for any quoted links it encounters. The web server does not insert any white space or paragraph marks (such as <P>) when it encounters carriage returns, etc.

## Built-in Target Files

Intermapper provides a number of built-in target files. These file names all begin with "!", and are required because Intermapper refers to them explicitly. Here are some of the built-in files:

- **!index.html** - Displays the default page, when none is specified in the URL.
- **!document.html** - Displays a graphical image of the specified map.
- **!network.html** - Displays detailed information about the specified network.
- **!device.html** - Displays detailed information about the specified device.
- **!link.html** - Displays detailed information about the specified link.
- **!chart.html** - Displays the specified strip chart.

**Note:** The !network.html, !device.html, !link.html, and !chart.html files are targets intended to display information about a specific network, device, link or chart. The macros that display lists of maps, networks, devices, or charts create links to these targets. The easiest way to create custom versions of these targets is to edit them directly.

# Template Files

To allow all the web pages to have the same look, InterMapper uses *template files* to control the formatting of pages. A template file is composed of HTML commands that provide the skeleton for a web page. In addition, template files often contain macros and quoted links that are replaced by appropriate text when the page is generated.

## Template File Example

Here is a simple template file that could be used with InterMapper:

```
        <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML
3.2//EN">
<HTML>
<HEAD>
    <TITLE>${pagetitle}</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
    ${imageref:logo.gif}
    ${bodytext}
    ${include:footer.incl}
</BODY>
</HTML>
```

This sample contains several important macros:

- **\${pagetitle}** - Replaced with the text of the #title directive of the target file.
- **\${bodytext}** - Replaced with the body text of the target file, that is, everything from the target file that is not a directive.
- **\${imageref:logo.gif}** - Replaced with an `<img . . . >` tag that refers to the logo.gif file in the ~GuestImages folder on the InterMapper server.
- **\${include:footer.incl}** - Replaced with the contents of the file named 'footer.incl'.

# Directives

A *directive* is a special command interpreted by the web server to control the way a page is formatted.

- Directives must start with a "#" in the first column.

## Summary of Directives

Intermapper has several directives that can change the way pages are formatted. They are defined below:

|                   |  |
|-------------------|--|
| <b>#template</b>  | <pre>#template "othertemplate.html"</pre> <p>A target file may specify a template file with the <code>#template</code> directive. The <code>#template</code> directive is optional; if none is present, Intermapper uses the file named <code>!template.html</code> as the page's template.</p>  |
| <b>#title</b>     | <pre>#title "This is a Test Page"</pre> <p>The text in quotes becomes the title of the page - it enclosed in <code>&lt;title&gt;...&lt;/title&gt;</code> tags in the generated page. The <code>#title</code> directive also provides a destination for quoted links on other pages.</p> <p>Every target file must have a <code>#title</code> directive to give it a name.</p> <p>You may include a macro within the quoted text of this directive. This is useful for inserting the device name or other information into the title of the web page.</p> |
| <b>#alt_title</b> | <pre>#alt_title "Test Page"</pre> <p>The optional <code>#alt_title</code> directive provides a way to give a page an alternate name that can be used with a quoted link.</p>   |

**#filename**      `#filename "otherpage.html"`

The optional `#filename` directive causes InterMapper to treat the file as if named with the quoted string.

For example, a target file named "xindex.html" could have a directive of

`#filename "!index.html"`. This causes the target file to be used in place of the file named "`!index.html`" *if its version number is higher*. This can be useful for debugging, or for creating alternate versions of pages.

**#version**

```
#version "2.1"
```

The optional `#version` directive determines which file is used when there are two or more instances of the same filename (as a result of using `#filename` directives.)

The optional `#version` directive is used to break "ties" between several files having the same name to determine which should be used. This comes into play in several cases:

- InterMapper has its own internal copy of the web template files, which it uses to create the original set on disk.
- If you edit one of the default web template files, you could change its `#version` to make it take precedence over InterMapper's built-in copy.
- If you edit the file without changing the `#version`, its date-last-modified value is later, causing it to take precedence over InterMapper's built-in copy.
- If you install a new version of InterMapper with updated web files, the `#version` value is incremented, causing the built-in copy to take precedence over the user-edited files,

**Note:** The user files are not overwritten.

- Through the use of the `#filename` directive, it is possible to have two files on-disk that have "essentially" the same name. One could be named "foo.html", and the other (named something else) can use the `#filename` directive to set its "virtual filename" to "foo.html". In such a case, the `#version` is used to determine which file takes precedence.



Version numbers must be in a 'digit.digit' format. InterMapper uses the file with the highest version number. This is useful for debugging as well as experimenting with alternate pages.

If `#version` directive is not present in the file, the default version, "1.0" is used.

### **#redirect**

```
#redirect "otherpage.html"
```

The `#redirect` directive causes the InterMapper to find `otherpage.html` and use that in place of the original target file.

This can be used to force a well-known page (such as `!index.html`) to display a user-selected page.

**Note:** This directive creates a static redirection that works only for web pages that exist on the disk when the web server is started. To redirect to web pages that are generated dynamically by InterMapper (such as map web pages), use the HTML refresh meta tag instead.

### **#target**

```
#target "window_name"
```

The `#target` directive forces a page to be opened in a new window named "window\_name".

When generating the web page, InterMapper generates an HREF link with a `...target = "window_name" ...` reference. This causes the detailed information to appear in a separate window when the user clicks a map's device or link.

# Quoted Links

You can create a link to another page by entering the page's title in double-quotes. For example, the text "Test Page" creates a link to a page with a `#title` or `#alt_title` directive that contains the text "Test Page".

**Note:** Two target files may have the same `#title` or `#alt_title`. When this happens, Intermapper chooses one of the target files, but you cannot predict which one is chosen.

## Preventing a Quoted String From Becoming a Link

If you place a string in quotes, and the string that does not match another page's `#title` or `#alt_title`, Intermapper displays the quoted string as-is.

You may want text to appear in quotes, even when the text matches another page's `#title` or `#alt_title`. (Remember, you can create quoted links only to pages which have `#title` or `#alt_title` directives, and only quoted text that matches one of those directives results in a link.)

**To prevent a string in quotes from being interpreted as a quoted link:**

- Place backslashes ("`\`") in front of *both* the first and second quote character.

# Macro Reference

A macro is a text string with the format `${macroname:other-information}`. The *macroname* is required, and some macros use or require *other-information* which follows the ":". The entire macro is replaced by the appropriate text when the page is generated.

Macros fall into these general categories:

- [The Include Macro](#)
- [Macros that generate "content" on an Intermapper web page](#)
- [Macros that describe Intermapper and its environment](#)
- [Macros to place images onto a page](#)
- [Macros that control the interval between page refreshes](#)
- [Macros related to links and URLs](#)

## The Include Macro

Your template files and target files may include other files.

`${include:file-to-be-included.html}` The named file is inserted into the web page. The file must be in the same folder.

## Macros that generate the "content" of an Intermapper web page

Intermapper often uses these macros either as the `${bodytext}` of the page, or as a major part of a page's contents. All the macros below work on the map named in the request URL. If the URL is for a page in the ~admin directory, Intermapper displays information about all items in all maps.

|                              |   |
|------------------------------|---|
| <code>\${fullstatus}</code>  | shows a list of all the devices and links for the map named in the URL.                           |
| <code>\${errorstatus}</code> | Shows only the devices and links which are in warning or alarm states, or down for the named map. |

|   |   |
|---|---|
| <b><code>\${errorstatus_orig}</code></b>          | <p>Output the original "errors" status report. Differences from <code>\${errorstatus}</code>:</p> <ul style="list-style-type: none"> <li>• <code>\${errorstatus_orig}</code> doesn't show device alarms.</li> <li>• <code>\${errorstatus}</code> first outputs interfaces in error, then interfaces with high utilization.</li> </ul> <p><code>\${errorstatus}</code> lists interfaces in random order.</p>   |
| <b><code>\${currentoutages}</code></b>            | Shows the list of current outages -- devices or links that are currently in warning, alarm, or are down in the named map.   |
| <b><code>\${currentlinkoutages}</code></b>        | Output a table of current interface outages. The table's column names are <i><b>Date</b></i> , <i><b>Time</b></i> , <i><b>Interface</b></i> and <i><b>Duration</b></i> .  |
| <b><code>\${previousoutages}</code></b>           | Shows the list of devices that had been listed as outages but have since returned to normal.  |
| <b><code>\${previousoutages:hours=xx}</code></b>  | Shows the list of previous outages within the last xx hours.  |
| <b><code>\${previousoutages:maxrows=x}</code></b> | Shows a list of the last x previous outages.  |
| <b><code>\${maplist}</code></b>                   | Shows an HTML unnumbered list (<UL>) of the maps available.   |
| <b><code>\${chartlist}</code></b>                 | <p>Outputs a sorted list of charts from the current context, one per line with each line preceded by a &lt;LI&gt; tag. You are required to supply your own &lt;UL&gt; or &lt;OL&gt; tags. Each chart title is a hyperlink to the related chart web page.</p> <p>Within an administrator context, <code>\${chartlist}</code> generates a list of all charts. In a "per-map" context, <code>\${chartlist}</code> generates a list of charts from the current map.</p> |

|   |   |
|---|---|
| <code>\${chartname}</code>                        | Outputs the title of the chart related to the current web page. If you are not on a chart-related page, the output is "". |
|   | <b>Note:</b> Similar to <code>\${mapname}</code>  |
| <code>\${maplistwithcharts}</code>                | Shows an HTML unnumbered list of the maps available, with sub-lists of the charts for each map.                           |
| <code>\${include:file-to-be-included.html}</code> | Inserts the specified file into the web page.   |

## Miscellaneous macros that describe Intermapper and its environment

|                                     |   |
|-------------------------------------|---|
| <code>\${abouthtml}</code>          | Shows the "About" page with the current version of Intermapper.   |
| <code>\${statshtml}</code>          | Shows Intermapper's statistics: uptime, memory usage, etc.  |
| <code>\${httpuserid}</code>         | The name the user typed when asked for authentication.  |
| <code>\${httplocaladdress}</code>   | Output the IP address of the web server's side of the connection. If the Intermapper server is multi-homed, this will be the local side IP address of the current TCP connection.<br><br><b>Note:</b> Use caution with this address; URL's produced using this address may break in NAT situations. |
| <code>\${httpremoteaddress}</code>  | The IP address of the remote browser.   |
| <code>\${intermapperaddress}</code> | The IP address of this Intermapper server.  |
| <code>\${version}</code>            | The version of this copy of Intermapper.  |
| <code>\${date}</code>               | The current date.   |
| <code>\${time}</code>               | The current time.   |
| <code>\${timestamp}</code>          | Timestamp in Unix epoch format.   |

|   |   |
|---|---|
| <code>\${imagesuffix}</code>                | Set to ".png" if the web client can display PNG images, or ".jpeg" otherwise.   |
| <code>\${telnetserverurl}</code>            | The <code>telnet:</code> URL that connects to this Intermapper Telnet server.   |
| <code>\${webserverurl}</code>               | The <code>http:</code> URL that connects to this Intermapper server.  |
| <code>\${mapname}</code>                    | The current map's name.   |
| <code>\${deviceaddress}</code>              | The IP or AppleTalk address of the particular device. For anything that isn't a device, an empty string is returned.  |
| <code>\${deviceid}</code>                   | Outputs the device ID of the device related to the current page, in the form: "gMMMM-rNN". If the current page is not device-related, output "".  |
| <code>\${devicelist}</code>                 | Outputs a table showing the device list for the current context. The table's columns are <i>Status</i> , <i>Name</i> , <i>Condition</i> , <i>Date</i> , <i>Time</i> , <i>Probe</i> , and <i>Port</i> .<br><br>Within an administrator context, <code>\${devicelist}</code> generates a list of all devices. In a "per-map" context, <code>\${devicelist}</code> generates a list of devices from the current map. |
| <code>\${devicelist_kml}</code>             | Generates a device list in <a href="#">KML format</a> for use by Google Earth.  |
| <code>\${devicename}</code>                 | The DNS name or AppleTalk NBP name of the particular device. It is an empty string for anything that isn't a device.  |
| <code>\${ifadmin: ADMIN : NONADMIN }</code> | Outputs ADMIN if the user has admin privileges. Otherwise outputs NONADMIN.   |
| <code>\${pagetitle}</code>                  | Displays the value set by the <code>#title</code> directive   |
| <code>\${SetNameFieldWidth:xx}</code>       | Set the width of the name field. Intermapper pads the name up to xx characters wide. Use -1 to set the width of the field to the width of its contents. The default width is 20 chars.  |

# Macros to place images onto a page

**`${imageref:IMAGEFILE [tags]}`** Creates an `<img ... >` tag to place an image on the page.  
Example:

```
${imageref: photo, class='grade4'}
```

outputs

```
<IMG SRC="/images/photo.gif"
class='grade4'>
```

## Notes:

Unlike every other macro, this one uses a comma-delimiter in the parameter section instead of a colon.

This macro searches the images folder alphabetically for the first file whose name matches the IMAGEFILE parameter. If you have two files, "photo.gif" and "photo.png", photo.gif is found first.

**`${imagesuffix}`** Set to ".png" if the web client can display PNG images, or ".jpeg" otherwise.

**`${intermapper logo}`** Creates an `<img ... >` tag that includes the "Made with Intermapper" logo image.

**`*chart`** Displays a strip chart, generally has a suffix of `${chart}.${imagesuffix}` to send the desired format for the client's browser. Takes parameters of width (for the chart, in pixels) and the other parameters of the URL.

Usage:

```
<IMG
SRC="*chart.${imagesuffix}?${clientwidth}&${
httpparams}">
```

or

```
<IMG
SRC="*chart.${imagesuffix}?width=300&${http
arams}">
```

|                        |   |
|------------------------|---|
| <b>*popuptext.html</b> | <p>Displays the contents of the current device, network, or interface Status Window (these were formerly called "pop-up windows") as HTML.</p> <p>Usage:</p> <pre>     \${include: *popuptext.html}, generally enclosed in     &lt;pre&gt; ...&lt;/pre&gt; tags. </pre>   |
| <b>*map</b>            | <p>Displays an image of the devices, networks, and links (the "foreground") of the selected map against a transparent background. The objects in this image match with the *imagemap.html, below. Takes an option of a timestamp, to provide for auto-refresh.</p> <p>Usage:</p> <pre>     &lt;img     src="/\${httpdocument}/document/main/*map.\${i     magesuffix}?\${timestamp}"&gt; </pre> |
| <b>*mapbg</b>          | <p>Displays the background image of the selected map. This provides the customer-selected background to the map as an image. It takes an option of a timestamp to provide for auto-refresh.</p> <p>Usage:</p> <pre>     &lt;img     src="/\${httpdocument}/document/main/*mapbg?\${     timestamp}"&gt; </pre>  |
| <b>*imagemap.html</b>  | <p>Displays an HTML imagemap that corresponds to the map image; clicking in the image, follows the links in the (automatically-generated) image map.</p> <p>Usage:</p> <pre>     \${include:/\${httpdocument}/document/main/*im     agemap.html} </pre>   |

**Note:** The web pages combine \*map with the \*mapbg and \*imagemap to create a <div> that superimposes all three items into a single visual unit. See [Intermapper Settings/Web Pages/PerMapHTML/map.html](#) for an example.



## Macros that control the interval between page refreshes

Intermapper's web server can automatically refresh a particular web page at a desired interval. Include these tags on your page to take advantage of this facility.

|  |   |
|--|---|
| <b><code>\${htmlrefreshmetatag}</code></b>     | Either an empty string or the previous refresh choice from the web client. (Inserts a <code>&lt;meta http-equiv="refresh" . . .&gt;</code> tag on the resulting page.)  |
| <b><code>\${htmlrefreshmetaoptions}</code></b> | The option list that a web client can choose from. The current <code>\${htmlrefreshmetatag}</code> value is selected. Note that your HTML template should supply the<br><code>&lt;form&gt;&lt;select&gt;...&lt;/select&gt;&lt;/form&gt;</code> surrounding this<br><code>\${htmlrefreshmetaoptions}</code> macro. |
| <b><code>\${jsrefreshoptions}</code></b>       | The option list that a web client can choose from, generated with JavaScript. The current <code>\${htmlrefreshmetatag}</code> value is selected. Note that your HTML template should supply the<br><code>&lt;form&gt;&lt;select&gt;...&lt;/select&gt;&lt;/form&gt;</code>   |

## Macros related to links and URLs

These macros all return a fully-escaped string, that is, a space character (" ") will be replaced with a %20; a "?" with %3F; etc.

Here is a sample URL. The result of using this URL is shown in parentheses after each macro:

`http://localhost/Map1/device/192.168.0.1%3ASNMP/!device.html`

|                                      |   |
|--------------------------------------|---|
| <b><code>\${webpageurl}</code></b>   | The full URL of the requested web page. (e.g., the full URL as shown above).  |
| <b><code>\${httppath}</code></b>     | The full path to the file requested (e.g., <code>"/Map1/device/192.168.0.1%3ASNMP/!device.html"</code> ).               |
| <b><code>\${httpdocument}</code></b> | The top level directory of the page requested. Also an alias for <code>\${mapname}</code> (e.g., <code>"Map1"</code> ). |

|  |   |
|--|---|
| <b><code>\${httpclass}</code></b>        | The second level directory of the page requested (device, chart, link, document, network) (e.g., "device").   |
| <b><code>\${httpinstance}</code></b>     | The third level directory of the page requested (e.g., "192.168.0.1%3ASNMP").   |
| <b><code>\${httpmethod}</code></b>       | The fourth-level part of the page requested (e.g., "!device.html").   |
| <b><code>\${httpinstancepath}</code></b> | A concatenation of <code>\${httpdocument}</code> , <code>\${httpclass}</code> , <code>\${httpinstance}</code> separated by "/" (e.g., "/Map1/device/192.168.0.1%3ASNMP").   |
| <b><code>\${httpparam: NAME}</code></b>  | <p>Outputs the value of the HTTP parameter specified by NAME. An HTTP parameter is one passed with the originating GET request, affixed to the URL following a question mark. If there is no HTTP parameter by the given name, outputs "".</p> <p>Example, given the following URL:</p> <pre>http://www.example.com/TestMap/?color=red&amp;style=bold</pre> <p><code>\${httpparam: color}</code> outputs "red"<br/> <code>\${httpparam: style}</code> outputs "bold"<br/> <code>\${httpparam: font}</code> outputs "" (because the parameter doesn't exist)</p> |
| <b><code>\${httpparams}</code></b>       | <p>Outputs all the HTTP parameters from the originating GET request in their original format. If there were no parameters attached to the original request, output "".</p> <p>Example, given the following URL:</p> <pre>http://www.example.com/TestMap/?color=red&amp;style=bold</pre> <p><code>\${httpparams}</code> outputs "color=red&amp;style=bold".</p>  |

**`${httpparams_endchart}`** `${httpparams_endchart}` - Replaces the value of the "endtime" parameter with the chart's last time. (For scrolling to the end of the chart).

**`${httpparams_nextchart}`** `${httpparams_nextchart}` - Replaces the value of the "endtime" parameter with a new time value that effectively scrolls the chart one page into the future.

**`${httpparams_prevchart}`** `${httpparams_prevchart}` - Replaces the value of the "endtime" parameter with a new time value that effectively scrolls the chart one page into the past.

`${httpparams_startchart}` - Replaces the value of the "endtime" parameter with the chart's starting time. (For scrolling to the beginning of the chart).

`${httpparams_timescale: VALUE}` - Replaces the value of the "timescale" parameter with the specified value.

**Note:** These five macros are nearly identical to `${httpparams}`. They implement support for chart scrolling and scaling in the web interface. They generate output *only* within a web page associated with a chart.

**`${anchor: value}`** Sets the current "anchor" value.  
**`${attr}`** Outputs the current "anchor" parameter value as set using `${anchor: value}`. If no "anchor" value has been set, output is "".

**Note:** These two macros are closely related. `${anchor}` sets the value; `${attr}` retrieves it.

Example:

```
${anchor:class="header"}
  <A HREF="maplist.html" ${attr}>Map
List</A>
  <A HREF="${TelnetServerURL}"
${attr}>Telnet</A>
  <A
HREF="${WebServerURL}"${attr}>Home</A>
${anchor:}
```

In this example, the anchor is set to 'class="header"'. Then the `${attr}` macro is used to place the attribute string in each link. Afterward, `${anchor:}` sets the anchor to an empty string.

# Web Pages Folder

Web target files and template files are in the *Web Pages* folder within the *Intermapper Settings* folder. Except for the folders described below, the Intermapper web server serves out only those files located in the top level of the *Web Pages* folder.

## Overriding the Built-in Pages

Intermapper ships a single zip archive named *BuiltinWebPages.zip*.

In order to customize pages, you need to create a directory structure that matches the structure within *BuiltinWebPages.zip*. Any files you place in those folder override pages of the same name in the zip archive.

## Contents of BuiltinWebPages.zip

The *BuiltinWebPages.zip* file contains four folders:

### ~AdminHTML

This folder contains HTML templates for pages that show the overall status of the Intermapper program. People who have access to these pages may also view all the separate map pages. You can access these files from the default web URL, or by using a URL in this form:

```
http://intermapper.domainname.com/~admin/filename.html
```

### ~GuestHTML

This folder contains HTML templates for reporting errors such as missing or invalid file names, and for responding to web clients who are not authorized for the web server. These files bypass the usual access list mechanism; you can access them using this URL form:

```
http://intermapper.domainname.com/~error/filename.html
```

### ~GuestImages

This folder contains images used by the Intermapper web server. These images may be placed in a target or template file using the `${imageref: ... }` macro.

### PerMapHTML

This folder contains HTML templates that are used when displaying a map's information. To view a specific document's information, use this URL form:

```
http://intermapper.domainname.com/docname
```

The zip archive also contains some supporting files, including JavaScript files, located in the root of the folder.

## How the Web Page Files are Used

### Main Web Page

The main web page is the `~admin/!index.html` target file. When an unqualified URL request arrives (that is, a request for `"/`", without any additional path of file information), Intermapper sends out the file specified by `~admin/!index.html`.

### Main Template file

By default, all target files use the same template, `!template.html` (note the exclamation point at the beginning of the filename). A target file may specify a different template file by using the `#template` directive.

### Default HTML page

For both the `"~AdminHTML"` and `"PerMapHTML"` folders, the default HTML page is `!index.html`. A request for `http://intermapper.domain.com/` is treated like a request for

```
http://intermapper.domain.com/~admin/!index.html.
```

Similarly, a request for

```
http://intermapper.domain.com/docname
```

will be treated like a request for

```
http://intermapper.domain.com/docname/document/main/!index.html.
```

# MIME Types

You can associate a template or target file's suffix with MIME-type information you want to send with the file. You create this association by placing a file named "mimetypes" at the top level of the *Web Pages* folder.

Below is a sample mimetypes file:

```
# Sample MIMEtypes file
# Format is: <file-suffix> <whitespace> <MIME-descriptor>
wml      text/vnd.wap.wml
wmls     text/vnd.wap.wmlscript
wbmp     image/vnd.wap.wbmp
wbxml    application/vnd.wap.wbxml
wmlc     application/vnd.wap.wmlc
wmlsc    application/vnd.wap.wmlscriptc
```

# Tip for Calling Charts

When calling charts through the web server, you can control the height and width of the chart by passing parameters with the URL. You can also control the time scale.

## **To control the height and width of the chart:**

- Enter height & width tags for charts as

```
http://.../!chart.html?height=xxx&width=yyy
```

- Tip: To enter different time scale,

```
http://.../!chart.html?timescale=XXXXX where the time value is in minutes.
```



# Command-line Options for Intermapper

You can call Intermapper and Intermapper Remote Access from a command line, and control a significant number of functions. This can be useful for automating the updating of maps, or for various testing purposes.

For detailed information on the use of the command-line for scripting Intermapper and Intermapper Remote Access, see *Command-line Options for Intermapper* and *Command-line Options for Intermapper Remote Access* in the User Guide's *Reference* section as well as [Intermapper HTTP API](#) in this manual.

# Index

## \$

`${bodytext}` 227

## A

Abs 95

Add 134

Comments 134

Addition, Subtraction 95

ADDRESS 166

Address\_type 19, 144

Admin 227

Admin/!index.html 238

AdminHTML 237

Administrators 218

ALARM 134

Alarm Point 84, 87, 92

Alarm/warning 24

Alert 17

ALRM Response 145

Alt\_title 222, 226

anchor 236

APC-UPS MIB 73

AppleTalk 143, 230

AppleTalk datagrams 13

AppleTalk NBP 230

Argument Format 134

ASN.1 126

Auth 191

Authpassword 191

autorecord 27

## B

B/Bold 44

B5/Custom TCP Information/OP 144

Base-64 139

Base64 102, 139

Big-endian 102

Bitand 95

Bitlshift 95

Bitor 95

Bitrshift 95

Bitwise 94

functions 94

Bitxor 95

BODY 221

BODY BGCOLOR 221

Bodytext 221

Build/compile 148, 166

Built-in Macros 139

Built-in Numeric Functions 95

Built-in String Functions 95

Built-in Target Files 220

Builtin 18

## C

CALCULATION 66

Calculation Variables 66

- Calling Charts 240
- Carriage Return 135
- Case-insensitive 136
- Case-Sensitivity 136
  - Controlling 136
- Changed Web Page Files 218
  - Reloading 218
- Chart 64
- CHART LEGEND 64
- Chart.html 220
- Chart.html?height 240
- Chart.html?time 240
- Chartable 68
- Chartlist 228
- chartname 229
- Com.dartware 19, 22, 144
- Com.dartware.tcp.custom 19, 22, 144
- Comma-separated 64
- Comma-separated list 19
- Command-line 18, 147
  - Create 147
- Command-line Probe 147
  - Installing 149
- Command Line Interface 241
- Command Line Nagios Plug-in Example 161
- Command Line Probes 147
- Comments 134
  - Adding 134
- Comparisons 71
  - Creating 71
- Concatenation 94
- Conditional Expression 94
- CONN 144
- Consequently 190
- Controlling 136
  - Case-Sensitivity 136
- Cos 95
- CPU 62
- CR-LF 144
- Creating 71, 147, 166
  - Command-line 147
  - Comparisons 71
  - Nagios Probes 166
- Currentlinkoutages 228
- Currentoutages 228
- Custom-snmp 18
- Custom Probe File Format 15
- Custom SNMP Probes 18, 41, 62, 81
  - Header Section 20
- Custom TCP 18, 22, 25, 144
- Custom TCP script 144
  - Dartware-provided probe 144
- Customizing 41, 218
  - Status windows 41
  - Web Pages 218
- Customtimer 142
- CVSPASSWORD 139

## D

Daemon 147  
 Dartware MIB 126  
 Datagram 134  
 Datagrams/sec 41, 81  
 Date-last-modified 224  
 Debug file 191  
 Debug window 190  
     Open 190  
     See 191  
 Debugyyyyymmddhhmm.txt 191  
 Default  
     Per-second 65  
 DEFAULT 65  
 Default HTML 238  
 Default Values 138  
 Defines 64  
 Defines:OIDs 64  
 DES 191  
 Des|none 191  
 Description 22  
 Device.html 220, 233  
 Deviceaddress 230  
 Deviceid 230  
 Devicelist 230  
 Devicename 230  
 Digit.digit 225  
 Directives 222

DISCONNECT 145  
 display\_name 20  
 DNS 22, 230  
 DOCTYPE HTML PUBLIC 221  
 Document.html 220  
 DONE ALRM 145  
 DONE DOWN 145  
 DONE OKAY 145  
 DONE WARN 145  
 Double-precision 94  
 Double Quote 136  
 DOWN 13, 134, 144  
     set 144  
 DOWN Response 145

## E

Edit menu 218  
 ELSE 135  
 End 191  
     MIB 191  
 Endian 102  
 Enter 192  
     snmpwalk 191  
 Enterprise 6306 126  
 Equality Tests 95  
 Error Conditions 193  
 Errors/minute 41, 81  
 Errorstatus 227  
 Errorstatus\_orig 228

Example TCP Probe FileThe 144

Excel 94

EXIT\_CODE 153

Exp 94

Expr 94

EXPT 145

Extract 101

    substring 101

## F

FALSE

    value 94

File-to-be-included.html 227

File Names 22

FLAGS 21

Fmt 96

Folder Structure 237

Footer.incl 221

Format 62, 239

Formfeed 135

Forwarded datagrams 64, 66

From

    Server Settings 218

Fullstatus 227

FUNCTION 94

Function Descriptions 96

FUNCTION substr 101

Functions 94

    bitwise 94

## G

GB/Custom TCP Probe/P 144

Geneva 43

Get-Next-Request 73

Get Info window 17

GetNextRequest 190

GOTO 135, 145

GOTO Command 137

GuestHTML 237

GuestImages 221, 237

Guests 218

## H

Handling 134

    Script Failures 134

HEAD 221

Header Parts 18

Header Section 20

    Custom SNMP Probes 18

HelpSystems-provided probe 144

    Custom TCP script 144

Hexadecimal 65

Hexadecimal - Displays 62

Hexadecimal Number 136

Horizontal Tab 135

Hosting

    Intermapper 147

HREF 220, 225

HTML 45, 140, 219, 221-222, 227, 237  
    shows 227  
    Use 222  
Htmlrefreshmetaoptions 233  
Htmlrefreshmetatag 233  
httpparams\_endchart 235  
httpparams\_nextchart 235  
httpparams\_prevchart 235  
httpparams\_startchart 235  
httpparams\_timescale 235  
Human\_name 19, 144  
Hyperlinks 43  
    making 43

I

ICMP 21, 143  
    send 21  
IDLE 138, 145  
IDLELINE 145  
If ap cups 73  
Ifadmin 230  
IfDescr 190  
IfTable 192  
IfXTable 192  
Imagefile 231  
Imageref 221, 231, 237  
Imagesuffix 230  
Img 221, 231

Include 62, 94  
    CPU 62  
    SNMP MIB 94  
Index.html 220, 223, 238  
Indicates 191  
    SNMP 191  
Installing 149  
    Command-line Probe 147  
Integer 95  
Intermapper Logs 192  
Intermapper Remote 241  
Intermapper Server  
    testing 241  
Intermapper Settings 22, 166, 192, 237  
Intermapper.domain.com 238  
Intermapperaddress 229  
Intermapperlogo 231  
Invoking 192  
    snmpwalk 191  
IP 17, 62, 66, 144, 147, 166, 191, 229  
IP-address 191  
IP-address startOID 191  
IpForwarded datagrams 71  
IpForwDatagrams 41, 64, 66, 81  
IpInHdrErrors 41, 64, 66, 81

J

Jpeg 230  
jsrefreshmetaoptions 233

Jumping 138

Label 134

### L

Label 134

Jumping 138

Label\_name 138

LDAP 13

Len 96

Level 237, 239

Web Pages 237, 239

Limit 22

Macintosh file 22

LINE 135, 144

Link 226

Link.html 220

Link/alias/shortcut 166

Little-endian 102

localhost/Map1/device/192.168.0.1%3  
ASNMP/!device.html 233

Logical And 94

Logical Or 94

Logo.gif file 221

### M

M1++/Big 44

Mac OS 147

Macintosh file 22

limit 22

Macro Reference 227

Macroname 227

Main Template file 238

Main Web Page 238

maintenance mode 86

Maplist 228

Maplistwithcharts 229

Mapname 230

Markup Commands 43

Markup Tag Summary 43

Matches 72, 94, 136

String 62, 94, 134

Maxrows 228

Maxvars 73

MD5 191

Md5|sha|none 191

Measuring 142

Response Times 142

Meta 225

Meta http-equiv 233

MIB 62, 64, 191

end 191

MIME-descriptor 239

MIME-type 239

MIME Types 239

Mimetypes 239

Mimetypes file 239

Min 94

MINIMAL 21

Modulo 95

Monaco 43  
 Monospace 43  
 Monospaced 41, 43, 81  
 Msecs 145  
 MTCH 135, 145  
 Multiplication, Division 95  
 MUST 135

## N

N/A 190  
 Nagios 148, 166  
     uses 147  
 Nagios Plugins 148, 166  
 Nagios Probes 166  
     Creating 166  
 Nagios Template 166  
 Name 147  
     program/script 147  
 Name/value 23  
 Network.html 220  
 NOICMPFALLBACK 21  
 NOLINKS 21  
 Nomib2 73  
 Non-interpreted 149  
 NoSuchName 190  
 NTCREDENTIALS 22  
 NULL 190  
 Num 134  
 Num-OIDs 191

Number 64, 66, 191  
     OIDs 191  
     TCP 64  
 Number:TCP 66  
 Numeric Argument Format 134  
 Numeric Comparisons 72

## O

Object ID 65  
 Octal Number 136  
 OctetString 193  
 OID 64, 190-191  
     number 191  
 OID Error 190  
 OID:defines 64  
 OID:request 65  
 OK 134, 144  
 OK Response 144  
 Old\_protocol 21  
 Old\_script 21

## P

P/Plain Text 44  
 Pagetitle 221, 230  
 Param 139  
 Parameter Section Example 25  
 PASSWORD 24  
 Password Fields 24  
 PATH 151  
 Pdtype 73



- Per-minute 65
- PER-SECOND 65
- Perl 94, 147
- Perl-like 94
- PerMapHTML 237
- PI 95
  - value 94
- Plugin 166
- Png 230
- PORT 152, 166
- Port\_number 19, 144
- Precedence Table 94
- Preventing 226
  - Quoted String From Becoming a Link 226
- Previousoutages 228
- Priv 191
- Privpassword 191
- Probe Calculations 94
- Probe Comments 45
- Probe Configuration window 22-23
- probe data, recording 27
- Probe File Description 22
- Probe Files 22
- Probe Parameters 23
- Probe Properties 73
- Probe Type 15, 18, 166
- Probe Variables 64
- Probe\_name 19, 144

- Probes 22, 147
- Probes/plugins 148, 166
- Program Control 134
  - Using Labels 134
- Program/script 147
  - name 147
- Python 149

## Q

- Query 73
  - APC-UPS MIB 73
  - sysUptime MIB-2 73
- Quoted Links 219, 226

## R

- Relational Tests 95
- Relative Offsets 137
  - Transfer Control 138
- Reload 17, 218
  - Changed Web Page Files 218
- Request 65
- Request:OID 64
- Response Times 142
  - Measuring 142
- RFC 23
- Runnable 149
- Runnable/executable 166

## S

- Sample <snmp-device-threshold 62

|  |                                    |
|--|------------------------------------|
| Sample <snmp-device-variables 62           | Snmp-device-variables 15-16, 63-64 |
| Sample Header Section 20                   | SNMP FAQ 189                       |
| Sample MIMETypes file 239                  | SNMP Get-Next-Request 65           |
| Script Command Format 134                  | SNMP Get-Request 73                |
| Script Failures 134                        | SNMP MIB 94                        |
| Handling 134                               | SNMP OID 64                        |
| Script Process Flow 134                    | SNMP Response 189                  |
| Script Termination 138                     | SNMPv1 191                         |
| Seconds 24                                 | SNMPV2C 21, 191                    |
| Secs 96, 145                               | SNMPv3 191                         |
| Select Misc 191                            | Snmpwalk 191                       |
| Select Probe window 20                     | Invoking 192                       |
| SEND 21, 144                               | Snmpwalk stopall 193               |
| ICMP 21                                    | SNMPwalks 193                      |
| Sending 13, 191                            | Special Character Example 136      |
| SNMP 13                                    | Special Characters 135             |
| Server Command 192                         | Sprintf 96                         |
| Server Settings 218                        | Sqrt 95                            |
| Servers 191, 218                           | Start 191                          |
| Set 144                                    | SNMP 191                           |
| DOWN 144                                   | StartOID 191                       |
| SetNameFieldWidth 230                      | Statshtml 229                      |
| Simple snmpwalk facility 191               | Status 41                          |
| SNMP 13, 18, 41, 62, 64, 73, 126, 189, 191 | Status Window Text 64              |
| Snmp-device-display 16, 41, 63, 81, 142    | Status windows 41, 64              |
| Snmp-device-properties 16, 63, 73          | Customizing 41                     |
| Snmp-device-threshold 16, 63               | Str 94                             |
| Snmp-device-thresholds 16, 63              | Strftime 96                        |

- String 62, 94, 134
    - Argument Format 134
    - Matches 72
    - Matching 95, 134
  - Strlen 96
  - Strptime 96
  - STRT 142
  - Stylings 43
  - Sub-expressions 94
  - Substr 96
  - Substring 101, 144
    - Extract 101
  - SysContact 72
  - SysDescr 64, 66
  - SysUptime MIB-2 73
- T**
- Target File Example 219
  - Target File Is Read 220
  - TCP 13, 18, 41, 64, 66, 134, 144
    - Number 64
  - TCP-based 41
  - Tcp-script 18, 144
  - TCP Script Commands 142
  - TCP Timers 142
  - Tcp.custom 19, 22, 144
  - TCP:Number 66
  - TcpCurrEstab 41, 64, 66, 81
  - Telnet 194, 230
  - Telnet window 194
  - Telnetserverurl 230
  - Template File Example 221
  - Template Files 221
  - Template.html 222, 238
  - Test Page 222, 226
  - Thresholds 62
  - Time Measurement Probe Variables 142
  - TIME varname 142
  - Timeout 145, 191
  - TimeTicks 193
  - Title 221
  - Tools 166
  - Total-value 65
  - Transfer Control 138
    - Using Relative Offsets 138
  - Traps
    - TrapVariable 113
  - TrapVariable 113
  - TRUE
    - value 94
  - Trunc 95
  - Type 64
- U**
- UDP 13
  - Unary 94
  - Unexpected 139
  - Unix 147

Unix Linefeed 135

Unless-e 191

UP 134

UPPER CASE 135

Uptime 229

URL 219, 227, 237, 240

Url-to-invoke 20

Url\_hint 20

Used 237

Username 191

## V

VALUE 94

Values 25, 64, 94

FALSE 94

PI 95

TRUE 94

x1 95

Var 64, 94

Variable-name OID 62, 64

VariableName 65

Variables 134

Varname 142

Vertical Tab 135

## W

WAIT 137, 144

WAIT timeout

Wait/p 144

i/Seconds 144

WARN 134

WARN Response 138, 144

Web Page Files 238

Web Pages 218, 237, 239

Customizing 218

level 237, 239

Web Server 218

Webpageurl 233

Webserverurl 230

Whitespace 239

Wild-card Character Matching 136

Window\_name 225

Windows 147

probes 147