CHAPTER 6

Customizing MWTM GUI Troubleshooting Commands

The MWTM GUI provides a Troubleshooting tab that enables operators to run commonly used IOS commands on nodes and objects that the MWTM manages. Most of these IOS commands require no user input and are run on the object when you click the Execute button.

Some commands require user input to complete. You identify user-input commands by the ellipsis (...) that follows the command. For example:

```
APN Statistics for APN Input...
```

The following sections in this chapter describe how to create troubleshooting commands for the MWTM GUI:

For information about the troubleshooting features of the MWTM, see the *User Guide for the Cisco Mobile Wireless Transport Manager 6.1.2*.

Creating User-Defined Variables

The following topics describe how to define a variable in a troubleshooting command:

Understanding System-Defined and User-Defined Input Data

The MWTM provides two files for controlling input data:

File Name and Location	Description
/opt/CSCOsgm/etc/SystemDefinedInputData.ts	Enables *Cisco engineers* to define system-level variables for the MWTM GUI.
/opt/CSCOsgm/etc/UserDefinedInputData.ts	Enables *MWTM system administrators* to define user-defined variables for the MWTM GUI.

If you want to modify a command that is already defined in **SystemDefinedInputData.ts**, you *redefine* it in **UserDefinedInputData.ts**. This approach allows you to change the way the MWTM GUI queries its users.

Note Changes to these two files do not require a restart of the server. However, you must refresh your GUI client by selecting a different managed object in the navigation tree to apply the change.

Understanding the User Variable File and Format

To define a user variable in the **UserDefinedInputData.ts** file (see Understanding System-Defined and User-Defined Input Data, page 6-2), create a line with this format:

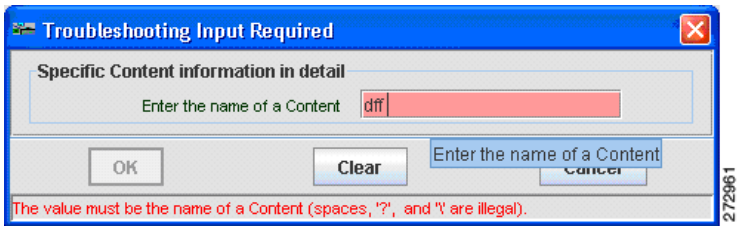
```
UserVariable=label string| tool tip text|regular expression| formatting error message
```

Example 1—User-Defined Variables

```
CsgContent= Enter the name of a Content| Enter the name of a
Content|IOS_WORD_WITH_SPECIAL_CHARS| The value must be the name of a Content (spaces, '?',
and '\' are illegal).
```

The resulting user input window for this variable is shown in Figure 6-1:

Figure 6-1 Troubleshooting Input Required for CSG Content



Notice the correlation between the values in the columns of the **CsgContent** variable and the fields in the resulting window:

User Variable Column	Window Fields in Figure 6-1
label string	Enter the name of a Content
tool tip text	Enter the name of a Content
regular expression	IOS_WORD_WITH_SPECIAL_CHARS
formatting error message	The value must be the name of a Content (spaces, '?', and '\' are illegal)

Example 2—User-defined Variables

To define **CsgContent** without a tool tip, leave the tool tip column (column 2) empty. No tool tip would appear in the resulting window. The format for this line would be:

```
CsgContent= Enter the name of a Content| |IOS_WORD_WITH_SPECIAL_CHARS| The value must be
the name of a Content (spaces, '?', and '\' are illegal).
```

Understanding the Regular Expression

To use a regular expression (REGEX) when creating a user variable, you must first define the REGEX in the **UserDefinedInputData.ts** file. The REGEX must follow this format:

REGEX (REGEX_NAME,REGULAR_EXPRESSION)

In the previous example (Example 1—User-Defined Variables, page 6-2), IOS_WORD_WITH_SPECIAL_CHARS is a defined regular expression and appears in the **UserDefinedInputData.ts** file like this:

```
REGEX(IOS_WORD_WITH_SPECIAL_CHARS,[.[^\\?\s]]+)
```

Defining a REGEX is not a requirement. For example, if we chose not to define a REGEX in Example 1—User-Defined Variables, page 6-2, then the user variable would need to be:

```
CsgContent= Enter the name of a Content| Enter the name of a Content|[.[^\\?\s]]+ |
The value must be the name of a Content (spaces, '?', and '\' are illegal).
```

This approach is acceptable, however:

- If you do not define a REGEX, then you will not be able to reuse it when creating other user variables where the expression might be needed. For example, the variables ip_subnet and ip_address both use this defined regular expression: IP_ADDRESS.
- If you do not define a REGEX, then you must be careful not to insert extraneous white space (spaces or tabs) in the regular expression within the user variable. The MWTM attempts to interpret extraneous white space before or after the expression between the delimiters. See Cautions about White Space in the REGEX, page 6-3, for more information.

Cautions about White Space in the REGEX

Continuing with our example, assume that we do not define a REGEX for the **CsgContent** variable but include the regular expression like this:

```
CsgContent= Enter the name of a Content| Enter the name of a
Content|     [.[^\\?\s]]+     | The value must be the name of a Content (spaces, '?', and
'\' are illegal).
```

In this case, we inserted five spaces before and after the [.[^\\?\s]]+ expression. This extraneous white space means that the name of the CSG content must begin with five spaces and end with five spaces. Of course, requiring the user to enter these spaces is not desirable!

But if we defined the REGEX as follows:

```
REGEX(IOS_WORD_WITH_SPECIAL_CHARS,[.[^\\?\s]]+)
```

and then created the user variable with extraneous white space around the defined REGEX:

```
CsgContent= Enter the name of a Content| Enter the name of a
Content|     IOS_WORD_WITH_SPECIAL_CHARS     | The value must be the name of a Content
(spaces, '?', and '\' are illegal).
```

then the extra white space is ignored and the user is not required to enter spaces before or after the CSG content value.

Remember, you must never have extra spaces in the defined REGEX (unless you intend the spaces to be part of the REGEX).

For example, this REGEX is valid:

```
REGEX(IOS_WORD_WITH_SPECIAL_CHARS,[.[^\\?\s]]+)
```

but this REGEX is not valid:

```
REGEX(IOS_WORD_WITH_SPECIAL_CHARS,     [.[^\\?\s]]+     )
```

Also, by defining a REGEX, you only have to be careful about white space once. Thereafter, when using the REGEX to create user variables, you only need to insert the REGEX name in the correct column of the user variable.

Note The MWTM trims white space around other elements of the user variable (for example, the label string and tool tip text). But the MWTM does not trim extraneous white space around the REGEX.

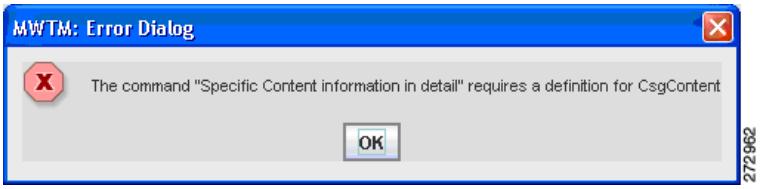
Undefined Variable Error Messages

If you use a variable that has not been defined, the MWTM returns an error message when you run the command. For example, if you run this command:

```
CSG2 Application Traffic - System, CSG2, show ip csg content name %CsgContent detail,
Specific Content information in detail
```

but you did not previously define **%CsgContent** in the **UserDefinedInputData.ts** file, the MWTM produces the error dialog box shown in Figure 6-2:

Figure 6-2 Error Dialog for Undefined User Variable



Creating User-Defined Commands

The MWTM troubleshooting framework supports Managed Information Base (MIB) capabilities. An MWTM engineer defines a MIB capability as a set of variables that can optionally span multiple MIBs belonging to a node. For example, the C7600 MIB capability defines variables that only a Cisco 7600 node can have.

The following topics describe how to create or modify commands that define MIB capabilities:

Related Topics

For a list of the MIB capabilities defined for MWTM 6.1.2, see Appendix J, "MWTM MIB Capabilities".

Understanding System-Defined and User-Defined Commands

After you have defined variables in the **UserDefinedInputData.ts** file, you can use them to create user-defined commands. The MWTM provides two files for defining commands:

File Name and Location	Description
/opt/CSCOsgm/etc/SystemCommands.ts	Enables *Cisco engineers* to create commands.
/opt/CSCOsgm/etc/UserCommands.ts	Enables *MWTM system administrators* to modify these commands or create new ones.

Cisco defines commands in the **SystemCommands.ts** file. You can modify these commands or create new ones in the **UserCommands.ts** file.

Understanding the Command Syntax

The syntax for defining commands is:

<*category*>, <*MIB capability*>, <*command*>, <*description*>, <*condition*>

where:

- category—is a user defined category used for grouping related commands
- capability—is one or more MIB capabilities. For a list of the capabilities defined for MWTM 6.1.2, see Appendix J, "MWTM MIB Capabilities".
- command—is a valid IOS command string
- description—is a description of the command
- condition—is an optional parameter which, if present, must evaluate to true for the command to be displayed to the user. See Defining Conditions, page 6-8.

Notes

- The "," character is used as a field delimiter character and therefore may not be used within any of the fields.
- Any lines not entered in the above format will be ignored.

Example

For example, the following command is from the **SystemCommands.ts** file:

```
General, ALL, show interfaces, All network interfaces
```

Prior to this release (MWTM 6.1.2), the MIB capability column (ALL) was restricted to these MIB values:

Legacy MIB Capability Value	Used For
ITP	ITP nodes
CSR	Cell Site Router nodes
BWG	Broadband wireless gateway nodes
CSG2	CSG2 nodes
RAN_SVC	RAN service modules
ALL	All node types

These legacy MIB capabilities are still valid, but now any MIB capability can be included. For a list of the available capabilities, see Appendix J, "MWTM MIB Capabilities".

Defining User Commands

Continuing with the CSG content example, the following line appears in the **UserCommands.ts** file:

```
CSG2 Application Traffic - System, CSG2, show ip csg content name %CsgContent detail,
Specific Content information in detail
```

Note When adding a user-defined variable in the **UserCommands.ts** file, always include a % character in front of the user variable.

A single command can have more than one user-defined variable. The following command incorporates two user variables:

```
CSG2 Application Traffic - User, CSG2, show ip csg sessions users rtsp %ip_address
%ip_subnet detail, RTSP Session users in detail for IP Address
```

Example 1

This command example from the **SystemCommands.ts** file demonstrates the use of a new MIB capability:

```
SAMI, SAMI&HA, show sami ipcp statistics detail, Display SAMI ipcp statistics detailed
information
```

In this example, the troubleshooting command (**show sami ipcp statistics detail**) will appear in the MWTM GUI only if the selected node has both SAMI and HA MIB capabilities. You can add (&) as many MIB capabilities as required.

Example 2

If you want the AAA and (&) BBB capabilities or the CCC and (&) DDD capabilities (not real capabilities), enter the line in the file twice:

```
SAMI, AAA&BBB, show sami ipcp statistics detail, Display SAMI ipcp statistics detailed
information
SAMI, CCC&DDD, show sami ipcp statistics detail, Display SAMI ipcp statistics detailed
information
```

The troubleshooting command will appear only if the selected node has both AAA and BBB capabilities, or if the node has both CCC and DDD capabilities.

If a network node has all four capabilities, the troubleshooting command will only appear once in the MWTM GUI because the framework allows one only description (the fourth column) for each category (the first column). If your network had a node with all four capabilities, the description (Display SAMI ipcp statistics detailed information) would appear only once for the SAMI category.

Note You should always ensure the variable exists before you use it. For example, the following code fragment shows how you can ensure that an object exists before you invoke a method for that object:

```
$This.SysName && $This.SysName.startsWith("emssami")
```

In this example, **$This.SysName** is the object and **startsWith** is the method. When testing compound conditionals, the conditionals are tested left to right, and processing continues to the right only if the left conditional evaluates to true. So in this case, if **$This.SysName** evaluates to false (it does not exist), then the **startsWith** method will not be evaluated.

Example 3

This example adds the show log command to the General category with the Description of System log.

The System log command appears in the drop-down menus of all nodes because of this line in the **SystemCommands.ts** file:

```
General, ALL, show log, System log
```

The ALL value ensures that the command can run on any managed node that has support for troubleshooting.

If you want to allow operators to run this command only on Cisco 7600 nodes that run the ITP feature, you can insert this line in the **UserCommands.ts** file:

```
General, C7600&ITP, show log, System log
```

Based on this line, MWTM operators can run the System log command only if they select a Cisco 7600 node that is configured as an ITP. If the operator selects an ITP node on a Cisco 7200 device or a RAN node on a Cisco 7600 device, the System log command will not appear in the MWTM GUI drop-down menus.

Example 4

If you want to extend the System log command (see Example 3) to all devices that carry PWE3 traffic, use the MIB capability CISCO_IETF_PW_MIB. For example:

```
General, CISCO_IETF_PW_MIB, show log, System log
```

Adding this line in the **UserCommands.ts** file enables MWTM operators to run the System log command on nodes that are carrying PWE3 traffic.

The **UserCommands.ts** file in our example now has these lines for the System log command:

```
General, C7600&ITP, show log, System log
General, CISCO_IETF_PW_MIB, show log, System log
```

With these lines, an MWTM operator can run the System log command on either a Cisco 7600 ITP node or a node that is carrying PWE3 traffic.

Defining Conditions

The **SystemCommands.ts** and **UserCommands.ts** files allow an optional fifth column to appear in the command line. This fifth column allows you to define conditions (see Understanding the Command Syntax, page 6-5).

The Apache Velocity Engine evaluates the condition contained in the fifth column. To understand more about this tool and how to define conditions, read the documentation that is available at the `velocity.apache.org` web site.

Example 1

The following example is from the **SystemCommands.ts** file:

```
CS7, ITP, show cs7 $SP.InstanceID linkset $This.RDN, Current linkset, $This.NEType &&
($This.NEType == "Linkset")
```

Note The commas are delimiters that help you distinguish the contents of each column.

The velocity engine evaluates the condition shown in the fifth column based on the underlying managed object. In our example, if the MIB capability in column two (ITP) is satisfied, and the selected object is an ITP linkset, the command (shown in the third column) will appear in the Description drop-down menu if the operator chooses the CS7 category.

You can define conditions in the same way that you define regular expressions (see Understanding the Regular Expression, page 6-3). This approach facilitates reusing defined conditions in multiple commands and enables you to debug a condition once rather than many times.

The **SystemCommands.ts** file defines the condition contained in Example 1 as follows:

```
CONDITIONAL(LINKSET,$This.NEType && ($This.NEType == "Linkset"))
```

With this condition defined, the command definition shown in Example 1 becomes:

```
CS7, ITP, show cs7 $SP.InstanceID linkset $This.RDN, Current linkset, LINKSET
```

Example 2

While %variables are used in *commands* for variable substitution, $variables are used in *conditions* that determine whether to present a command to a user. The $variables can be anything defined as a network element attribute for the currently selected object.

The following code fragment shows how a $variables are evaluated in a conditional statement:

```
$This.NEType && $This.Type && ($This.Type != "ds1") && ($This.NEType == "Interface")
```

In this example:

- If **$This.NEType** exists, the processing continues right.
- If **$This.Type** exists, the processing continues right.
- If **$This.Type** is not **ds1,** the processing continues right.
- If **$This.NEType** is **Interface,** the current conditional evaluates to true and the current command is included in the list for the current object.

Network element attributes are listed in Appendix E, "MWTM Monitor Attributes".

Example 3

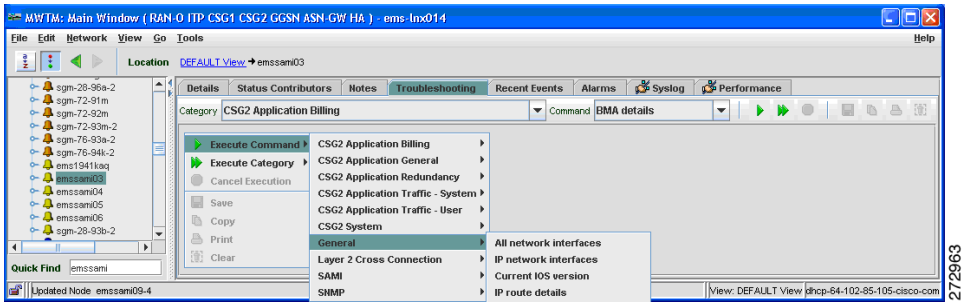
In the **SystemCommands.ts** file, this command definition appears:

```
General, ALL, show log, System log
```

This command, System log, will appear in the MWTM GUI for any managed node.

You can customize this command definition to allow MWTM operators to run the System log command only on managed nodes that begin with a specified naming convention. In this example, we would like to allow the show log command to appear for node names that begin with emssami (see Figure 6-3 for example).

Figure 6-3 Example of Node Name Beginning with emssami



Note Figure 6-3 demonstrates that the System log command does not appear in the GUI for emssami03, the node that is selected in the navigation tree (left pane).

Table E-18 in Appendix E, "MWTM Monitor Attributes," describes attributes for managed nodes. This table shows that the SysName attribute provides the system name for a node:

```
SysName = sysName for Node
```

To restrict the node types that we want to define, we will use the $This variable. Adding the Sysname attribute from Table E-18, we get:

```
$This.SysName.
```

Because we want to include only those nodes that start with emssami, we further define the variable like this:

```
$This.SysName.startsWith("emssami")
```

To ensure that $This.SysName will include the managed node that the MWTM operator selects in the navigation tree, we arrive at this definition:

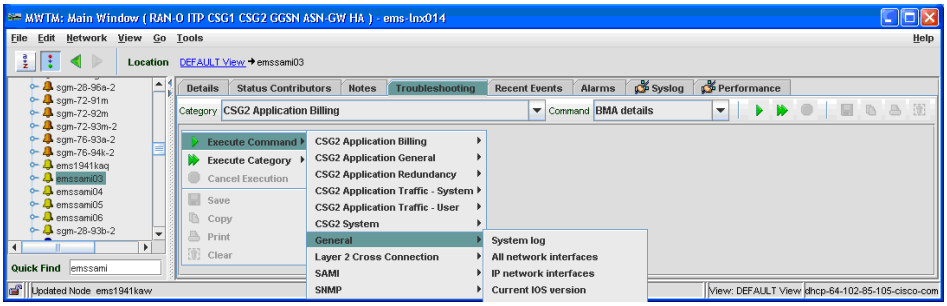
```
$This.SysName && $This.SysName.startsWith("emssami")
```

Now we can add the variable definition to the command definition in the **UserCommands.ts** file:

```
General, ALL, show log, System log, $This.SysName &&
$This.SysName.startsWith("emssami")
```

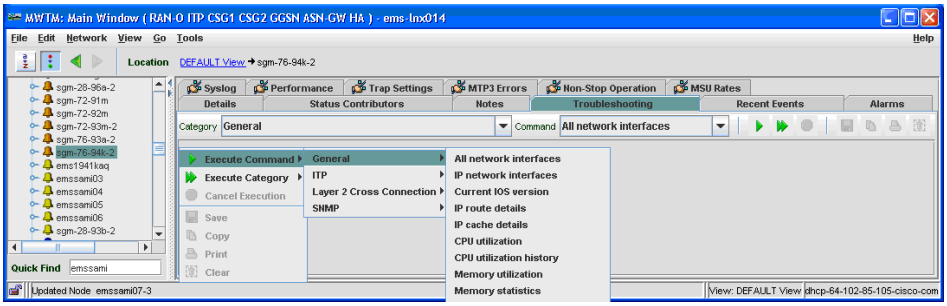
Figure 6-4 shows an example of a node that matches the variable definition (that is, the selected node begins with emssami).

Figure 6-4 Example of Node Matching Variable Definition



If the MWTM operator selects a node that begins with anything other than emssami, the System log command does not appear in the GUI drop-down menu (Figure 6-5). Despite the fact that the MIB value is set to ALL, the System log command only appears for managed nodes that meet our naming convention.

Figure 6-5 Example of Node Not Matching Variable Definition



Example 4

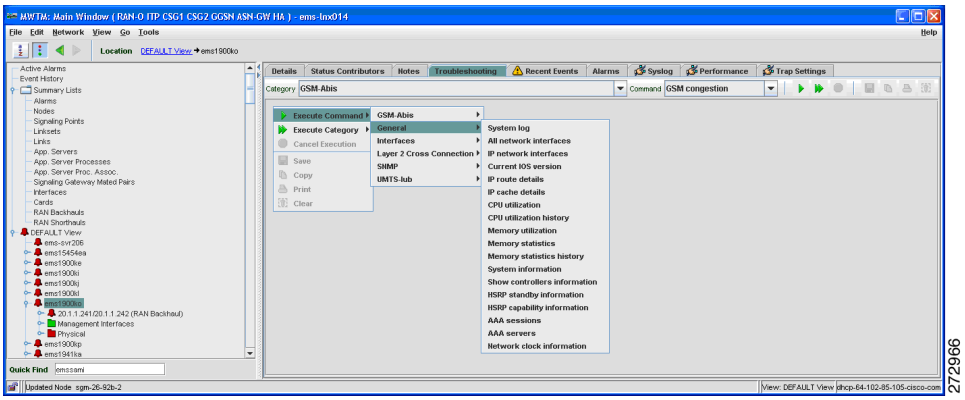
Building on the variable defined in Example 3, we will modify the definition to include node names that begin with ems19.

Our expanded variable definition becomes:

```
$This.SysName && ($This.SysName.startsWith("emssami02") ||
$This.SysName.startsWith("ems19"))
```

With this change, a node name beginning with ems19 now has the System log command appearing in the GUI drop-down menu (Figure 6-6).

Figure 6-6 Example of Node Matching the Expanded Variable Definition



Example 5

Further expanding the variable defined in Example 4, we can restrict the System log command to appear only for top-level managed objects by adding another condition:

```
General, ALL, show log, System log,$This.SysName && $This.NEType &&
(($This.SysName.startsWith("emssami") || $This.SysName.startsWith("ems19"))&&
($This.NEType == "Node"))
```

In this example, the managed node must have the network element type (NEType) defined. The additional condition restricts the System log command to nodes and not to objects that subtend nodes in the tree hierarchy.