



InterMapper

Network Monitoring & Alerting Software

Developer Guide

Version 4.5

October 2006

Table Of Contents

InterMapper Developer Guide: An Introduction	1
Licensing Information.....	3
Software License Agreement	3
License.....	3
Disclaimer of Warranty.....	3
US Government.....	3
Acknowledgement	3
Third-party Licenses	4
Apache Group.....	4
JSSE License	5
Creating Custom Probes.....	7
Custom Probes.....	7
Kinds of Probes	7
Installing Custom Probes.....	8
Finding Existing Custom Probes	8
Installing a Custom Probe and Reloading Probes	8
Creating Your Own Custom Probes	8
Probe Files and File Names	9
Custom Probe File Format.....	10
Probe File Header	11
Header Part Format	11
Header Parts	11
Sample Header Section	12
Header Section of Custom SNMP Probes.....	12
Probe File Description.....	13
The Markup Commands.....	13
Probe Parameters	15
Password Fields	15
Parameter Section Example	15
Probe Properties.....	16
Probe Variables	17
Probe Calculations	20
Reserved keywords	20
Precedence Table (Least to Most)	20
Built-in Numeric Functions	20
Built-in String Functions	21
Function Descriptions	22
Probe Comments	24
Comments	24
Probe Script Reference	25
Probe Script Reference.....	25
Probe Command Reference	31
Customizing Status windows	41
Custom SNMP Probes.....	41
Custom TCP-Based Probes.....	41
TCP Probes	42
Example TCP Probe File	42
Errors with Custom Probes	46
Error: A device shows a "Reason: No SNMP Response." at the bottom of the status window	46
Error: When I build a custom probe, the status window shows "[N/A]" for certain values	46

Error: When I build a custom probe, the status window shows "[noSuchName]" for certain values.....	46
Measuring Response Times.....	47
Time Measurement Probe Variables.....	47
TCP Script Commands.....	47
The <script-output> Section.....	47
A Note on Accuracy.....	47
Custom SNMP Probes.....	48
SNMP Probe Variables - The <snmp-device-variables> Section.....	49
Thresholds - The <snmp-device-thresholds> Section.....	51
Creating Comparisons.....	52
Status Window Text - The <snmp-device-display> Section.....	52
SNMP Query Settings - The <snmp-device-properties> Section.....	53
The Dartware MIB.....	54
Command Line Probes.....	55
Installing a Command-line Probe.....	55
Command-line Probe File Format.....	55
Example Notification from a Command Line Program.....	58
Implementing an iPing Notifier.....	58
Nagios Plugins.....	59
Big Brother Probes.....	60
Using the Command Line Interface.....	61
Examples for Import and Export commands.....	62
Customizing Web Pages.....	63
Reloading Changed Web Page Files.....	63
Target Files.....	64
Target File Example:.....	64
Quoted Links.....	64
What Happens When a Target File Is Read?.....	64
Built-in Target Files.....	64
Template Files.....	65
Template File Example.....	65
Directives.....	66
Summary of Directives.....	66
Quoted Links.....	68
Preventing a Quoted String From Becoming a Link.....	68
Macro Reference.....	69
The Include Macro.....	69
Macros that generate the "content" of an InterMapper web page.....	69
Miscellaneous macros that describe InterMapper and its environment.....	70
Macros to place images onto a page.....	70
Macros that control the interval between page refreshes.....	70
Macros that describe the requested URL.....	71
Folder Structure.....	72
How the Web Page Files are Used.....	72
MIME Types.....	73
Tip for Calling Charts.....	74
Glossary.....	75

InterMapper Developer Guide: An Introduction

InterMapper is a network monitoring and alerting program. It continually tests routers, servers, hubs, and other computer devices that are attached to your network. If InterMapper detects a failure, it sends notifications to one or more individuals via sounds, e-mail, pagers, or by running a program to correct the problem.

Use this manual to learn about how to develop custom probes and web pages for InterMapper.

Customizing InterMapper's Probes

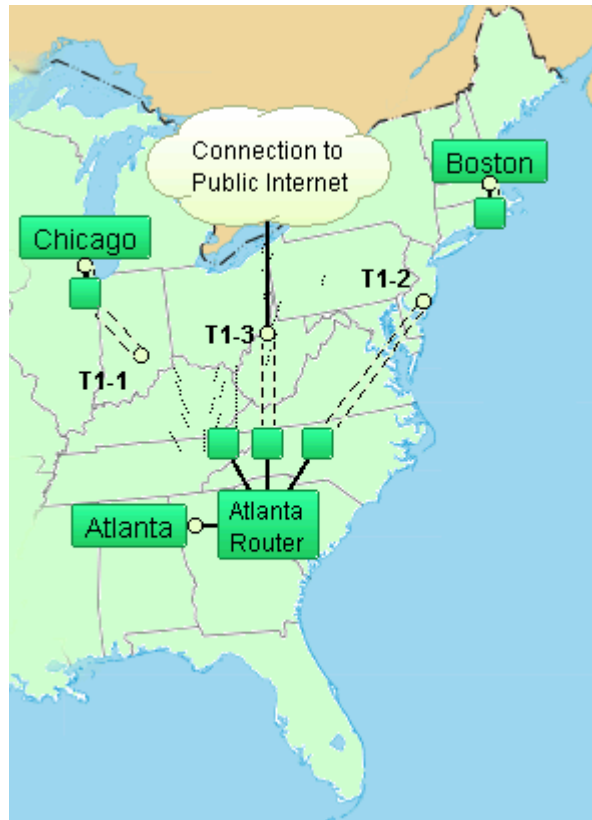
Explains how to create your own custom probes, and how to configure them to add power and flexibility to your network monitoring.

Customizing Web Pages

Tells how you can change the look and function of the web pages available from InterMapper's built-in Web server.

Advanced Data Importing

Tells how you can import data into maps using advanced import features, such as changing the locations of devices on a map, and changing device settings.



Please give us comments at the address listed below. Thanks!

Dartware, LLC
InterMapper Feedback

Licensing Information

Software License Agreement

This is a legal agreement between you and Dartware, LLC covering your use of InterMapper®, InterMapper Remote™, and other Dartware products and the associated documentation (the "Software"). Be sure to read the following agreement before using the Software. BY USING THE SOFTWARE (REGARDLESS IF YOU HAVE REGISTERED THE SOFTWARE OR NOT), YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT USE THE SOFTWARE AND DESTROY ALL COPIES IN YOUR POSSESSION.

The Software is owned by Dartware, LLC and is protected by United States copyright laws and international treaty provisions. It is licensed to you. Therefore, you must treat the Software like any other copyrighted material (e.g., a book or musical recording).

License

This license allows you the right to use one copy of the Software on a single computer. You may make one (1) copy of the software as a backup. You may not network the Software or otherwise use it or make it available for use on more than one computer at the same time. You may not rent or lease the Software, nor may you modify, adapt, translate, decompile, or disassemble the Software. If you violate any part of this agreement, your right to use this Software terminates automatically and you must then destroy all copies of the Software in your possession.

Disclaimer of Warranty

The Software and its related documentation are provided "AS IS" and without warranty of any kind. The person using the software bears all risk as to the quality and performance of the software. If you paid for the product, and within 30 days find that it doesn't do what you want, then you can notify Dartware, LLC and your money will be refunded and your license canceled. Dartware, LLC hereby disclaims all warranties relating to this software, whether express or implied, including without limitation any implied warranties of merchantability or fitness for a particular purpose. Dartware, LLC will not be liable for any special, incidental, consequential, indirect or similar damages due to loss of data or any other reason, even if Dartware, LLC or their agent has been advised of the possibility of such damages. In no event shall Dartware, LLC be liable for any damages, regardless of the form of the claim.

US Government

Government End Users: If you are acquiring the Software on behalf of any unit or agency of the United States Government, the following provisions apply. The Government agrees: (i) if the Software is supplied to the Department of Defense (DoD), the Software is classified as "Commercial Computer Software" and the Government is acquiring only "restricted rights" in the Software and its documentation as that term is defined in Clause 252.227-7013(c)(1) of the DFARS; and (ii) if the Software are supplied to any unit or agency of the United States Government other than DoD, the Government's rights in the Software and its documentation will be as defined in Clause 52.227-19(c)(2) of the FAR or, in the case of NASA, in Clause 18-52.227-86(d) of the NASA Supplement to the FAR. The manufacturer is Dartware, LLC, 10 Buck Road, PO Box 130, Hanover, NH 03755-0130 USA.

This Agreement shall be governed by the laws of the State of New Hampshire. If for any reason a court of competent jurisdiction finds any provision of the Agreement, or portion thereof, to be unenforceable, that provision of the Agreement shall be enforced to the maximum extent permissible so as to effect the intent of the parties, and the remainder of this Agreement shall continue in full force and effect.

Acknowledgement

You acknowledge that you have read this agreement, understand it, and agree to be bound by its terms and conditions. You further agree that it is the complete and exclusive statement of the agreement between you and Dartware, LLC which supercedes all proposals or prior agreements, oral or written, and all other communications between you and Dartware, LLC relating to the subject matter of this agreement.

Dartware, LLC
10 Buck Road, PO Box 130
Hanover, NH 03755-0130 USA
Voice: 603-643-2268; Fax: 603-643-2289
Web: <http://www.dartware.com>

Third-party Licenses

Portions of this Software from Dartware, LLC may use the following copyrighted material, the use of which is hereby gratefully acknowledged.

Apache Group

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>). It is used under the following license:

```
/*
 * =====
 *                               The Apache Software License, Version 1.1
 * =====
 *
 * Copyright (C) 1999 The Apache Software Foundation. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modifica-
 * tion, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. The end-user documentation included with the redistribution, if any, must
 *    include the following acknowledgment: "This product includes software
 *    developed by the Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself, if
 *    and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "log4j" and "Apache Software Foundation" must not be used to
 *    endorse or promote products derived from this software without prior
 *    written permission. For written permission, please contact
 *    apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache", nor may
 *    "Apache" appear in their name, without prior written permission of the
 *    Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
 * FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLU-
 * DING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * This software consists of voluntary contributions made by many individuals
 * on behalf of the Apache Software Foundation. For more information on the
 * Apache Software Foundation, please see .
 *
 */
```

JSSE License

Sun Microsystems, Inc. Binary Code License Agreement

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE OPENING THE SOFTWARE MEDIA PACKAGE. BY OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCESSING THE SOFTWARE ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR, IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

- 1. LICENSE TO USE.** Sun grants you a non-exclusive and non-transferable license for the internal use only of the accompanying software and documentation and any error corrections provided by Sun (collectively "Software"), by the number of users and the class of computer hardware for which the corresponding fee has been paid.
- 2. RESTRICTIONS.** Software is confidential and copyrighted. Title to Software and all associated intellectual property rights is retained by Sun and/or its licensors. Except as specifically authorized in any Supplemental License Terms, you may not make copies of Software, other than a single copy of Software for archival purposes. Unless enforcement is prohibited by applicable law, you may not modify, decompile, or reverse engineer Software. You acknowledge that Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this Agreement.
- 3. LIMITED WARRANTY.** Sun warrants to you that for a period of ninety (90) days from the date of purchase, as evidenced by a copy of the receipt, the media on which Software is furnished (if any) will be free of defects in materials and workmanship under normal use. Except for the foregoing, Software is provided "AS IS". Your exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace Software media or refund the fee paid for Software.
- 4. DISCLAIMER OF WARRANTY.** UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.
- 5. LIMITATION OF LIABILITY.** TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In no event will Sun's liability to you, whether in contract, tort (including negligence), or otherwise, exceed the amount paid by you for Software under this Agreement. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.
- 6. Termination.** This Agreement is effective until terminated. You may terminate this Agreement at any time by destroying all copies of Software. This Agreement will terminate immediately without notice from Sun if you fail to comply with any provision of this Agreement. Upon Termination, you must destroy all copies of Software.
- 7. Export Regulations.** All Software and technical data delivered under this Agreement are subject to US export control laws and may be subject to export or import regulations in other countries. You agree to comply strictly with all such laws and regulations and acknowledge that you have the responsibility to obtain such licenses to export, re-export, or import as may be required after delivery to you.
- 8. U.S. Government Restricted Rights.** If Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in Software and accompanying documentation will be only as set forth in this Agreement; this is in accordance with 48 CFR 227.7201 through 227.7202-4 (for Department of Defense (DOD) acquisitions) and with 48 CFR 2.101 and 12.212 (for non-DOD acquisitions).
- 9. Governing Law.** Any action related to this Agreement will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.
- 10. Severability.** If any provision of this Agreement is held to be unenforceable, this Agreement will remain in effect with the provision omitted, unless omission would frustrate the intent of the parties, in which case this Agreement will immediately terminate.

11. Integration. This Agreement is the entire agreement between you and Sun relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification of this Agreement will be binding, unless in writing and signed by an authorized representative of each party.

**JAVA OPTIONAL PACKAGE
JAVA™ SECURE SOCKET EXTENSION, VERSION 1.0.3_XX
SUPPLEMENTAL LICENSE TERMS**

These supplemental license terms ("Supplemental Terms") add to or modify the terms of the Binary Code License Agreement (collectively, the "Agreement"). Capitalized terms not defined in these Supplemental Terms shall have the same meanings ascribed to them in the Agreement. These Supplemental Terms shall supersede any inconsistent or conflicting terms in the Agreement, or in any license contained within the Software.

1. Software Internal Use and Development License Grant. Subject to the terms and conditions of this Agreement, including, but not limited to Section 3 (Java(TM) Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce internally and use internally the binary form of the Software, complete and unmodified, for the sole purpose of designing, developing and testing your Java applets and applications ("Programs").

2. License to Distribute Software. In addition to the license granted in Section 1 (Software Internal Use and Development License Grant) of these Supplemental Terms, subject to the terms and conditions of this Agreement, including but not limited to, Section 3 (Java Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce and distribute the Software in binary code form only, provided that you (i) distribute the Software complete and unmodified and only bundled as part of your Programs, (ii) do not distribute additional software intended to replace any component(s) of the Software, (iii) do not remove or alter any proprietary legends or notices contained in the Software, (iv) only distribute the Software subject to a license agreement that protects Sun's interests consistent with the terms contained in this Agreement, (v) agree to defend and indemnify Sun and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection with any claim, lawsuit or action by any third party that arises or results from the use or distribution of any and all Programs and/or Software, and (vi) include the following statement as part of product documentation (whether hard copy or electronic), as a part of a copyright page or proprietary rights notice page, in an "About" box or in any other form reasonably designed to make the statement visible to users of the Software: "This product includes code licensed from RSA Data Security".

3. Java Technology Restrictions. You may not modify the Java Platform Interface ("JPI", identified as classes contained within the "java" package or any subpackages of the "java" package), by creating additional classes within the JPI or otherwise causing the addition to or modification of the classes in the JPI. In the event that you create an additional class and associated API(s) which (i) extends the functionality of the Java platform, and (ii) is exposed to third party software developers for the purpose of developing additional software which invokes such additional API, you must promptly publish broadly an accurate specification for such API for free use by all developers. You may not create, or authorize your licensees to create additional classes, interfaces, or subpackages that are in any way identified as "java", "javax", "sun" or similar convention as specified by Sun in any naming convention designation.

4. Trademarks and Logos. You acknowledge and agree as between you and Sun that Sun owns the SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET trademarks and all SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET-related trademarks, service marks, logos and other brand designations ("Sun Marks"), and you agree to comply with the Sun Trademark and Logo Usage Requirements currently located at <http://www.sun.com/policies/trademarks>. Any use you make of the Sun Marks inures to Sun's benefit.

5. Source Code. Software may contain source code that is provided solely for reference purposes pursuant to the terms of this Agreement. Source code may not be redistributed unless expressly provided for in this Agreement.

6. Termination for Infringement. Either party may terminate this Agreement immediately should any Software become, or in either party's opinion be likely to become, the subject of a claim of infringement of any intellectual property right.

For inquiries please contact: Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303 (LFI#119058/Form ID#011801)

Creating Custom Probes

Custom 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. In that case, you can create your own probes. InterMapper's probes are defined by *probe files*, which are text files, and 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 **Probe Type:** popup menu along with the built-in ones.

All InterMapper's probes follow the same logic:

- The probe sends one or more queries, as SNMP requests, UDP or AppleTalk 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.

Kinds of Probes

InterMapper has several kinds of probes that you can use:

SNMP Probes

InterMapper tests the device's status by sending SNMP queries and comparing the results to user-specified thresholds.

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.

Command-line Probes

InterMapper will invoke a program (as if "from the command line"), and use the results of the program to determine the device's status.

Big Brother Probes

Big Brother(tm) 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 at support@dartware.com so that we can make it available to other customers. You can also check the list of probes that have already been contributed at <http://www.intermapper.com/contrib/>.

Note: Before modifications will become effective, you must choose **Reload probes...** from the **Probe Type:** popup menu in a device's **Get Info** window.

Installing Custom Probes

Use custom probes to enhance InterMapper's capabilities. These are probes created for special purposes or for certain devices.

Finding Existing Custom Probes

You can use any of the probes that Dartware has created, or use probes contributed by our other customers. These probes are available from:

<http://www.intermapper.com/contrib/customprobes.html>

Installing a Custom Probe and Reloading Probes

Before you can use a custom probe, you must do two things:

1. **Place the Probe file in the *Probes* folder** of the *InterMapper Settings* folder.
2. **Reload probes**, as described in Step 5 below, to activate a new or modified probe.

To install a custom probe:

1. Download the probe, and unstuff it with StuffIt Expander if necessary.
2. Drag the resulting file to the *InterMapper Settings:Probes* folder.
3. Open an InterMapper map.
4. Add a new device with the DNS name/IP address of the device you want to test.
5. Get to the Probe Types menu:
 - On MacOS X: Double-click the device. You'll see the Device Info window open, with a Probe Type popup at the lower left corner.
 - With InterMapper Remote or InterMapper Console: Right/Control-click on a device, and choose Set Info > Set Probe... You'll see a probe configuration window open, with a Probe Type drop down.
6. Choose **Reload Probes...** from the **Probe Type** menu.
7. Select the new probe type from the popup menu.
8. Configure the probe by filling in the fields as required.
9. When finished, click **OK**. InterMapper begins using the new probe to test the device.

Creating Your Own Custom Probes

You can also create your own custom probes. Creating your own probe is pretty simple to do, using the tool described in Building Custom Probes. If you create a probe you find useful, please contribute your new probe so others can use it as well. Send us an e-mail at support@dartware.com and we'll post it to the Contributions page mentioned above.

Probe Files and File Names

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

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. (There is a 31 character limit to Macintosh file names which must be observed.)

- **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. Dartware 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"
```

Custom Probe File Format

Probe files have several sections, described below. Each section contains a number of lines bracketed by a `<section-name> ... </section-name>` lines. You might want to open a separate window with the example probe file while reading the subsequent sections.

Probe Files

Learn where probe files are saved on the hard drive, and how the files are named.

Header Information

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

Text Description

Learn about the **description** section of the probe definition, which specifies the help text that appears in the user's probe configuration window. The text typically explains the function of the probe, and describes any parameters. You can also learn about the markup language you can use to apply a certain amount of formatting and text styling to the probe's help text.

Parameters to the Probe File

Learn about the **parameters** for the probe. InterMapper automatically creates fields for these entries when it opens the probe configuration window.

Scripting Language

Learn about the probe **scripting language**, a sequence of statements that specify the data to send out a TCP connection, the expected response, and how to interpret that data to set the display status for the device being probed.

Comments

Learn about the format to use when adding **comments**. Comment format is similar to HTML comments: full details on comment are included in this section.

Custom SNMP Probes

Learn about how to create **custom SNMP Probes**. With custom SNMP probes, you can query specific MIB variables besides those built into InterMapper. You can also specify your own thresholds for MIB variables to control warnings and alarms.

Note: Before a modification to a probe becomes effective, you need to choose **Reload probes...** from the Alert by pop-up menu in a device's Get Info window.

Probe File Header

The *header* of a probe file contains a formal description of the probe. It is defined using the following tags:

```
<header> ... </header>
```

Header Part Format

The header is composed of several parts. Each part has a name, and a corresponding value, written in this format:

```
part-name = "value"
```

where *value* is enclosed in double-quotes.

Header Parts

type	<p>Describes the type of the probe file. InterMapper supports the following probe types:</p> <ul style="list-style-type: none"> • builtin • tcp-script • custom-snmp • command-line <p>For <i>custom TCP probes</i> use the tcp-script type. For <i>custom SNMP probes</i> use the custom-snmp type. For <i>Command-line probes</i> use the command-line type.</p>
package	<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 Dartware, 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>Note: 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.</p>
probe_name	<p>The second part of the probe's full identifier. The probe_name may be whatever string the creating organization chooses.</p>
human_name	<p>The string that appears in the Probe Type pop-up menu. This string helps guides the user to select the probe for a particular device.</p>
version	<p>Provides a means to determine which probe file is the current one. The format of the version is "#.#".</p>
address_type	<p>A comma-separated list of one or more address types. InterMapper implements "IP" and "AT".</p>
port_number	<p>The IP port used by this probe.</p>

old_protocol
old_script Both used for backward compatibility with existing probes in InterMapper. Newly-created probes do not need the "old_script" part.

category Specify the category and subcategory in which this probe appears in the Select Probe window. To do this, add the following line to the <header> section of the probe:

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

Example:

```
category = "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}"
```

Sample Header Section

This is a sample header from the Custom TCP script.

```
<header>
type = "tcp-script"
package = "com.dartware"
probe_name = "tcp.custom"
human_name = "Custom TCP"
version = "0.1"
address_type = "IP"
port_number = "23"
old_protocol = "8" # Backward compat. with
old numbering scheme.
old_script = "8001"
</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:

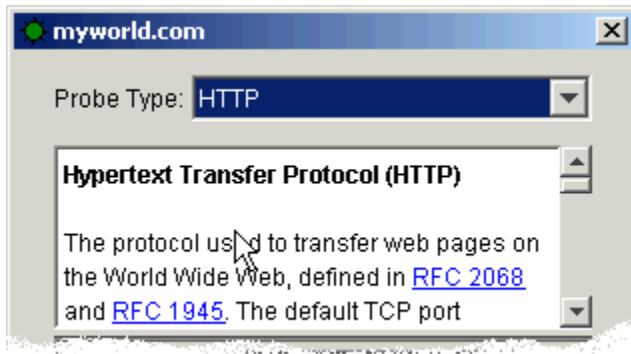
- The "type" for a Custom SNMP probe is "custom-snmp".
- There is a `FLAGS=xxx,xxx` command that takes the following optional items as parameters:
 - `NOLINKS` - InterMapper will not poll links (interfaces) with SNMP
 - `SNMPV2C` - InterMapper will use SNMPv2c to poll the device
 - `NOICMPFALLBACK` - InterMapper will not send an ICMP ping to a device if no SNMP responses return.

Probe File Description

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

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

A sample is shown below. Note that the description can be marked up with bold, italic, plain text. The underlined text becomes a link to an external URL, as described below.



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

The description section is delimited by `<description>` and `</description>` tags. The entire text between those tags is placed into the probe configuration window.

The Markup Commands

You can apply text styles to the description text using markup 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 `>` or `<` and `>`. 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

The markup tags are:

Tag	Action
M	Set the font to Monaco (a monospaced font).
G	Set the font to Geneva (a proportional spaced font).
+	Increase the font size by one. Multiples (" <code>++</code> ") increase the font size by the corresponding amount.
-	Decrease the font size by one. Multiples are allowed.
B	Set following text in bold.
I	Set following text in italic.
P	Set following text to plain. This undoes all other stylings

U Set following text to underlined. See **Creating a link** section below for making hyperlinks.

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

Bold text *Bold Italic* text Plain Text

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

```

Big red monospace

```
\2U\http://www.example.com\P0\  

```

<http://www.example.com>

Creating a link

The last example above shows the script code to create a link.

- If the text between an opening `\U\` and a closing `\P\` tag is a URL, it is treated by InterMapper as a link. When the user clicks the link, InterMapper invokes the web browser to retrieve the URL.
- In the previous example, "`\2U\`" means "set the color to blue, underline the text."
Note: As a special case, if the only text between the opening and closing tags is a URL (e.g., `http://www.example.com`), InterMapper will treat it as a link and will invoke the user's web browser to retrieve that URL when it's clicked.
- In the previous example, "`\P0\`" means "set the text style back to plain, set the color to black."

Probe Parameters

A probe can have one or more parameters. These parameters are set by the user in the Probe Configuration 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:

```
"parameter name" = "parameter value"
```

Each parameter name/value appears in its own entry field in the Probe Configuration 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}").

Password Fields

InterMapper also allows you to create input parameters that conceal the string from casual view (so-called *password parameters*.) The data will be 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:

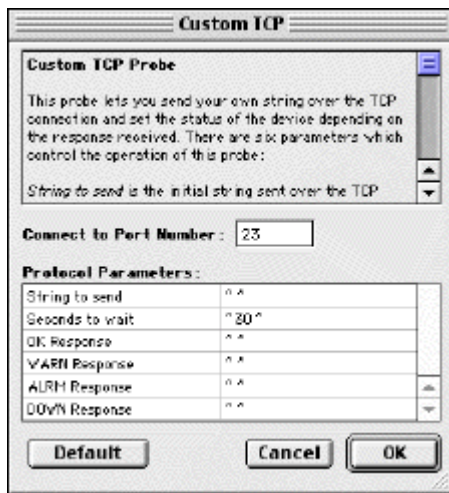
```
"Password*" = ""
```

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 configuration window.

Parameter Section Example

This is the parameter section from the Custom TCP script. Note that the list of parameters at the right matches the values shown below.

The "Seconds to wait" parameter contains a default value of 30.



Probe Parameters.

```
<parameters>
"String to send" =
" "
"Seconds to wait" =
" 30 "
"OK Response" = " "
"WARN Response" =
" "
"ALRM Response" =
" "
"DOWN Response" =
" "
</parameters>
```

Probe Properties

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"

</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.

Probe Variables

InterMapper can retrieve MIB variables from a device and then test 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: the variable's name, its OID, its Type, and an optional Chart legend. Their definitions are:

- **VariableName** is the name used to represent the particular MIB value in this probe. A VariableName consists of letters, digits and an underscore, and must begin with a letter. VariableNames are not case-sensitive. These variable names will be represented in the probe as `$(VariableName)`. It is not necessary that a particular VariableName match the corresponding name in the MIB, although that is definitely convenient.

Note: Versions of InterMapper prior to 4.2 required that variable names be enclosed in curly braces, e.g., `$(VariableName)`. This not necessary in version 4.2 and later, and both forms are now identical.

- **OID** is the Object ID for the particular probe variable. Note that a scalar's OID must end in ".0" according to the SNMP specifications. Although it is common to off leave the suffix for scalar values, it is technically correct to include it and InterMapper requires the suffix to construct the proper queries. InterMapper will request the OID and wait for the response. [Technically, InterMapper issues an SNMP Get-Next-Request for the previous OID.] Currently, there is no facility for iterating across all rows of a particular table.

Note: Calculation variables have a slightly different form, as described below.

- **Type** may be one of: Default, Per-second, Per-minute, Total-value, or Hexadecimal.
 - **Default** should be used for variables of type Integer and DisplayString. InterMapper will automatically determine the type and display the value properly.
 - **Per-second** and **Per-minute** will force InterMapper to compute a rate for the particular variable by computing the difference between two successive samples and dividing by the elapsed time.
 - **Total-value** displays the actual value of a counter or gauge, not a computed rate value.
 - **Hexadecimal** displays the data as hex digits.
 - **Calculation** sets the variable to the result of the calculation shown in the OID field.
 - **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)`.

Here is a 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

-- Non-pollled values:
-- Calculation variables are computed each poll time
-- TrapVariables are updated when a trap arrives

    SineValue, (10*sin(0.01*time())), CALCULATION, "10 *
sin(0.01 * time())"

    InterMapperTimeStamp, 1.3.6.1.4.1.6306.2.1.1.0,
TRAPVARIABLE, "Timestamp"

</snmp-device-variables>
```

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

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:

```
    SineValue, (10*sin(0.01*time())), CALCULATION, "10 *
sin(0.01 * time())"
```

The variable "SineValue" will be 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.

Macros

InterMapper supports several macros that show information about a variable:

- **`${variablename}`** or **`$variablename`** In a `<snmp-device-display>` section of a probe file, an occurrence of a variable name (with or without the curly braces `{...}`) will be 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 will result in the value of "3"; if the variable had the value 4.75 it would be displayed as "5". This value will be chartable, that is, clicking it will make a new strip chart, or dragging it will add it to an existing chart. The *ipForwDatagrams* variable defined above could be referred to in either of these forms:

```
${ipForwDatagrams} or
$ipForwDatagrams
```

- **`#{chartable: xxx : yyy}`** In a `<snmp-device-display>` section of a probe file, the `#{chartable: ...}` macro controls the number of decimal places. There are two parameters: a *formatting string* that indicates the number and placement of the digits near the decimal point, and the variable to be formatted. For example, if `$var` is 3.14159265:

```
#{chartable: #.## : $var } --> 3.14
#{chartable: #.##### : $var } -->
3.1415927
```

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

```
#{ipForwDatagrams:legend}
```

- would result in the string "Forwarded datagrams".

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.

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.
- 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.

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' to nearest integer; e.g. round(3.987) = 4.
- round(x, y) - Round 'x' to 'y' places after the decimal point; e.g. round(3.14159, 3) = 3.142.
- 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

- `strlen(str)`
- `sprintf(fmt, ...)`
- `strftime(fmt)`
- `strftime(fmt, secs)`
- `strptime(str, fmt)`
- `substr(str, offset, len)`
- `unpack(binary str, fmt)`

Function Descriptions

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 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, we will use the target platform's byte order (little endian on Windows, big endian on Mac).
- If format specifier is not supplied, the function will return `str`.

Format specifier	Description
<code>c</code>	signed character value (1 byte)
<code>C</code>	unsigned character value (1 byte)
<code>l</code>	signed long value (4 bytes)
<code>L</code>	unsigned long value (4 bytes)
<code>s</code>	signed short value (2 bytes)
<code>S</code>	unsigned short value (2 bytes)
<code>#B</code>	a base64 string (all bytes) (non-standard; Bill made up this format specifier)
<code>></code>	big-endian modifier
<code><</code>	little-endian modifier

Examples:

```
unpack( "\1", "c" )      --> 1
unpack( "\1\2\3\4", ">L" ) --> 16909060
unpack( "\1\2\3\4", "<L" ) --> 67305985
unpack( "BS64", "#B" )  --> "... " (where "... " is "\x5\x2e\xb8")
unpack( "\1\2\3", ">L" ) --> 16909056
```

Note: This is a simplified version of `unpack()`, which only supports one format code; the expression language doesn't support multiple return values.

Probe Comments

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.

Probe Script Reference

Probe Script Reference

Use the InterMapper probe files 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 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*, *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:

- `\r` Carriage Return
- `\n` Unix Linefeed
- `\t` Horizontal Tab
- `\f` Formfeed
- `\b` Backspace
- `\v` Vertical Tab
- `\a` Alert (bell) Character
- `\"` Double Quote
- `\\` Backslash
- `\ooo` Octal Number
- `\xhh` Hexadecimal Number

Special Character Example:

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

String Matching

Some commands specify a string to match.

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 define an expression containing special characters which match any character. When an 'r' follows the final quote, a '.' in the expression matches any single character.

Example:

".red"r matches "fred", "Fred", "tred", "bred", etc. It does not match "freD".

Combined Matching

You can combine the 'r' and the 'i' to indicate both expression matching and case-insensitivity.

Example:

".red"ri matches "fred", "freD", "rEd", etc.

Note: Previous versions of InterMapper allowed a perl-like syntax of m"xxx" to search for a pattern that might include a ".". This continues to work in version 3.6, but is deprecated, and no longer recommended for new scripts.

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 are defined in the `<parameter>` section of the probe file.
- Variable names appear are preceded by a dollar sign (\$), and are enclosed by curly braces.
- Variable names are case-insensitive.

Example:

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

Built-in Variables

InterMapper has a number of built-in variables.

- To prevent confusion with user-defined variables, all built-in variables are prefixed with a single underscore.

<code>\${_REMOTEADDRESS}</code>	The IP address of the other side of the connection.
<code>\${_REMOTEPORT}</code>	The TCP port of the other side of the connection.
<code>\${_LOCALADDRESS}</code>	The IP address of this side of the connection.
<code>\${_LOCALPORT}</code>	The TCP port of this side of the connection (usually an ephemeral port).
<code>\${_GMTTIME}</code>	The current GMT time in RFC 822 format.
<code>\${_VERSION}</code>	The current version of InterMapper (e.g. "3.5b4").
<code>\${_IDLETIMEOUT}</code>	The number of seconds specified as an IDLE timeout.
<code>\${_STRINGTOMATCH}</code>	The last argument to a MTCH or EXPT command (to be used for error messages.)
<code>\${_IDLELINE}</code>	The line number in the TCP script where we were IDLE.
<code>\${_SECSCONNECTED}</code>	The number of seconds the TCP probe has been connected.

Built-in Macros

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

<code>\${_LINE:<line>}</code>	The first <code><num></code> characters of the last line received.
<code>\${_BASE64:<param>}</code>	The Base-64 encoding of the string that follows the ":".
<code>\${_CVSPASSWORD:<param>}</code>	The IP address of the other side of the connection.

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.

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.

Example:

```
<-- The following section contains handlers for unexpected responses
to commands -->
```

Probe Command Reference

This is a list of commands defined in the InterMapper Scripting Language. For an example of a custom probe script, see the Annotated Example of the FTP (Login) Script.

Device I/O Commands

The following commands operate on a device by sending data to the device or by reading one or more lines from the input (from the connection to the device being tested). Each command that reads a device compares its string to the *current line* - which is the most recently-read line from the connection. If there is no current line (for example, if a SEND command has been executed), these statements will read one or more lines to get the current line.

- EXPT "string" #fail - Searches incoming lines for the specified *string*.
- MTCH "string" #fail - Searches the next incoming line for the specified string.
- SKIP "string" #fail - Ignore all incoming lines containing the specified *string*.
- DISC #discfail - Jump to a specified line number if the probe is suddenly disconnected.
- CONN #timeout ["TELNET"] - Specifies the connect timeout of the probe and whether to process Telnet options.
- PORT #port_num #connect_timeout - No longer required (the remote port number is now a separate parameter in the configuration dialog.)
- LINE [ON | OFF] - Specifies whether the script should read incoming data as lines or as raw data.
- NEXT - Clears the input buffer so that subsequent **MTCH** commands will operate on newly-received information.
- SEND "string" - Sends the specified string to a remote device.

Commands that control script flow

The following commands control the order of operations in the script.

- CHCK "string" #fail - Determines whether "string" is non-empty.
- DONE *status* ["message"] - Terminates a script with a specified condition.
- EXIT - Terminates a script with the condition specified previously by **STAT**.
- GOTO #statement - Branches immediately to the specified statement number.
- NBGT #arg1 #arg2 #line - (**N**umeric **B**ranch **G**reater **T**han) branches to #line if #arg1 is greater than #arg2.
- NBNE #arg1 #arg2 #line - (**N**umeric **B**ranch **N**ot **E**qual) Compares two numeric arguments and branches to the indicated #line if they are not equal.
- SBNE "arg1" "arg2" #line - (**S**tring **B**ranch **N**ot **E**qual) Compares two string arguments and branches to the indicated #line if they are not equal.
- STAT *status* ["message"] - Specifies the status condition of a script when it ends.
- WAIT #secs #idlefail #discfail - Specifies the number of seconds the probe waits for a response.

String processing commands

The following commands process and manipulate strings.

- STOR "variable" "string" - Stores the string into the variable named "variable"
- SCAT "variable1" "variable2" #fail - Concatenates variable1 and variable2, placing the resulting string in variable1.
- NADD "variable" #number - (**N**umeric **A**dd) Adds a numeric value to a variable.

Commands that measure time

- **STRT** - Starts a millisecond timer that InterMapper can use to determine the elapsed time for some event.
- **TIME "variable"** - Sets the named variable to the current number of milliseconds from the most recent **STRT** command.
- **WAIT #secs #idlefail #discfail** - Specifies the number of seconds the probe waits for a response.

Command List - Alphabetical

The following is a list of available InterMapper probe scripting language. The commands are listed alphabetically.

Probe Command Details - Alphabetical

CHCK "string" #fail

Use the **CHCK** command to determine whether "string" is non-empty. If the string is empty, the script jumps to the specified #fail line.

This command can be used to construct scripts whose control changes depending on whether an optional parameter is supplied.

Possible failures: None

CONN #timeout ["TELNET"]

Use the **CONN** command to specify the connect timeout of the probe and whether to process Telnet options.

If you are going to use the **CONN** command, it *must be the first statement of the script*. When the script executes, the parameters of the **CONN** statement determine the options InterMapper uses to connect to the remote computer.

Parameter 1 - #timeout: The number of seconds to wait while trying to connect before giving up.

Parameter 2: If the second parameter of the **CONN** command is "TELNET" (including the quotes), then the connection is created in a mode where the TCP stream automatically processes and negatively acknowledge any incoming Telnet options. This allows a Telnet probe to ignore the telnet options and work in simple line-by-line mode for the remainder of the script.

Possible Failures: None

DISC #discfail

Use the **DISC** command to cause the script to jump to a specified line number if the probe is suddenly disconnected. You can use this command to identify scripts that fail because of a TCP disconnection.

The script's disconnect line can also be set using the third parameter to the WAIT command. [does this set the same parameter that WAIT sets?]

DONE status ["message"]

Use the **DONE** command to terminate the script with the specified condition, which must be one of

[OKAY | WARN | ALRM | DOWN]

The optional "message" parameter lets you provide more detail about the condition. The status values for the **DONE** command must be in *UPPERCASE*.

Example:

```
DONE ALRM "[HTTP] 500 Response received."
```

This example sets the status of the device to *ALRM*. The condition of the device (which is displayed in the pop-up window and the Device List window) is set to "[HTTP] 500 Response received." to give the user an indication of the reason for the alarm.

Possible Failures: None.

Tip: If the final statement of your script is not a **DONE** command, the script automatically terminates with a "DONE OKAY" status.

EXIT

Use the **EXIT** command to terminate the script, setting the status and condition string to whatever is specified by a previous **STAT** command.

EXPT "string" #fail

Use the **EXPT** command to "**EXPECT**", or search for the specified *string* in any number of incoming lines.

- If the string is found, the script falls through to the next statement.
- If the string is not found, the script jumps to the statement specified in the *#fail* parameter.

Example:

```
EXPT "220 " #14
```

Possible Failures:

This command can fail if the expected text is not received before the connection closes. In that event, the script jumps to the statement specified by *#fail*.

However, if the timeout specified by a previous **WAIT** command expires before the connection closes, the script jumps to the *#idlefail* line specified by the **WAIT** command instead.

GOTO #statement

Use the **GOTO** command to branch immediately to the specified statement number.

Possible Failures:

If the statement number is out of bounds, the script terminates with a **DONE** command and **DOWN** status.

LINE [ON | OFF]

Use the **LINE** command to specify whether the script should read incoming data as lines or as raw data.

Notes:

- By default, the script reads in **LINE ON** mode. That is, the incoming data is read until it is terminated by a CR-LF or just a plain LF, and then the line is processed.
- If you issue a **LINE OFF** command, data is read without regard for line delimiters.

Reading raw data is useful for scanning HTTP data since web pages are not necessarily broken into lines. InterMapper's TCP probe has a maximum line buffer of 500 chars, so if lines are longer than that, they may be treated as separate lines.

Tip: Once you've matched some data in **LINE ON** mode, you shouldn't match any more because your position in the buffer is not restored and you may miss something.

Possible Failures: None

MTCH "string" #fail

Use the **MTCH** command to "**MaTCH**", or search for the specified string in the next incoming line. If found, the script falls through to the next statement.

Example:

```
MTCH "331" #16
```

Possible Failures:

If the next incoming line does not contain the desired string, or if the connection closes before the next line can be read, this script fails. In either case, the script jumps to the statement specified by *#fail*.

If the idle timeout expires instead, the script jumps to the *#idlefail* line specified by the previous **WAIT** command.

NADD "variable" #number

The **NADD** (**N**umeric **A**dd) command adds a numeric value to a variable. The variable is looked up and converted to a numeric value. The number is added, and the result is converted back to a string and placed into the variable.

Example:

```
NADD "fred" #3
adds 3 to the value of the variable fred. If fred contains "3", the result will be "6". If fred contains "golf", the result will be "3" (because the conversion from a string to a number yields zero).
```

If the number is missing, the script adds zero to the value.

Possible Failures: None.

NBGT #arg1 #arg2 #line

Use the **NBGT** (**N**umeric **B**ranch **G**reater **T**han) command to branch to *#line* if *#arg1* is greater than *#arg2*.

Example:

```
NBGT #${arg1} #${arg2} @exit
branches to the label @exit if the numeric ${arg1} is greater than ${arg2}.
```

Note: Use the leading **#** to force InterMapper to treat the arguments as numeric values.

Possible Failures: None.

NBNE #arg1 #arg2 #line

Use the **NBNE** (**N**umeric **B**ranch **N**ot **E**qual) command compares the two numeric arguments, and branches to the indicated *#line* if the arguments are not equal.

Example:

```
NBNE #${arg1} #${arg2} @exit
branches to the label @exit if the numeric ${arg1} is not equal to ${arg2}.
```

Possible Failures: None.

NEXT

The **NEXT** command clears the input buffer (represented by the `#{LINE}` variable) so that subsequent **MTCH** commands will operate on newly-received information.

Notes:

- The **SEND** command incorporates an implicit **NEXT** command.
- The **NEXT** command has no effect if input is not in **LINE** mode.

Possible Failures: None.

PORT #port_num #connect_timeout

Deprecated This command is no longer required in a script because the remote port number is now a separate parameter in the configuration dialog.

If present, this command must be in the first statement of the script. The first parameter specifies the default TCP port to connect to on the remote computer. The `#connect_timeout` parameter is the number of seconds to wait for the probe to connect.

Possible Failures: None

SBNE "arg1" "arg2" #line

The **SBNE** (**S**tring **B**ranch **N**ot **E**qual) command compares the two string arguments, and branches to the indicated `#line` if the arguments are not equal.

Example:

```
SBNE #{arg1} #{arg2} @exit
branches to the label @exit if the string #{arg1} is not equal to #{arg2}.
```

Possible Failures: None.

SCAT "variable1" "variable2" #fail

The **SCAT** (**S**tring **C**on**C**atenate) command concatenates `variable1` and `variable2` and places the resulting string in `variable1`.

Example:

```
STOR "name" "Fred"
sets the variable #{name} to the string "Fred"

SCAT "name" "Flintstone" @TOO_LONG
sets the variable #{name} to the value "FredFlintstone"
```

Possible Failures:

If the sum of the lengths of the strings exceeds 255, the SCAT command will fail, and transfer to the `@TOO_LONG` label.

SEND "string"

Use the **SEND** command to send the specified string to the remote device.

To send a line of data, you must explicitly specify the CR-LF using the quoting convention.

Example:

```
SEND "Greetings!\r\n"
transmits the data "Greetings!" followed by a CR-LF.
```

Possible Failures:

This command can't fail. If the data can't be sent because of a network failure or device failure, the failure shows up in a subsequent **EXPT** or **MTCH** command.

SKIP "string" #fail

Use the **SKIP** command to ignore all incoming lines containing the specified *string*. The script falls through to the next statement when an incoming line does *not* contain the *string*.

Possible Failures:

If the connection closes unexpectedly, the script jumps to `#fail`.

If the **WAIT** timeout (as defined by the **WAIT** command) expires, the script jumps to `#idlefail`.

STAT status ["message"]

Use the **STAT** command to specify the status of the device when the script ends. This command *does not* terminate the script. You can also specify a condition string as the second argument.

The status must be one of

```
[OKAY | WARN | ALRM | DOWN]
```

Example:

```
STAT ALRM "[HTTP] 500 Response received."
```

Note: A subsequent **STAT** or **DONE** command overrides the value set by this command.

STOR "variable" "string"

The **STOR** command stores the string into the variable named *variable*. If you wish to set a variable to a numeric value, enclose the number in quotes (""). Subsequent parts of the script can refer to this variable as `${variable}`.

Examples:

```
STOR "fred" "foobar"
```

sets the variable `fred` to the text string `foobar`. Subsequent parts of the script can refer to this variable as `${fred}`.

```
STOR "fred" "3"
```

sets the variable `fred` to the string value `"3"`.

Note: String variables can be any length up to 65,535 characters.

Possible Failures:

None.

STRT

The **STRT** command starts a millisecond timer that InterMapper can use to determine the elapsed time for some event. See the **TIME** command.

Example:

```
STRT Starts the timer.
```

Possible Failures: None.

TIME "variable"

The **TIME** command sets the named variable to the current number of milliseconds from the most recent **STRT** command.

Example:

```
TIME "connecttime"
```

sets the variable `connecttime` to the number of milliseconds since the most recent **STRT** command. If there was no previous **STRT** command, the variable will be set to zero.

Possible Failures: None.

WAIT #secs #idlefail #discfail

Use the **WAIT** command to specify the number of seconds the probe waits for a response.

Parameter 1 - #secs: The number of seconds to wait for a response. If you do not include a **WAIT** command in your script, the default timeout 60 seconds is used.

Parameter 2 - #idlefail: If present, the script jumps to this line number if the probe is idle for the specified number of seconds. This idle handler supercedes the error line number specified by the **EXPT**, **SKIP**, or **MTCH** commands. If the *#idlefail* parameter is not included, the script branches to the failure handler of the current command. [\[link\]](#)

Parameter 3 - #discfail: If present, the script jumps to this line if the probe is unexpectedly disconnected. This allows you to identify scripts that fail because of a TCP disconnection.

Possible Failures: None

Tip: You should specify all three parameters in the **WAIT** command.

Annotated Example of the FTP (Login) Script

```

01) PORT #21 (default tcp port)
02) WAIT #30 seconds
03) EXPT "220 " else goto -1- #14
04) SEND "USER ${User ID}\r\n"
05) MTCH "331" else goto -2- #16
06) SEND "PASS ${Password}\r\n"
07) MTCH "230" else goto -3- #20
08) SEND "NOOP\r\n"
09) MTCH "200" else goto -4- #24
10) SEND "QUIT\r\n"
11) EXPT "221" #+1 (i.e. can't fail)
12) DONE OKAY
13)
14) DONE DOWN "[FTP] Unexpected greeting from port ${_REMOTEPORT}.
${_LINE:50})" -1-
15)
16) MTCH "500" else goto #+2 -2-
17) DONE ALRM "[FTP] Port ${_REMOTEPORT} did not recognize the 'USER'
command."
18) DONE ALRM "[FTP] Unexpected response to USER command.
(${_LINE:50})"
19)
20) MTCH "530" else goto #+2 -3-
21) DONE WARN "[FTP] Incorrect login for \"${User ID}\"."
22) DONE ALRM "[FTP] Unexpected response to PASS command.
(${_LINE:50})"
23)
24) DONE ALRM "[FTP] Unexpected response to NOOP command.
(${_LINE:50})"

```

Explanation of the Script

```

01) PORT #21 (default tcp port)
02) WAIT #30 seconds

```

Line 1: The PORT command at the beginning of the script specifies the default TCP port number for FTP, port 21.

Line 2: The WAIT command specifies that if the script doesn't hear responses back within 30 seconds, it fails.

```

03) EXPT "220 " else goto -1- #14

```

Line 3: FTP servers normally send one or more "220" lines to greet new FTP control connections. Our script scans the incoming lines for "220 ".

Note the space following the 220; we don't want to match an incoming "220-"; the incoming dash indicates there are still more 220 lines to be read -- we only want to match the final 220 line.

If the script fails to find "220 " before the connection closes or within 30 seconds, the script branches to statement 14. The "-1-" is an arbitrary label used to make the destination of the branch more easily visible.

The string "else goto -1-" has no function (except readability) in the script command text; this statement could have been written equally well as EXPT "220 " #14 . Note that statement #14 also has comment of "-1-" to show it is the destination.

```

04) SEND "USER ${User ID}\r\n"

```

Line 4: Send the FTP USER command. With this command, we send the user ID specified by the user, e.g. "anonymous". Note that you must include the carriage-return and line-feed at the end of the string sent, to denote the line ending.

```
05) MTCH "331" else goto -2- #16
```

Line 5: The script looks for the 331 response to the USER command.

If something else arrives, the script jumps to statement 16. Unlike the *EXPT* command, the *MTCH* command fails immediately if the next line doesn't contain the required text.

[...] (Skipping down to statement 16).

```
16) MTCH "500" else goto #+2 -2-
17) DONE ALRM "[FTP] Port ${_REMOTEPORT} did not recognize the \"USER\"
command. "
18) DONE ALRM "[FTP] Unexpected response to USER command.
(${_LINE:50}) "
```

Line 16: Statement 16 is executed only if statement 5 fails; that is, if an unexpected response to the USER command is received. The response is checked to see if it matches "500", which would indicate that the command isn't supported. This is possible if you accidentally try to pass the USER command to a TCP service other than FTP.

If the server's response matches "500", the script is terminated with the device in the *ALARM* status (in statement 17). The message reports that the server did not recognize the USER command.

If the server's response does not match "500", the script skips two lines to statement 18. This statement terminates the script with the *ALARM* status and uses the `${LINE}` macro to include the first 50 characters of the response line in the message.

Customizing Status windows

If you are using a custom TCP or SNMP probe, you can override the default contents of pop-up Status windows. This is done using an optional section that defines data that will be added to the bottom of the Status window.

Custom SNMP Probes

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

The default font for the Status window's text is a monospaced font, so alignment of text is straightforward. You can change the appearance of the text in the Status window using the markup commands of the Description section.

Here is a sample `<snmp-device-display>` section. Note that the variables will be replaced with the values retrieved from the device.

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

Custom TCP-Based Probes

Use the optional `<script-output>` section to describe the text that appears in a TCP-based custom probe's pop-up 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.

TCP Probes

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.

Example TCP Probe File

The following is the Dartware-provided probe for the Custom TCP script.

```
<!--
Custom TCP (com.dartware.tcp.custom)
Copyright © 2000-2003 Dartware, 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>
```

Errors with Custom Probes

When working with custom probes, you may see results that you don't expect. Here are some common problems.

Error: 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.

The SNMP FAQ lists many other reasons.

Error: 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.

Error: 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.

Measuring Response Times

You can measure the response time of device as it is being tested. Times are measured in milliseconds. Depending on the probe type, different responses are measured:

- **TCP-based 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.
- **Timing of Pings and other datagram-based probes** - InterMapper measures the interval between sending the ping and receipt of its response time. The time is appears in the pop-up window, and can be charted and logged.

Time Measurement Probe Variables

TCP Timers are:

Connection initiation interval	<code>\${_connect}</code>	Records the time required to establish a connection.
Connection duration interval	<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:

STRT	Starts the probe's custom timer.
TIME varname	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 pop-up 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 Pop-up 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 pop-up 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 interrupt time, and 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 by such user interface actions as pulling down menus and switching applications. InterMapper's timing measurements are relatively more accurate when it's running in the background, since the user interface won't interfere.

Custom SNMP Probes

Using custom 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, custom 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 a custom 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 pop-up window.

Custom SNMP probes follow the same general format as other probe files. The `<description>` and `<parameter>` sections work as described for other probe files.

The `<header>` section of a custom SNMP probe file is similar to the standard `<header>` section, with a few differences. For more information, see Header Section of Custom 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 pop-up windows
- **The `<snmp-device-properties>` section** - Specifies certain aspects of the SNMP queries sent to the device

SNMP Probe Variables - 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.

`<snmp-device-variables>` Format

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	<p>The name used to represent the particular MIB value in this probe.</p> <ul style="list-style-type: none"> • A variable-name consists of letters, digits and an underscore, and must begin with a letter. • Variable-names are not case-sensitive. <p>Variable names are represented in the probe as <code>\${Variable-name}</code>. The variable-name does not have to match the corresponding name in the MIB, although it may be more convenient.</p>
OID	<p>The Object ID for the particular probe variable.</p> <p>Note: A scalar's OID must end in ".0" according to the SNMP specifications. Although it is common to leave off the suffix for scalar values, it is technically correct to include it, and InterMapper requires the suffix to construct the proper queries.</p> <p>InterMapper requests the OID and waits for the response. Currently, there is no facility for iterating across all rows of a particular table.</p>
Type	<p>May be one of the following:</p> <ul style="list-style-type: none"> • Default - Should be used for variables of type <i>Integer</i> and <i>DisplayString</i>. InterMapper automatically determines the type and displays the value properly. • 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. • Total-value - Displays the actual value of a counter or gauge, not a computed rate value. • Hexadecimal - Displays the data as hex digits.
Chart-legend	<p>(optional) - Provides a text label to be placed on any strip charts that incorporate this variable. Chart legends can contain variable names in the form <code>\${variable-name}</code>.</p>

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.

Thresholds - 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.)

Sample <snmp-device-threshold> Section

```
<snmp-device-thresholds>
  alarm: ${ipForwDatagrams} > 10 "ipForwarded datagrams too high"
  alarm: ${tcpCurrEstab} >= 1
  warning: ${ipForwDatagrams} > 5
  warning: ${ipForwDatagrams} <= 2
  warning: ${ipInHdrErrors} > 5
</snmp-device-threshold>
```

Creating Comparisons

Comparisons are evaluated in order from top to bottom until a comparison is triggered (result is *true*). 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 pop-up 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.

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.

SNMP Query Settings - 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. The items 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 does 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.

```
<snmp-device-  
properties>  
  
nomib2="true"  
  
pdutype="get-  
request"  
  
apcups="false  
"  
  
maxvars="10"  
<snmp-device-  
properties>
```

The Dartware MIB

Dartware 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 DEFINITIONS ::= BEGIN

IMPORTS
    enterprises
        FROM RFC1155-SMI
    DisplayString
        FROM RFC1213-MIB
    OBJECT-TYPE
        FROM RFC-1212
    TRAP-TYPE
        FROM RFC-1215;

-- Define SNMPv1 Traps sent by InterMapper 3.0 and later
-- (23 October 2000)

dartware      OBJECT IDENTIFIER ::= { enterprises 6306 }
notify        OBJECT IDENTIFIER ::= { dartware 2 }
intermapper   OBJECT IDENTIFIER ::= { notify 1 }

intermapperTimestamp OBJECT-TYPE
    SYNTAX      DisplayString (SIZE(0..255))
    ACCESS      read-only
    STATUS      mandatory
    DESCRIPTION
        "The current date and time, as a string."
    ::= { intermapper 1 }

intermapperMessage OBJECT-TYPE
    SYNTAX      DisplayString (SIZE(0..255))
    ACCESS      read-only
    STATUS      mandatory
    DESCRIPTION
        "The type of event: DOWN, UP, ALARM, WARN, OK, or TRAP."
    ::= { intermapper 2 }

intermapperDeviceName OBJECT-TYPE
    SYNTAX      DisplayString (SIZE(0..255))
    ACCESS      read-only
    STATUS      mandatory
    DESCRIPTION
        "The device's DNS name, as a string."
    ::= { intermapper 3 }

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

intermapperTrap TRAP-TYPE
    ENTERPRISE dartware
    VARIABLES { intermapperTimestamp, intermapperMessage, intermapperDeviceName,
intermapperCondition }
    DESCRIPTION
        "The SNMP trap that is generated by InterMapper as a notification option."
    ::= 1

END

```

Command Line Probes

InterMapper provides *command-line* probes for the Windows, Unix, and Mac OS X daemon versions. You can create a script or program (written in perl, C, C++, or your favorite language), then let InterMapper invoke it periodically. Your program's return value becomes the device's status on the InterMapper map.

InterMapper uses the name of the program/script and its arguments specified in the Command-line probe file, then invokes the program, passing it the arguments. InterMapper then sets the device's status based on the return code from the program/script. In addition, any data (up to 255 characters) written to the script's standard output file is used as the device's condition string, and will appear in the popup status window.

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. 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 (found at <http://nagiosplug.sourceforge.net/developer-guidelines.html>). This will result in probes/plugins that work for both InterMapper and Nagios.

Installing a Command-line Probe

To install and use a command-line probe:

1. 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.
2. Create a Command-line probe file (see the next section on this page for the format of this file) and place it in the *Probes* directory of the *InterMapper Settings* directory.
3. Reload the probes to make it available.

Command-line Probe File Format

The Command-line probe file has the following format. (Follow the Sample Command-line Probe at the end of this page.)

- Command-line probes follow the same format as described in Custom Probe File Format.
- In the **<header>** section of the file, make sure that the "type" is "cmd-line". As with all probes, the "package" should be your reversed domain name. By convention, it is also recommended that the first part of the "probe_name" be "cmd". The rest of the "probe_name" should be something which will indicate further what type of probe it is, as in "cmd.nagios.ping". The "version" should be "1.0" and the "address_type" should be "IP".
- The **<description>** section should contain a description that will remind users what the probe actually does.
- The **<command-line>** section allows you to specify the information needed to execute the commands for the probe. There are three values: **path**, **cmd**, and **arg**.
 - Use **path** 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 will not be used. The "path" 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 ":".
 - Use **cmd** 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" value if you'd like.
 - Use **arg** to specify arguments to the executable. This may be instead of or in addition to specifying them in the "cmd" value. We could have just as easily written our sample "cmd" value as a command and argument, like this:

```
cmd = "check_ping"  
arg = "-H ${ADDRESS} -w 100,10% -c  
1000,90%"
```

- Note the use of the "\${ADDRESS}" macro. This will be replaced with the address given when the device was created. You can use the "\${PORT}" macro to indicate the port given when the device was created.
- The **<command-exit>** section allows you to specify which results from the command indicate the four InterMapper device states. The four states are "down", "alarm", "warning", and "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-data>** section is not currently used and is ignored.
- The **<parameters>** section allows you to pass arguments into the probe. The named arguments can be added to the command line. For example, use \${Timeout} for an parameter specified as:

```
"Timeout" = "7"
```

- The **<output>** section is not currently used and is ignored.

Sample Command Line Probe

```
<!--
Command Line Nagios Plug-in Example (com.dartware.cmd.nagios.ping)
Copyright (c) 2003 Dartware, LLC. All rights reserved.
Nagios is a registered trademark of Ethan Galstad.
-->

<header>
type = "cmd-line"
package = "com.dartware"
probe_name = "cmd.nagios.ping"
human_name = "Nagios Ping"
version = "1.0"
address_type = "IP"
</header>

<description>
This probe uses the Nagios plug-in "check_ping" to ping the
monitored device. Nagios is a registered trademark of
Ethan Galstad.
</description>

<parameters>
</parameters>

<command-line>
path = "/Users/csweeney/bin:/usr/local/bin"
cmd = "check_ping -H ${ADDRESS} -w 100,10% -c 1000,90%"
</command-line>

<command-data>
  <!-- Currently unused. -->
</command-data>

<command-exit>
down: ${EXIT_CODE} = 2
alarm: ${EXIT_CODE} = 1
okay: ${EXIT_CODE} = 0
</command-exit>

<output>
  <!-- Currently unused. -->
</output>
```

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.

Example Notification from a Command Line Program

An interesting tool for notifications that we've found is iPing (<http://www.iping.com/>). This tool gives you several ways of generating by computer a notification that iPing will read to the recipient over the phone. You can use this tool in any version of InterMapper by making an email notifier where the email address is ping.<yourAccountName>@iping.com.

However, iPing also has an API, one method of which allows more control over how the notification is carried out. This method, putnotification, is invoked via http. With a command-line tool like "curl" and InterMapper's command-line notifiers, you can create a powerful and tailored iPing notifier for InterMapper.

Implementing an iPing Notifier

The curl utility is already available on many Unix systems. If it is not available on yours, or if you are using Windows, you can find out more about it and download a copy from <http://curl.haxx.se/>. You can find out more about the putnotification API method for iPing from http://www.iping.com/apinotes_putnotification.asp. The rest of this command-line notifier tutorial assumes that curl is installed and that you have an iPing account.

First, make sure that curl or an alias or soft link to it is in the Notifiers directory on the InterMapper server. The Notifiers directory is a subdirectory of the InterMapper Settings directory. If you do not have a Notifiers directory, you will need to create one. For security reasons, only executables in this directory may be executed as notifiers.

In InterMapper Console or InterMapper Remote, open the Server Settings dialog box and click on the **Notifier List** entry. Click on the **Add...** button and choose a type of **Command Line**. Give it a name.

In the "Command" field, enter the iPing notification command you want to use. A simple example to send a message to a phone number immediately would be:

```
curl https://www.iping.com/services/iping.asp?\
method_name=putnotification&user_name=testdriver\
&password=12345&phone_number=8448675309\
&notification_dt=now&msg_text_body=${ESCAPED_MESSAGE}
```

The parameters to the iPing message are:

- **method_name** must be "putnotification"
- **user_name** the name of your iPing account
- **password** the password of your iPing account
- **phone_number** the phone number to dial
- **notification_dt** Valid notification date/time (see the iPing documentation) or "now"
- **msg_text_body** The text of the message to be spoken.

Note 1: This text of the example above must be all on one line. If you're pasting the text into a script on a Unix system, leave the "\" characters at the end of lines and it will work properly.

Note 2: The msg_text_body must be in a form suitable for inclusion in a URL. In particular, the text should not contain spaces, ampersands (&), question marks (?) or a number of other characters. InterMapper provides two macros that make it easy to enter the text:

- **\${URLESCOPE: xxxxx}** This macro returns a string that contains the text ("xxxxx") with all the URL special characters escaped properly.
- **\${ESCAPED_MESSAGE}** This macro is a special case facility that performs the URL escaping function on the notifier's \${MESSAGE} string, as entered by the user. Click the "Edit Message" button to modify the default message. The default message is fairly short, as is appropriate for command-line notifications.

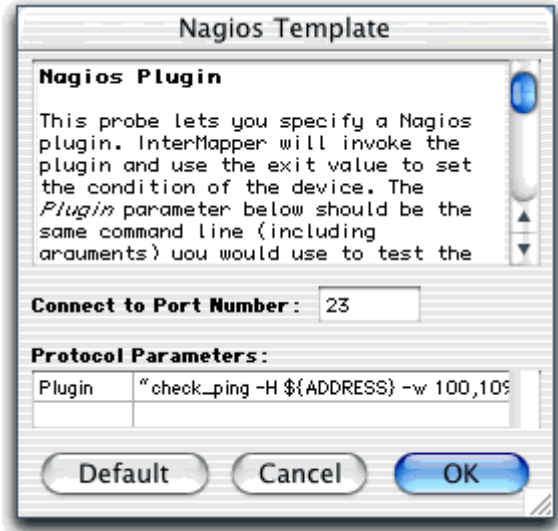
Note 3: The curl command generally exits with a code of zero. This avoids InterMapper log messages warning of unsuccessful notifications.

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.

To install and use a Nagios plugin:

1. Download the plugin. Make it runnable/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 Template**.
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-guidelines.html>). This will result in probes/plugins that work for both InterMapper and Nagios.

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.

Big Brother Probes

InterMapper can act as a Big Brother server. Big Brother is a popular network monitoring tool that allows you to create scripts ("clients") that run on remote systems and send status reports back to the Big Brother server. This allows a network manager to test additional kinds of network devices, either by writing scripts or using some of the many scripts that area already available.

When you specify a device to be tested with a Big Brother probe, InterMapper's built-in Big Brother server listens for messages coming from a Big Brother client on the corresponding machine.

Adding a device to use the Big Brother probe is simple, since there is only one configurable item—the port number. Normally this will be 1984, the default, but you may choose a different port. If you choose a different port, make sure that the Big Brother client on the corresponding machine is also configured for the same port, rather than 1984.

In order for InterMapper to receive Big Brother messages from the remote client, it must be configured correctly. In particular, the client must be configured so that its BBDISPLAY is set to the IP address of the machine where InterMapper is running.

The Big Brother states will be mapped to InterMapper states as shown in the table below:

Big Brother State	InterMapper Status
Okay (green)	Okay (green)
Attention (yellow)	Warning (yellow)
Trouble (red)	Alarm (orange)

At the moment, the only messages that InterMapper will process and represent are "status" (and "combo") messages.

Note that the Big Brother server for a given port will not start until at least one device has been configured for that port. Similarly, once the last device for that port has been removed, the server for that port will shut down.

For more information about Big Brother, check the Big Brother web site at <http://www.bb4.com/>. "Big Brother System and Network Monitor" is a trademark of BB4 Technologies, Inc. There's a good description of the Big Brother protocol at: http://spong.sourceforge.net/documentation/spong-2.7/developer-guide.html#BIG_BROTHER_PROTOCOL. You can also look through a large set of Big Brother clients that can be downloaded freely.

Using the Command Line Interface

You can call InterMapper Remote 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.

InterMapper Remote currently supports the following cmd-line arguments:

Argument	Description
-host --host <HOST>	connect to the specified HOST
-port --port <PORT>	connect to the specified PORT on HOST (defaults to 8181)
-map --map <MAP_NAME>	load the specified map(s) from HOST (separate map names with ":")
-f --file <FILE_NAME>	open the specified shortcut file
-d --debug <DEBUG_CONFIG_FILE>	use the specified configuration file to configure debugging output
-dmax --dmax <MAX_CHARS>	set the maximum number of characters in the debug window
-D<name>=<value>	set a system property
-user --user <USER>	log in as USER
-pass --pass <PASSWORD>	log in as USER with PASSWORD
-version --version	print product version
-env --env	print out system properties
-h -? --help	print this help message
-import --import <FILE_NAME>	import the specified file (use - for stdin)
-export --export <EXPORT-SPEC>	export the specified data to stdout

Examples for Import and Export commands

To import to a specified server, IM Remote is invoked as follows:

```
java -jar IM-Remote.jar --host big.dartware.com --user admin --pass adminpw
--import newdata.tab (read imported data from newdata.tab)

java -jar IM-Remote.jar --host big.dartware.com --user admin --pass adminpw
--import - (read imported data from stdin)
```

To export from a specified server, IM Remote will be invoked as follows:

```
java -jar IM-Remote.jar --host big.dartware.com --user admin --pass adminpw
--export 'format=tab table=devices fields=*' (write exported data to stdout)
```

The stdin form of the --import option allows Unix users to create self-contained, executable files that import stuff:

```
#!/usr/bin/java -jar IM-Remote.jar --host big.dartware.com --import -
#import blah blah blah
blah blah blah
blah blah blah
```

One use for this would be to automate testing of InterMapper Server.

Customizing Web Pages

InterMapper comes with a set of default web page layouts, as shown in The Web Server. Use this section to learn how to customize those pages by modifying the files that InterMapper uses creating them.

InterMapper's built-in web server generates pages based on files in its Web Pages directory. 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 Files directory** - Controls which web pages are available to Administrators and Guests.

Tip: The target and template files are simply text files. You may edit them with any text editor.

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 Settings category, choose **Servers**. The Servers 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 will simply display 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 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. The built-in files are:

- **!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.

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 `` 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:

#template	<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>
#title	<pre>#title "This is a Test Page"</pre> <p>The text in quotes becomes the title of the page - it enclosed in <code><title>...</title></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>
#alt_title	<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	<pre>#filename "otherpage.html"</pre> <p>The optional <code>#filename</code> directive causes InterMapper to treat the file as if named with the quoted string.</p> <p>For example, a target file named <code>"xindex.html"</code> could have a <code>#filename "xindex.html"</code>. This causes the target file to be used in place of the file named <code>"!index.html"</code> if its version number is higher. This can be useful for debugging as well as experimenting with alternate pages.</p>
#version	<pre>#version "2.1"</pre> <p>The optional <code>#version</code> directive determines which file is used when there are two or more instances of the same filename (as a result of using <code>#filename</code> directives.)</p> <p>The optional <code>#version</code> 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:</p> <ul style="list-style-type: none">• 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 <code>#version</code> to make it take precedence over InterMapper's built-in copy.• If you edit the file without changing the <code>#version</code>, 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 <code>#version</code> 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 take *other-information* which follows the ":". The entire macro will be replaced by the appropriate text when the page is generated.

Macros fall into these general categories:

- The Include Macro
- Macros that generate "content" of 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 that describe the requested URL

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.

`${fullstatus}` shows a list of all the devices and links for the map named in the URL.

`${errorstatus}` shows only the devices and links which are in warning or alarm states, or down for the named map.

`${currentoutages}` shows the list of current outages -- devices or links that are currently in warning, alarm, or are down in the named map.

`${previousoutages}` shows the list of devices that had been listed as outages but have since returned to normal.

`${previousoutages:hours=xx}` shows the list of previous outages within the last xx hours.

`${previousoutages:maxrows=x}` shows a list of the last x previous outages.

`${maplist}` shows an HTML unnumbered list () of the maps available.

`${chartlist}` shows an HTML unnumbered list of charts for the map.

`${maplistwithcharts}` shows an HTML unnumbered list of the maps available, with sub-lists of the charts for each map

Miscellaneous macros that describe InterMapper and its environment

- `#{abouthtml}`** shows the "About" page with the current version of InterMapper.
- `#{statshtml}`** shows InterMapper's statistics: uptime, memory usage, etc.
- `#{httpuserid}`** the name the user typed when asked for authentication
- `#{httpremoteaddress}`** the IP address of the remote browser.
- `#{intermapperaddress}`** the IP address of this InterMapper server
- `#{version}`** the version of this copy of InterMapper
- `#{date}`** the current date
- `#{time}`** the current time
- `#{imagesuffix}`** set to ".png" if the web client can display PNG images, or ".jpeg" otherwise.
- `#{telnetserverurl}`** a telnet: URL that connects to this InterMapper Telnet server
- `#{webserverurl}`** a http: URL that connects to this InterMapper server
- `#{mapname}`** the current map's name
- `#{deviceaddress}`** the IP or AppleTalk address of the particular device. For anything that isn't a device, an empty string is returned.
- `#{devicename}`** the DNS name or AppleTalk NBP name of the particular device. It is an empty string for anything that isn't a device.
- `#{pagetitle}`** displays the value set by the `#title` directive
- `#{SetNameFieldWidth:xx}`** 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}`** creates an `` tag to place an image on the page
- `#{imagesuffix}`** returns ".png" or ".jpeg", depending on the file format the web browser supports.
- `#{intermapperlogo}`** creates an `<img... >` tag that includes the "Made with InterMapper" logo.

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.

- `#{htmlrefreshmetatag}`** is either an empty string or the previous refresh choice from the web client. (Inserts a `<meta http-equiv="refresh"...>` tag on the resulting page.)
- `#{htmlrefreshmetaoptions}`** is the option list that a web client can choose from. The current `#{htmlrefreshmetatag}` value is selected. Note that your HTML template should supply the `<form><select>...</select></form>` surrounding this `#{htmlrefreshmetaoptions}` macro.

Macros that describe the requested URL

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
```

- `\${webpageurl}`** the full URL of the requested web page. (e.g., the full URL as shown above)
- `\${httppath}`** the full path to the file requested (e.g., "/Map1/device/192.168.0.1%3ASNMP!/device.html")
- `\${httpdocument}`** the top level directory of the page requested. Also an alias for `\${mapname}` (e.g., "Map1")
- `\${httpclass}`** the second level directory of the page requested (device, chart, link, document, network) (e.g., "device")
- `\${httpinstance}`** the third level directory of the page requested (e.g., "192.168.0.1%3ASNMP")
- `\${httpmethod}`** the fourth-level part of the page requested (e.g., "!device.html")
- `\${httpinstancepath}`** a concatenation of `\${httpdocument}`, `\${httpclass}`, `\${instance}` separated by "/" (e.g., "/Map1/device/192.168.0.1%3ASNMP")

Folder Structure

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.

InterMapper ships with four folders stored in the *Web Pages* folder:

~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`

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.wmlwmls      text/vnd.wap.wmlscript  
wbmp     image/vnd.wap.wbmp  
wbxml    application/vnd.wap.wbxmlwmlc  
         application/vnd.wap.wmlcwmlsc  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?time=XXXXX`

Glossary

C

CSR: Certificate Signing Request

I

InterMapper Console: Every InterMapper server ships with a copy of InterMapper Console. This is a copy of the InterMapper Remote application that can connect only to the InterMapper server that runs on the machine as InterMapper Console.

J

JPEG: Joint Photographers Experts Group

M

MIB Variable: MIB: (Management Information Base). A MIB defines all the settings and operational statistics that a device can report using SNMP. Among other things, a MIB defines these attributes: the exact technical definition for these statistics (the "MIB variables"), a text name for each variable, and a unique Object Identifier (OID) that a program (such as InterMapper) could use to retrieve them.

N

notification: The action performed by a notifier, as viewed by the notifier target, or recipient, of the action.

notifier: A little "robot" that watches the state of one or more devices, and performs a specified action when the device changes to a state identified as requiring action in the notifier's parameters.

notifier parameters: The specific information for the notifier, dependent on the Notifier type.

notifier schedule: The hours during which notifications are sent for that notifier.

notifier target: The recipient of a notification.

P

PNG: Portable Network Graphics

S

SNMP: Simple Network Management Protocol - an application-layer protocol which allows an application (like InterMapper) to query other network devices (such as a router, switch, server, etc.) to get information about the device, or to send control commands to the device.