IBM Tivoli Monitoring

Workbench User's Guide

Version 5.1

SH19-4571-00

IBM Tivoli Monitoring

Workbench User's Guide

Version 5.1

SH19-4571-00

IBM Tivoli Monitoring Workbench User's Guide, Version 5.1

ISO 9001 Certification

This product was developed using an ISO 9001 certified quality system.

Certification has been awarded by Bureau Veritas Quality International (BVQI) (Certification No. BVQI - 92086 / A).

BVQI is a world leader in quality certification and is currently recognized by more than 20 accreditation bodies.

Contents

Figures

Preface

IBM® Tivoli® Monitoring 5.1 provides services for monitoring the performance of your resources, including disks, CPU, and applications. This helps you to automatically detect bottlenecks, and potential resource problems, and act on them proactively. To monitor the system resources, IBM Tivoli Monitoring 5.1 uses standard resource models and new ones that can be created with IBM Tivoli Monitoring Workbench 5.1.

IBM Tivoli Monitoring Workbench 5.1 implements WMI technology to access management information, making it possible to combine information from various hardware and software management systems. This guide describes how to use IBM Tivoli Monitoring Workbench 5.1 to create new resource models or to modify existing resource models.

Who Should Read This Guide

This guide is intended for developers, who use IBM Tivoli Monitoring Workbench 5.1 to create resource models that can be used with IBM Tivoli Monitoring 5.1. It is also useful for performance analysts, who may wish to develop new resource models themselves, or modify the existing ones. System administrators may also need this book to understand how resource models are created, and what functions can be built into resource models using IBM Tivoli Monitoring Workbench 5.1. Users of this guide must be familiar with the following:

- IBM Tivoli Monitoring 5.1
- Visual Basic or Java™ Script programming
- Windows® Management Instrumentation (WMI)
- System fundamentals, including system resources and network characteristics
- Operating system fundamentals

What This Guide Contains

The *IBM Tivoli Monitoring Workbench User's Guide, Version 5.1* contains the following sections:

- Chapter 1, "Introduction"

  Provides an overview of IBM Tivoli Monitoring Workbench 5.1 and its integration with IBM Tivoli Monitoring 5.1.

- Chapter 2, "Resource Models and Related Concepts"

  Describes, resource models and their components, and illustrates some general concepts related to them.

- Chapter 3, "Installation"

  Describes the installation process.

- Chapter 4, "Designing a Resource Model"

  Provides some information about designing a resource model. It also gives a concrete example of a resource model.

- Chapter 5, "Creating a Resource Model"

  Describes the procedures for creating, debugging and building a new resource model.

- Chapter 6, “Resource Model Troubleshooting”

  Describes resource model possible outcomes.

- Appendix A, “Service Object Method Library”

  Provides syntax for creating objects for your resource models. It also includes a list of possible errors.

- Appendix B, “Examples of Resource Model Creation”

  Provides some examples of resource models, which are explained and documented on the basis of the tasks you can perform and the settings you can specify.

- Appendix C, “Resource Models for Microsoft Exchange Server”

  Provides a description of Microsoft Exchange Server resource models and some instructions to use them on Tivoli Monitoring.

- Appendix D, “Instrumentation Library Type Interface”

  Describes the Instrumentation Library Type (ILT) interface implemented by Tivoli Monitoring and provides guidelines for writing a UNIX provider.

- Appendix E, “Error Messages”

  Provides a list of the error messages you can get, with the corresponding explanations and user actions.

Publications

This section lists publications in the IBM Tivoli Monitoring library and any other related documents. It also describes how to access Tivoli publications online, how to order Tivoli publications, and how to make comments on Tivoli publications.

IBM Tivoli Monitoring Library

The following documents are available in the IBM Tivoli Monitoring library:

- *IBM Tivoli Monitoring Workbench User’s Guide, Version 5.1*, SH19-4571

  Describes how to use IBM Tivoli Monitoring Workbench 5.1 to create new resource models or to modify existing resource models.

- *IBM Tivoli Monitoring User’s Guide, Version 5.1*, SH19-4569

  Describes how to install, customize, and use IBM Tivoli Monitoring 5.1 to manage system and application resources.

- *IBM Tivoli Monitoring Resource Model Reference, Version 5.1*, SH19-4570

  Provides a list of resource models you can use with IBM Tivoli Monitoring 5.1.

- *IBM Tivoli Monitoring Release Notes, Version 5.1*, GI10-5797

  Provides late-breaking information about the products of this library.

Versions of these documents in PDF and HTML formats can be found on the IBM Tivoli Monitoring 5.1 product CD. They are stored in the `Books` directory, and can be accessed by selecting the file `Books/infocenter.html` with your Web browser. This displays an HTML page from which all of the documents can be accessed in either format.

Updated versions of these documents might be placed from time-to-time on the Tivoli Customer Support Web site (see “Accessing Publications Online” on page xi for more details).

Prerequisite Publications

To be able to use the information in this book effectively, you must have some prerequisite knowledge, which you can get from the following books:

- *Implementing Tivoli Manager for Windows NT®, SG24-5519*

  Provides information about IBM Tivoli Monitoring used on Windows NT platforms. This is a redbook, available from http://www.redbooks.ibm.com

- WMI documentation:

  Provides information about WMI technology and some important concepts used in this guide. This is available at:

  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/aboutwmi_1lpl.asp

- CIM documentation:

  Provides information about CIM technology and some important concepts used in this guide. This is available at:

  http://www.dmtf.org/spec/cims.html

Accessing Publications Online

You can access many Tivoli publications online at the Tivoli Customer Support Web site:

http://www.tivoli.com/support/documents/

These publications are available in PDF or HTML format, or both. Translated documents are also available for some products.

Ordering Publications

You can order many Tivoli publications online at the following Web site:

http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi

You can also order by telephone by calling one of these numbers:

- In the United States: 800-879-2755
- In Canada: 800-426-4968
- In other countries, for a list of telephone numbers, see the following Web site: http://www.tivoli.com/inside/store/lit_order.html

Providing Feedback about Publications

We are very interested in hearing about your experience with Tivoli products and documentation, and we welcome your suggestions for improvements. If you have comments or suggestions about our products and documentation, contact us in one of the following ways:

- Send an e-mail to pubs@tivoli.com.
- Complete our customer feedback survey at the following Web site: http://www.tivoli.com/support/survey

Contacting Customer Support

If you have a problem with any Tivoli product, you can contact Tivoli Customer Support. See the *Tivoli Customer Support Handbook* at the following Web site:

http://www.tivoli.com/support/handbook/

The handbook provides information about how to contact Tivoli Customer Support, depending on the severity of your problem, and the following information:

- Registration and eligibility
- Telephone numbers and e-mail addresses, depending on the country you are in
- What information you should gather before contacting support

Conventions Used in This Book

This book uses several conventions for special terms and actions, operating system-dependent commands and paths, and margin graphics.

Typeface Conventions

The following typeface conventions are used in this book:

Bold Lowercase and mixed-case commands, command options, and flags that appear within text appear like **this,** in **bold** type.

Graphical user interface elements (except for titles of windows and dialogs) and names of keys also appear like **this,** in **bold** type.

Italic Variables, values you must provide, new terms, and words and phrases that are emphasized appear like *this,* in *italic* type.

`Monospace` Commands, command options, and flags that appear on a separate line, code examples, output, and message text appear like `this`, in `monospace` type.

Names of files and directories, text strings you must type, when they appear within text, names of Java methods and classes, and HTML and XML tags also appear like `this`, in `monospace` type.

Operating System-dependent Variables and Paths

This book uses the UNIX® convention for specifying environment variables and for directory notation.

When using the Windows command line, replace **$***variable* with **%***variable***%** for environment variables and replace each forward slash (**/**) with a backslash (\) in directory paths.

Note: If you are using the bash shell on a Windows system, you can use the UNIX conventions.

1

Introduction

This chapter provides a general overview of the IBM ® Tivoli® Monitoring Workbench 5.1 (hereafter also referred to as the workbench) and its integration with IBM Tivoli Monitoring 5.1 (hereafter also referred to as Tivoli Monitoring).

The workbench provides an integrated environment for developing and modifying resource models for Tivoli Monitoring. The workbench produces a complete package that can be installed on the Tivoli Monitoring server through the command line.

What Workbench Can Do for You

The workbench is a programming tool for creating, modifying, debugging and packaging resource models. Resource models are used at an endpoint to collect and analyze data regarding the state and performance of different resources, such as disks, memory, CPU, or applications. After you have created a resource model, you install it on your Tivoli Monitoring server and include it in a profile.

Resource models created with former versions of the product can be reused with Tivoli Monitoring 5.1 (although resource models created for UNIX endpoints will need to be redistributed), while resource models created with Tivoli Monitoring 5.1 are not compatible with former versions of the product.

Tivoli Monitoring provides a set of default resource models. You can use the workbench to expand the collection of the available resource models according to the requirements of your system.

When you create a resource model you specify sets of values, for example:

- Thresholds you do not want your resource to exceed
- Ranges of instances within which to limit the data collection
- Events you want your system to generate if the monitored resource state is not satisfactory.

When you create a resource model you can also specify that the monitoring results must be sent to the Tivoli Enterprise Console® server, or to Tivoli Business Systems Manager® In this way you can perform a more complete management of all your resources.

You enter all these settings in the displayed dialogs and they are automatically included in the decision tree script. The script contains the settings you have entered in the dialogs, and the monitoring algorithm that governs the whole process. The monitoring algorithm is developed using either Visual Basic or Java™ Script and specifies how to use each setting to create the process more suitable to your needs.

The following figure illustrates the whole process and the connection between the workbench and Tivoli Monitoring:

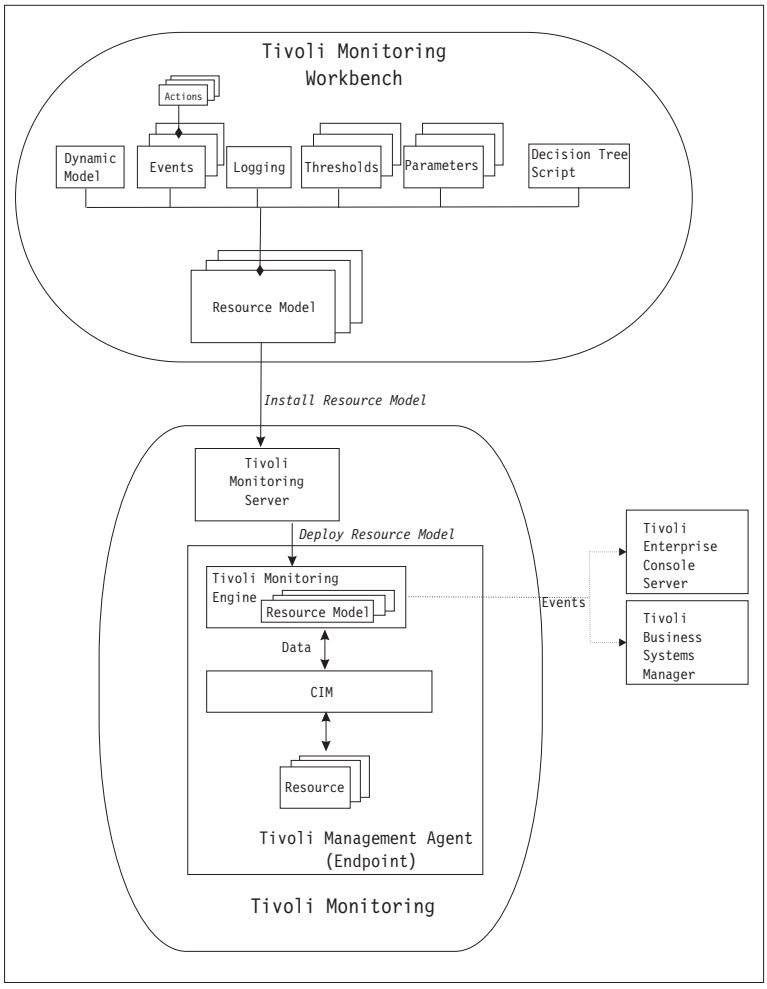


Figure 1. Overview of the whole process

Compatibility Mode

Starting with IBM Tivoli Monitoring 5.1, a compatibility mode is provided to allow Tivoli Monitoring users to use Tivoli Distributed Monitoring (Classic Edition) monitoring sources and custom scripts inside a Tivoli Monitoring resource model. This is achieved through the implementation of new procedures to be run on the workbench.

To collect data about system resources, Tivoli Monitoring relies on a Common Information Model (CIM) Object Manager implementation (WMI on Windows® or a Tivoli Monitoring endpoint engine on the other supported platforms). With the compatibility mode, Tivoli Monitoring can collect data not only from CIM data sources but also from Tivoli Distributed Monitoring (Classic Edition) monitoring sources.

The compatibility mode provides an easy way to import Tivoli Distributed Monitoring (Classic Edition) monitoring sources into a resource model without the user having to write any additional code. This mode is implemented through a set of procedures that import Tivoli Distributed Monitoring (Classic Edition) monitoring sources.

Common Information Model Implementation

Tivoli Monitoring implements the Common Information Model (CIM) standard from the Distributed Management Task Force (DMTF). CIM is a model for describing management information in a network environment. The model applies the basic techniques of the object-oriented paradigm.

To obtain data from the monitored resources, Tivoli Monitoring may use processes included in the endpoint's operating system. On Windows systems it uses the Windows Management Instrumentation (WMI), which is Microsoft's implementation of CIM. On UNIX® and Linux platforms the information collection agent is incorporated in the Tivoli Monitoring endpoint engine based on CIM specifications.

The following picture shows the architecture of the Tivoli Monitoring endpoint engine.

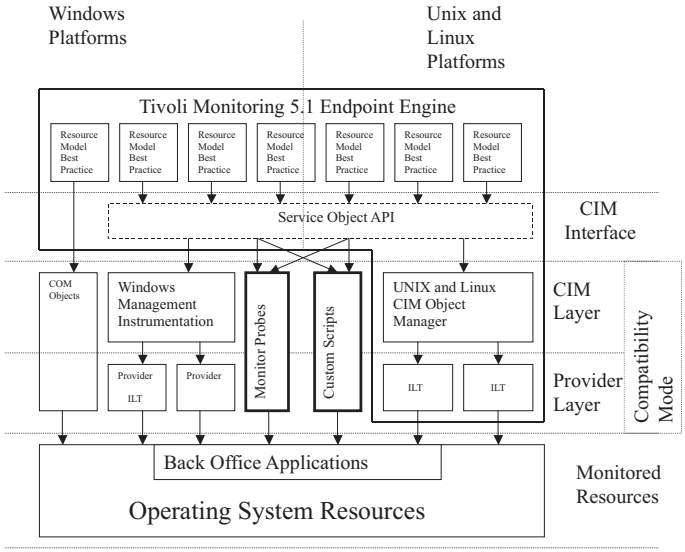


Figure 2. Architecture of the Tivoli Monitoring endpoint engine

The service object API enables the resource model scripts to use CIM objects, monitoring collections, and custom scripts in any combination. Specifically to Windows and UNIX/Linux environments, the WMI and CIM Object Manager (CIMOM) respectively are responsible for loading the providers that, in turn, get performance and availability data from system and application resources.

At the provider layer, an Instrumentation Library Type (ILT) interface is provided, using a parameters passing mechanism. The CIM provider is implemented by an ILT Manager for Java™. On Windows platforms, the ILT Manager for Java is a WMI provider DLL.

Refer to Appendix D, "Instrumentation Library Type Interface" on page 103 for details on the ILT interface available with Tivoli Monitoring and the guidelines to use it.

2

Resource Models and Related Concepts

This chapter provides a description of resource models and their elements, and it illustrates some general concepts that will help you to use the workbench in the Tivoli Monitoring environment.

Defining Resource Model Elements

Resource models represent the core of the workbench. In a resource model, you can specify the resources you want to monitor at runtime, the resource data to be collected, and the monitoring algorithm to gather and analyze the data.

Resource models contain the following elements:

- Dynamic model
- Events
- Thresholds
- Parameters
- Logging
- Decision tree script
- Dependencies

Cycles

When a resource model is run at an endpoint, it gathers data at regular intervals, known as cycles. The duration of a cycle is the *cycle time*. At each cycle the resource model collects the required data, analyzes it, generates the events and triggers whatever actions are required. Cycle times, which are expressed in seconds, are specified by the user when the resource model is defined.

Dynamic Models

The dynamic model is considered the essential element of a resource model. In the dynamic model is specified:

- Which resources are going to be monitored
- The properties of each resource
- The mechanism used for collecting the data

A dynamic model contains a set of predefined Common Information Model (CIM) classes that describe the status of each resource. A class is a list of properties relevant to a specific resource. The workbench shows the list of available CIM classes contained in the CIM

repository of WMI. The instances of the classes are provided by WMI providers. If the resource you need is not described by any class contained in CIM repository, you can add it, together with its associated provider.

When you define a dynamic model, you must specify a CIM class and its properties for the specific resource. For example, if you are monitoring your CPU usage, you can consider the process class with all its instances. You can use as class properties CPU usage for process, process name, and process ID.

Data Collection

Resource models collect data in two modes: synchronous and asynchronous. In addition, it is possible to filter the collected data and to sort it in different orders.

Synchronous Data Collection

Synchronous collection means that the data is routed to the resource model as soon as it is collected. By default, at each cycle, data is collected, routed to the resource model and analyzed. If needed, data collection may span multiple cycles.

Data Sorting

With synchronous data collection, data can be sorted in either ascending (from the lowest to the highest) or descending (from the highest to the lowest) order. For an example of data sorting, see “Defining a Filter and a Sort Order” on page 20. When data needs to be sorted, a key element has to be specified to rule the sorting.

Asynchronous Data Collection

Asynchronous collection is an across-cycle collection. This means that data collection starts within one cycle and continues into the following cycle, when the data is made available. Between two cycles, the collected data is stored in cache. Typically, you use an asynchronous collection when monitoring asynchronous resources, like the Windows NT® EventLog. In this case, collecting data synchronously might unnecessarily impact your system performance.

Asynchronous collection is available for Windows resource models only. It is not supported by UNIX resource models.

Data Filtering

You can filter the monitored data by writing a WQL query, asking the resource model to collect only those instances that satisfy certain conditions.

Filtering is available for Windows resource models, only. It is not supported by UNIX resource models.

Example: You can specify that you want to collect only process instances where CPU usage is greater than 60%. In this case, you can write a WQL WHERE clause that reads:

```
"where PercentProcessorTime > 60"
```

The value expressed in this query is fixed and cannot be changed from Tivoli Monitoring dialogs.

Note: WQL is a query language used by WMI. For more information see the WMI Web site.

Events

Generally speaking, an event is a change in the status of a resource. In a Tivoli Monitoring environment, an event notifies the system administrator about the state of a specific resource.

Events and Indications

In the workbench a distinction is made between indications and events. An indication is generated when the state of a given resource meets defined criteria. By itself, an indication does not trigger any specific action. When indications are aggregated they become an event.

When you define an event, you must specify under what conditions a certain number of indications are aggregated into an *event*. You also specify whether these indications must be consecutive, or whether the sequence may be interrupted by one or more monitoring cycles that do not register any indication. The cycles during which no indication is generated are called *holes*.

Only events can notify that there is a problem in the resource state, trigger an action and, if enabled, send a notification to the Tivoli Enterprise Console server or to the Tivoli Business Systems Manager.

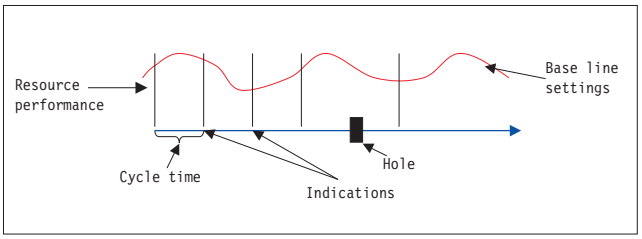


Figure 3. Holes and indications

Attributes

In the workbench it is possible to qualify an event by specifying attributes relevant to it. The attributes can be both string and numeric values, and can be chosen according to the information that must be collected by the resource model.

Example: An event may indicate that the disk space is not sufficient. By specifying attributes, such as disk name, or available disk space, a more precise indication of the problem can be generated.

Actions

With the workbench, it is possible to associate one or more recovery actions with a specific event. These actions are automatically triggered when the event occurs and, typically, are used for restoring satisfactory system service level.

Each time a given event occurs, the system provides a notification of the event, triggers a recovery action to restore satisfactory conditions and, if the action is successful, provides a notification that the action was performed.

Actions are associated with either the execution of a CIM method or the execution of a program.

Example: If you are monitoring Windows NT services and an event is generated when a service stops, you can associate the event with a recovery action that restarts the interrupted service.

CIM Methods

The workbench can associate events with CIM class methods, which can be invoked to restore a satisfactory resource state. In CIM, each class has attributes that characterize the object and a set of methods that are actions related to that object. Actions invoke *methods* that are provided by WMI. An action can invoke one method at a time.

When you select a class within the workbench, the related methods that are defined by WMI are automatically displayed.

A method can be *static* or *non static*. A static method is defined with respect to a class, a non-static method is defined with respect to an instance.

Programs

The workbench can associate an event with the execution of a program. In this case, the user can also indicate a set of arguments (for example environment variables) that must be associated with the execution of the program.

Thresholds

Each resource model defines one or more *thresholds*. A threshold is a numeric value that is specified in the customization phase and that is used according to the monitoring algorithm written in the script. The threshold value specified in the workbench dialogs is a default one, but it can be changed from within the Tivoli Monitoring dialogs when a profile is defined. Typically, this value represent a limit for a satisfactory resource state. If the monitored resource exceeds this limit, an indication is generated.

Example: If you are monitoring your disk space and you do not want it to drop under 70%, you can consider 70 your threshold, so that the system generates an indication each time your disk space is less than 70%.

Example: You are monitoring your CPU usage and are collecting data on the CPU used by each process; if you are interested only in the 10 most consuming processes, your threshold may be 10. You can specify the default value in the Thresholds dialog, then add the following instructions in the Init function of the decision tree script. For more information about the decision tree script, see “Decision Tree Script” on page 12.

```
"where PercentProcessorTime >"
+Str(Svc.GetThreshold("HighCpuThresh")
```

For each threshold a description can be added to explain how each value is used within the monitoring algorithm, and the logic and meaning behind the specific value. For more details on the use of thresholds, see “Thresholds Dialogs” on page 84.

Parameters

While thresholds can be only numeric values, parameters can be lists of numbers or strings. Using parameters enables you to customize your resource model. You can define different parameters, as required. For each parameter you can specify a value, which can be a list of numbers or strings. This list can represent the instances you want to monitor, or a limit you do not want your resource to exceed, depending on how you use this setting in your script. The lists will then be displayed in the Tivoli Monitoring dialogs, and depending on the type of list you select in the workbench, the lists will appear in different forms on Tivoli

Monitoring. Within the workbench you can define default values and then let the operator customize the settings in Tivoli Monitoring. When you specify the parameters, you can choose one among the following kinds of lists:

- Boolean List
- Choice List
- String List
- Numeric List

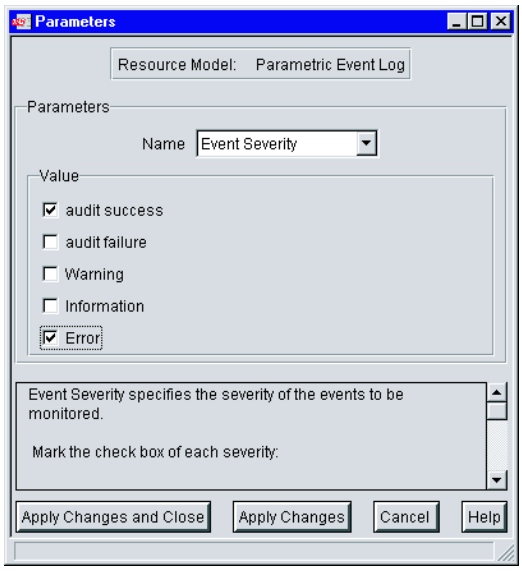
For more details on the use of parameters, see “Parameters Dialogs” on page 88.

Boolean List

Boolean lists enable you to specify a list of values, and then to decide which to consider (select them as TRUE) and which to ignore (select them as FALSE) during monitoring.

Example: If you are monitoring Windows EventLog but you want to monitor only two event types out of five available, in the workbench you can enter the names of all the event types and select them as TRUE or FALSE, accordingly.

When you use a boolean list in the workbench, the output you get in Tivoli Monitoring dialogs is a check box list, as shown in the following figure.



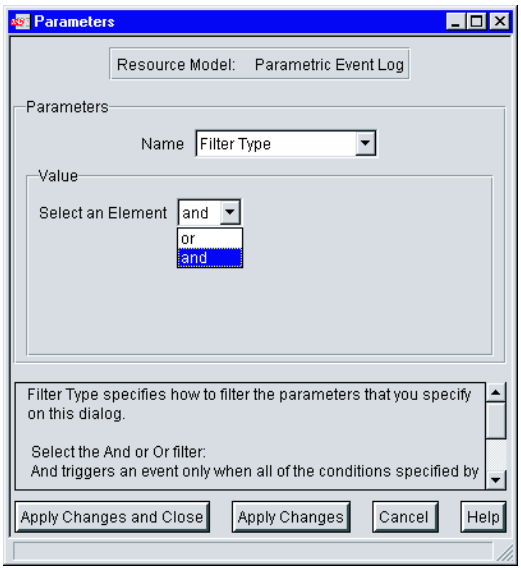
The values selected as TRUE in the workbench, appear as checked options in Tivoli Monitoring dialog, the values selected as FALSE in the workbench, appear as clear options in Tivoli Monitoring.

Choice List

Choice lists are lists of mutually exclusive values. These lists can contain several values, but the Tivoli Monitoring operator can select and use only one of them at a time.

Example: You can provide your resource model with two types of filters (AND / OR). In the choice list on the workbench you can insert both filter types, but the Tivoli Monitoring operator will select only one of them at a time.

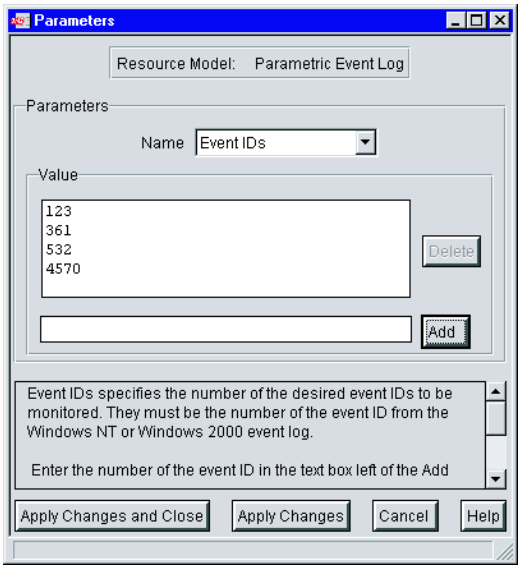
Choice lists in the workbench are represented in Tivoli Monitoring dialogs as drop down lists, as shown in the following figure.



Here, if the operator selects **And**, an event will be generated when all of the conditions specified are met. If the operator selects **Or**, an event will be generated when any of the conditions specified is met.

Numeric List

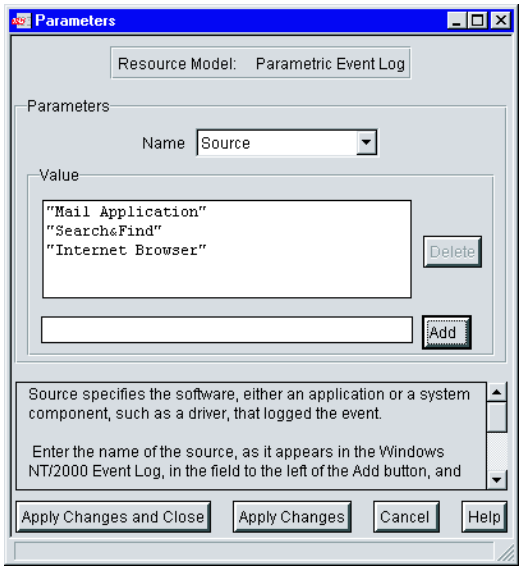
Numeric lists are lists of numeric values, such as event IDs. When you use a numeric list in the workbench, the output you get in Tivoli Monitoring dialogs is a simple list, as shown in the following figure.



String List

String lists are lists of string values, such as application names. When you use a string list in the workbench, the output you get in Tivoli Monitoring dialogs is a simple list, as shown in

the following figure.



Note: Within the workbench, regardless the type of list you choose, the list content is displayed in the script either as a numeric or a string list.

Logging

This function allows you to store data regarding the attributes of a resource. Data is stored in a local database and can be accessed using Tivoli Monitoring Web Health Console. For more information, see *Tivoli Monitoring User's Guide*. Within the workbench you specify what and how to log, however logging is disabled by default in Tivoli Monitoring. Tivoli Monitoring operator has to enable it selecting a specific option.

Logging Elements

The logging function contains the following elements:

Context
A general problem to which the resource activity relates. An example of context can be disk space.

Resource
The resource whose state you want to log, in relation to the defined context. An example of resource can be a logical disk. A single context can contain more than one resource.

Properties
Specific attributes of the defined resource. An example of properties of the resource logical disk can be free space. For each resource you can specify multiple properties, which can be both numeric and string values. Among the specified properties, you must define key properties that clearly identify the instance of a resource.

The database browser of the Tivoli Monitoring Web Health Console shows historical logged data on a specific endpoint and for a specific resource model. With logged data, you can use the database browser to identify specific instances of resource problems over the past 6, 12, or 24 hours.

For more information, see Tivoli Monitoring Web Health Console.

Decision Tree Script

The workbench automatically generates the decision tree script part that contains the information you have specified when you created and configured the resource model. If you change one of the settings in the workbench dialogs, the corresponding data is automatically updated in the script. You cannot modify these settings directly in the script. The script also contains the algorithm that you write in Visual Basic or Java Script to govern the whole process. The decision tree script contains three basic functions and one subroutine. By default, they produce a return value equal to 0, however, you can specify different codes to be returned under specific circumstances. For more information see Chapter 6, Chapter 6, "Resource Model Troubleshooting" on page 55. The default functions are the following:

Main A subroutine used by workbench for debugging the resource model. Tivoli Monitoring monitoring engine does not call it. This subroutine performs the following actions:

- Creates the TMWService.Utils.
- Calls the SetDefaultConfiguration function.
- Calls the Init function.
- Enters the monitoring loop (collect data, visit the decision tree script, and wait cycle time).

Note: Do not modify this subroutine. If you do, unpredictable results may occur.

SetDefaultConfiguration
Initializes the object on the basis of the settings defined in the Events, Thresholds, Parameters, and Actions dialogs. If you change one of the settings contained in these dialogs, the corresponding data is updated in this function. On the contrary, you cannot change those settings modifying them directly in this function. The SetDefaultConfiguration function is called just once, when the resource model is started. Therefore, if necessary, you can write additional initialization code at the end of this function.

Init Called after that the settings defined in the SetDefaultConfiguration function have been replaced with the values coming from the profiles of Tivoli Monitoring. At this point the settings are replaced with the new ones.

Visit Tree
Contains the monitoring algorithm and is called at the beginning of each cycle. You have to write the monitoring algorithm in Visual Basic or Java Script, and define how to use all the values and variables previously set. This function checks the algorithm and implements it. It processes the collected data according to thresholds and parameters settings, and, if necessary, sends an event. It is also possible to define the algorithm in such a way that the execution of the data collection is entirely controlled by the algorithm itself (see the details about the **Collect on demand** checkbox in "Task 2: Defining a Dynamic Model" on page 27) . As an example, a specific collection could be executed based upon the results of a previous collection.

Dependencies

To run a resource model on an endpoint, you may need to transfer additional files. For example, to add a new class to the CIM repository, you need the files that define the class (MOF) and the provider (DLL). A MOF file is a text file that contains definitions of classes and instances using the Managed Object Format (MOF) language.

In these cases you can add dependencies to your resource model, and transfer the required files with the model.

3

Installation

Before you install the product, be sure that you have satisfied all the prerequisites listed below.

Software Requirements

To install the workbench, you must have either of the following software installed and running:

- Windows NT 4.0 Service Pack 5 or higher, plus WMI Version 1.1 or higher (recommended 1.5)
- Windows 2000
- Windows XP Professional Code
- In order to debug Java Script resource models for Windows on Windows systems, you need a Java Script debugger. You can download a Microsoft debugger from this URL:
  http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28001169

Hardware Requirements

The minimum hardware requirements for the workbench are listed in the following table.

Hardware	Minimum Required
Disk space	10 MB
RAM	64 MB
Processor	133 MHz

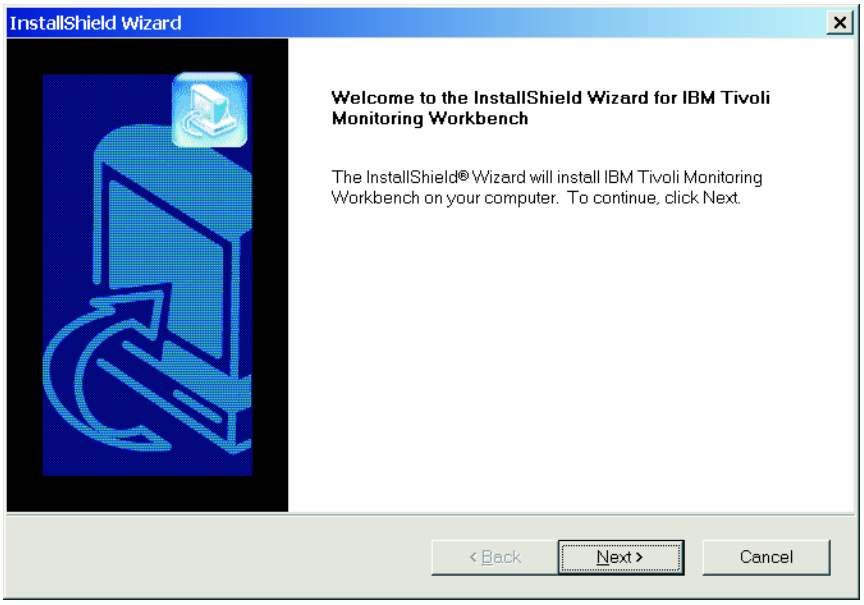
Installation Procedure

If you are running a previous version of the workbench, you need to uninstall it and install Tivoli Monitoring Workbench 5.1. All resource models created with a previous version of the product are supported by Tivoli Monitoring Workbench 5.1.

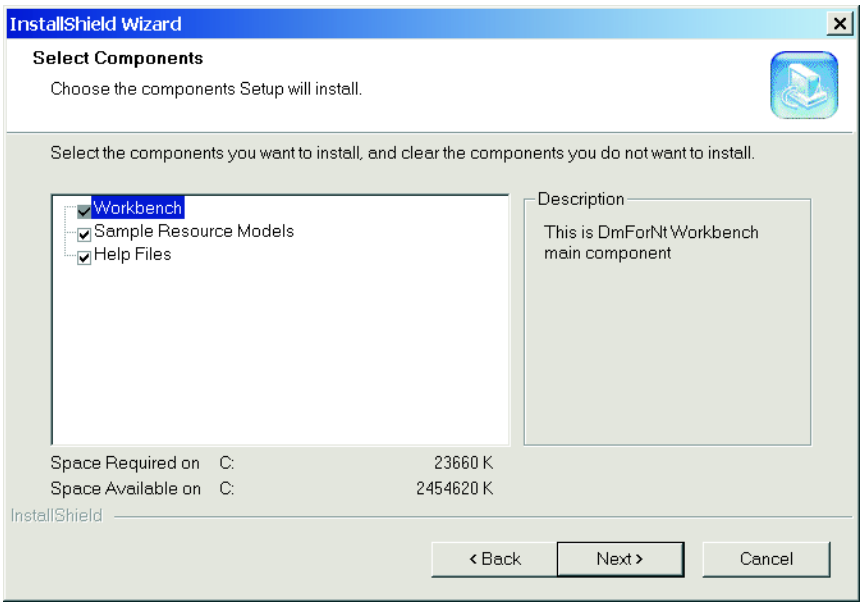
To install the workbench, perform the following steps:

1. Open the **ITM5.1WB** folder and browse to the **Disk 1** folder.
2. From **Disk 1**, double click the **Setup.exe** file.

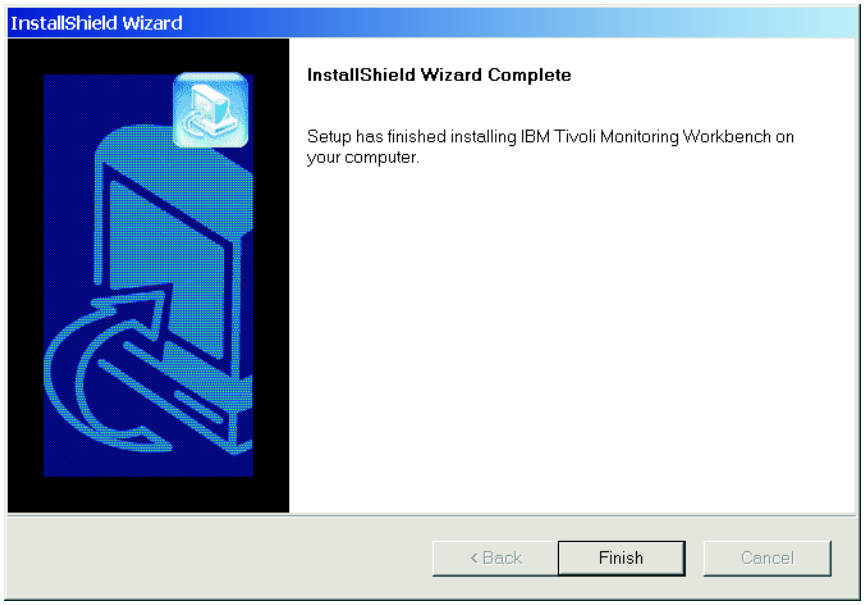
The following dialog opens:



3. From the InstallShield Wizard dialog click **Next** to proceed with the installation.
4. In the next InstallShield Wizard dialog, select the components you wish to install. For a full installation, selects all components. Click **Next** to proceed with the installation.



5. The installation runs. Once the execution is complete, you see the InstallShield Wizard Complete panel.



6. Click **Finish** to close the installation procedure.
   Now you have successfully installed the workbench.

4

Designing a Resource Model

This chapter, provides generic advices about designing resource models.

Before you create a resource model it is recommended that you have a clear idea of your objectives and prerequisites. Only at that point you should use the workbench to create the resource model.

Preliminary Steps

You need to create an overall design, considering all the steps you are going to take with your resource model. To do so, you determine the objectives and functions of your resource model, and draw some general specifications. The following tasks will help you design your resource model:

1. Gathering all the information necessary to perform an accurate problem analysis.
2. Determining which resources you want to monitor.
3. Defining the resource model you are going to create.
4. Defining which problems you want your resource model to highlight.
5. Defining a list of thresholds and parameters to use as external inputs.
6. Defining the monitoring algorithm (draw a simple flow chart of the decision tree script.)

Once you have executed these tasks, you can proceed with creating the resource model.

Example of a Resource Model

A server is configured with one processor, and we want to find out whether the CPU has enough capacity. In this case, the objective of the resource model is to monitor CPU usage.

Problem Analysis

First of all, we have to determine what data we need to collect for a proper assessment of our CPU usage. In this case, we want to gather some information about our processor capacity and about the CPU capacity used by the processes we run most frequently.

Determining Which Resources to Monitor

We now consider what kind of resource monitoring can provide us with the required information. Because resource models access data provided by WMI, we have to find out whether the resource we want to monitor is defined by WMI. In our example, we need the following data for the processor and its processes:

Processor

- Current CPU Usage

- Processor Type

Processes

- Process Name
- Process ID
- Process CPU Usage

In this case, all this data is made available by WMI.

Defining a Resource Model

Next, we must determine more in detail the objectives of our monitoring and the attributes required to meet the objectives. Many of the values that we set here are default values that can be changed later in the Tivoli Monitoring profiles. However, whenever possible, the default values should be appropriate.

Defining a Cycle Time

Starting from the type of resources that we are going to check, we can determine the cycle time of our monitoring. A thorough analysis of the resource allows us to obtain all the required information, reducing to the minimum the impact of the monitoring activity on our system performance. There are two main factors to consider: how fast a given resource changes its status, and the fact that, by running a resource model, we increase the overhead on our system. In this example, given that the resources we are considering change their status quite frequently, a cycle time of 10 seconds is a reasonable compromise between monitoring status variations and saving system performance.

Defining a Filter and a Sort Order

In this case, we are not interested in processes with low CPU consumption because they do not affect our CPU performance too severely. Thus, we can apply a filter to our data collection, so that we collect data on CPU usage only for those processes that use at least 20% of CPU capacity. Also, we can sort the obtained results in ascending order of CPU usage. So, if the most consuming process does not exceed the 20% threshold, the other processes will not be monitored at all: by acting this way we can save system performance.

Defining the Highlighted Problems

After determining the kind of monitoring we want to perform, we have to identify the possible problems we want to detect, and we must give a name to the event generated by each problem. Following our example, we want to identify the following problems:

- Excessive CPU usage
- Excessively consuming process

For each of the above problems, we want an event to be generated after persistent indication of the problem. How persistent an indication must be, before it is aggregated into an event? We define that 10 consecutive occurrences of either problem should generate an event. Now we need to determine what attributes may qualify each event more in detail. If we define adequate attributes, we can obtain useful information regarding the presence of a problem as well as its possible causes. For each event, the following information may be of interest:

Event: Excessive CPU usage	Event: Excessively consuming process
Current CPU usage	Process name
Most consuming process name	Process ID
Most consuming process ID	Process CPU usage

Event: Excessive CPU usage	Event: Excessively consuming process
Most consuming process usage	
Processor type	

To understand how attributes can help us, let's see what conclusion we can derive, assuming that the following results have been detected for the following event:

Excessive CPU Usage:	
Current CPU usage	90%
Most consuming process name	Mail Application
Most consuming process ID	443
Most consuming process usage	10%
Processor type	Pentium 133¬

Given this data, it does not seem that our most consuming process is too consuming, therefore it is not likely to be the cause of our problems. Consequently, our CPU performance must be low because of several concurrent processes. Considering the type of processor we are using, perhaps the system load is too heavy and we need a more powerful processor.

After specifying each event with its attributes, we must determine which is the key attribute among them, if any. A key attribute is the most representative one, and clearly identifies the event it refers to. For example, for the event concerning the excessive CPU usage, there is no key attribute, while for the event concerning the excessively consuming process, the key attributes can be the process name and the process ID.

Defining Thresholds and Parameters

For each of the events, we can then specify some values that govern the data collection. These values are thresholds (only numbers) and parameters (lists of numbers and strings). We do not necessarily need to use both, because sometimes, when we define one, we do not need the other. In our example we use thresholds and will not need parameters, however, in other cases, it may be useful to use both. The values we define here can be used in different ways in the script, therefore now we must know which values we need and how to use them. For our example, the following thresholds may be adequate:

Thresholds	Default Values
Maximum CPU usage	90%
Excessively consuming process	25%
Processes considered	10

These values mean that we want the system to generate an indication each time our current CPU usage is higher than 90%, and each time a single process uses more than 25% CPU. Furthermore, in our data collection, we do not want to consider all running processes, but only the 10 most consuming ones. To help the user of this resource model, we must explain the logic behind these values in the description of each threshold, and give each value its function when we specify it in the decision tree script. In this example, considering our objectives, we don't need to set any values for the parameters. For an example about the usage of parameters, see "Parametric Event Log" on page 86.

Defining the Monitoring Algorithm

After we have defined our resource model, we must determine the monitoring algorithm. We can start by drawing a flow-chart that shows how the algorithm works. The flow-chart must reflect what we will write in our script: in other words, it must describe how all the components interact with each other and what final result we will get. According to our example, the flowchart can be the following:

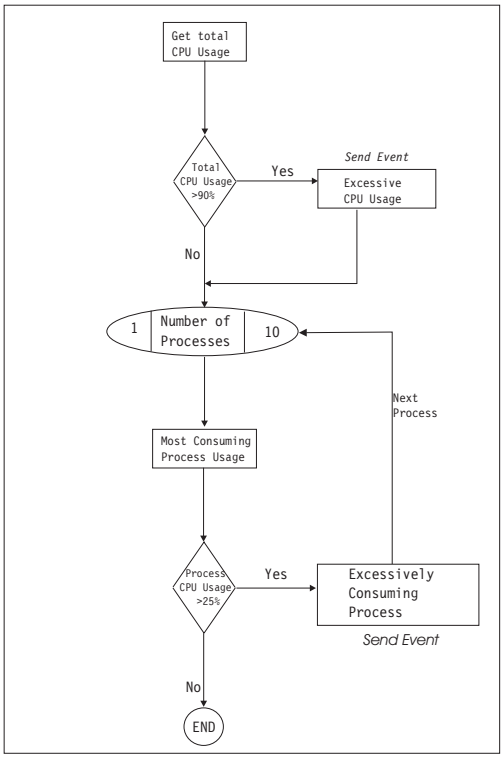


Figure 4. Flowchart showing the script process

Note: In this example processes are sorted in a descending order with respect to CPU usage. Therefore, when the loop meets the first process that does not exceed the threshold, it ends without considering any of the following processes.

5

Creating a Resource Model

This chapter describes the procedures to create, debug and build a new resource model.

It also explains how to import Tivoli Distributed Monitoring (Classic Edition) monitoring sources or a custom script into a resource model. Only synchronous monitors can be imported.

The workbench provides a wizard for creating new resource models. Basically, the wizard, starting from a selected CIM class taken from the WMI repository, or starting from a monitoring collection, or from a custom script, displays a sequence of dialogs with default values already filled in to drive the user to create simple resource models. At the end of the wizard process all the needed Visual Basic or JavaScript code is automatically generated.

About Resource Model Names

Resource models and their components have the following names and descriptions that are used in different contexts; for example, names used in scripts are different from those displayed in the Tivoli Monitoring dialogs.

Note: The workbench does not support double-byte characters, because the decision tree script does not handle them. It is recommended that text information and names are entered in English, so that the resource model can be run appropriately from the Tivoli Monitoring application. If you write text information in a language different from English, and then export the resource model and install it on the server, all text information will be handled altogether, regardless whether it is in English or not. So, it is recommended that you install translated message catalogs separately. For more information, see "Exporting the Message Catalog" on page 40.

Internal Name
The names of your resource models, thresholds, parameters and events as you call them in the decision tree script. These names are used inside a monitoring algorithm written in Visual Basic or Java Script: as such, they must be alphanumeric, start with an alphabetic letter, and contain no blanks.

Descriptive Name
The names of your resource models, thresholds, parameters, events and actions as they appear in the Tivoli Monitoring dialogs and in the message catalog. These names should be more meaningful to you than the internal names. As opposed to internal names, they are not subject to naming conventions.

Description Some text describing your resource models, thresholds, parameters, events and actions. Here you can specify what the specific element does and the logic on which it is based. For thresholds and parameters, you can explain in

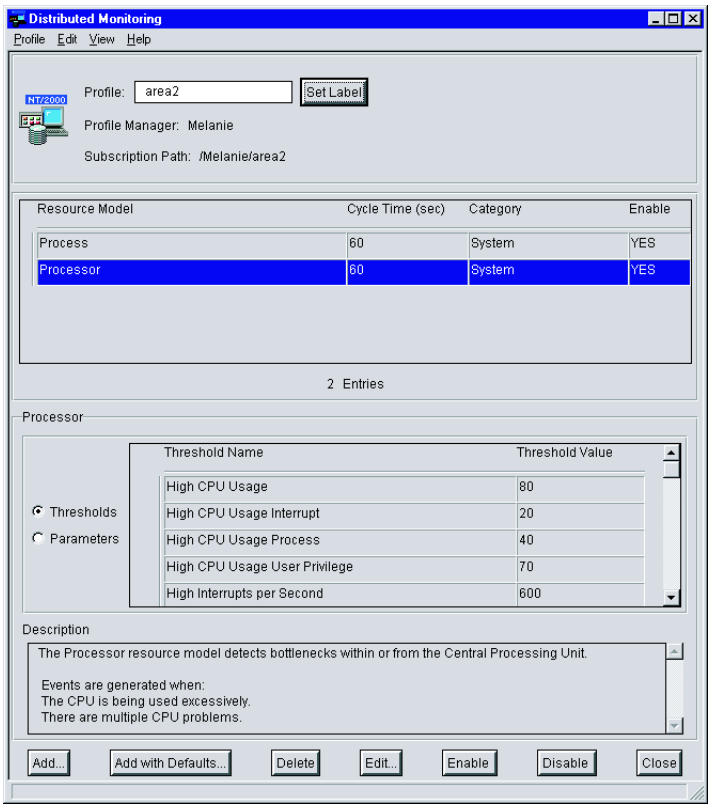
this text box how you intend to use the values specified inside the decision tree script. For example, you can include some information about the units of measurement of the inserted values and the range within which the values may vary.

Alias
A name for the selected class of a dynamic model. This name is used in the decision tree script. It is recommended that you use a meaningful but simple alias. If you do not type an alias, the class name called in the decision tree script is the Selected Class name with the whole path.

Category Internal Name
A name for the resource model. This name appears in internal databases and is used to define groups that include similar resource models.

Category Descriptive Name
A meaningful name for the resource model. This name appears in the Tivoli Monitoring dialogs when the operator opens a profile containing that resource model. It appears also in the message catalog. This name is used to define groups that include similar resource models.

The following figure shows an example of how these names and descriptions appear in the Tivoli Monitoring dialogs:

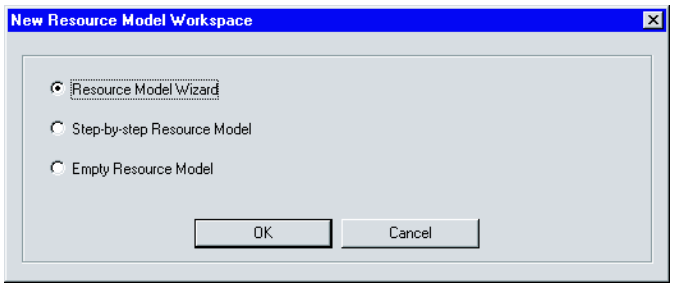


Resource Model Creation

When you create a resource model, you can choose to code the decision tree script in Visual Basic or Java Script. Visual Basic code can be used to create Windows resource models only. Java Script code can be used to create resource models for Windows and UNIX platforms.

The first displayed dialog requests to select the language that will be used to create the decision tree script.

After selecting the language, you can choose to create a resource model according to one of the different procedures shown in the following dialog:



If you select the **Resource Model Wizard** option, a series of dialogs guides you through the whole process of creating a simplified resource model without writing any scripts or code. The resource model produced in this way is quite simple and provides a limited range of possibilities. However, once is created, you can modify and customize it by editing the decision tree script and the information contained in the GUIs, just as you would do with any other resource model.

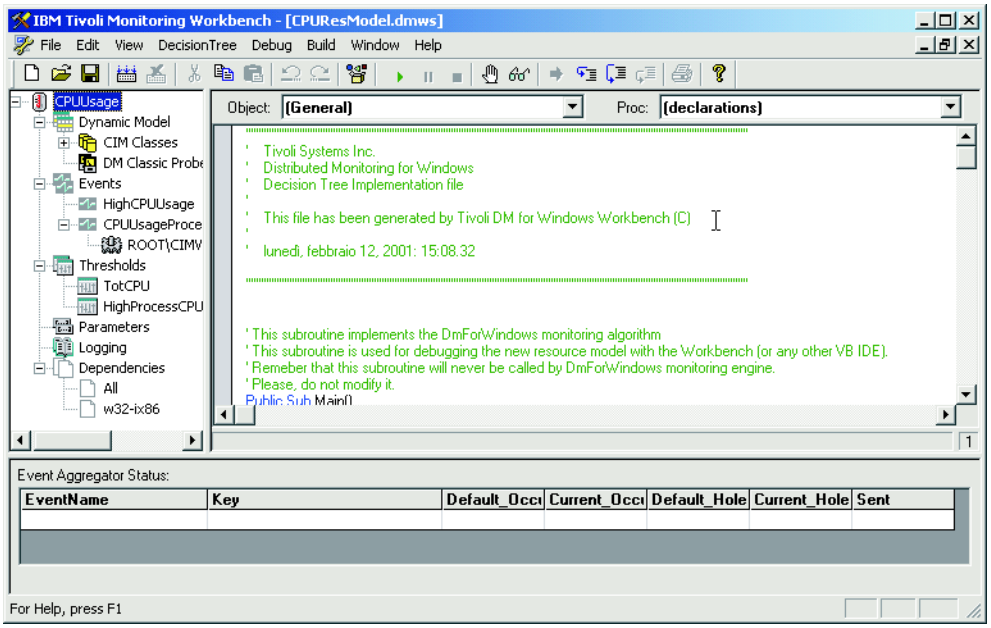
The **Resource Model Wizard** option also enables you to import a Tivoli Distributed Monitoring (Classic Edition) monitoring collection or custom script into a resource model.

To create a resource model using the wizard, see “Creating a Resource Model from a CIM class” on page 43.

If you select the **Step-by-step Resource Model** option, dialogs guide you through the procedure in the sequence of steps shown in Chapter 4, “Designing a Resource Model” on page 19. However, when you have completed this procedure, you still have to write the script that governs the resource model.

If you select the **Empty Resource Model** option, you can define the resource model elements without following a specific sequence. The definition is done by opening the resource model tree structure and double-clicking on the required element, as shown on the

left pane of the following figure.



After entering the required information in the dialogs that show for each element, you still have to write the script that governs the resource model.

Creating an Empty Resource Model

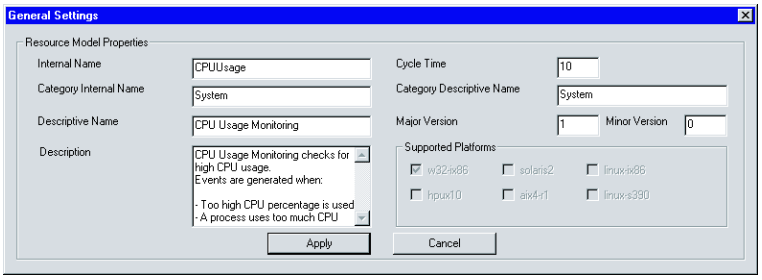
To create a new resource model following the standard menus, you have to perform the following series of tasks, but not necessarily in the order given. Also, you can skip those tasks for which your resource model does not include the described elements. For more information about resource models, see “Defining Resource Model Elements” on page 5.

The following task sequence applies to Windows systems. If you are creating a resource model for UNIX platforms some steps may be slightly different. Where necessary, these differences are pointed out.

Task 1: Defining a Resource Model

To define an empty resource model, perform the following steps.

1. From the New Resource Model Workspace, select **Empty Resource Model** and click **OK**. The General Settings dialog opens.



2. In the **Internal Name** text box, type a name for your resource model. For more information about naming conventions, see “About Resource Model Names” on page 23.

3. In the **Cycle Time** text box, type a value for the time interval (in seconds) between two successive data collections. This is the default value that is displayed and can be modified in the Tivoli Monitoring dialogs when the operator opens this resource model.
4. In the **Category Internal Name** text box, type a name for this resource model. This name will be used by internal databases to define groups that include similar resource models.
5. In the **Category Descriptive Name** text box, type a name for this resource model. This name will appear in the Tivoli Monitoring dialogs and in the message catalog.
6. In the **Descriptive Name** text box, type a meaningful name for your resource model.
7. In the **Major Version** text box, type the number you want to indicate the major version of your resource model.
8. In the **Minor Version** text box, type the number you want to indicate the minor version of your resource model.
9. In the **Description** text box, type a description of your resource model.
10. In the **Supported Platforms** box, select the platforms you want to support this resource model. The available options are enabled only if you are creating a resource model in Java Script.
11. Click **Apply** to create your resource model.

Although you have created the new resource model, this model is not complete; you still have to define the resource model components.

Task 2: Defining a Dynamic Model

The dynamic model is a necessary element of a resource model. Here you specify the resources you want to monitor and how to collect the data.

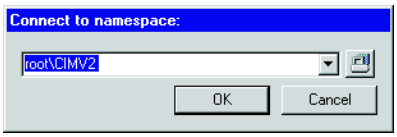
When you define a dynamic model you have the option to either select CIM classes for the resources you want to monitor, or to import a monitoring source (Distributed Monitoring Classic Probe). You can as well select both options at the same time. The steps below show the definition of a dynamic model through CIM classes, refer to “Creating a Resource Model from a Monitoring Source” on page 40 if you wish to import a monitoring source.

The available CIM classes are stored in the CIM repository of WMI. The instances of the classes are provided by WMI providers. If the resource that needs to be monitored is not described by any class contained in the CIM repository, then a new class can be added. To add a new class to the CIM repository, a Managed Object Format (MOF) text file that defines the class has to be written along with the provider (DLL). The MOF file must be installed on the WMI and, in order to run the resource model on an endpoint, the MOF file and the provider must also be transferred to the endpoint: this is done by adding a dependency to the resource model.

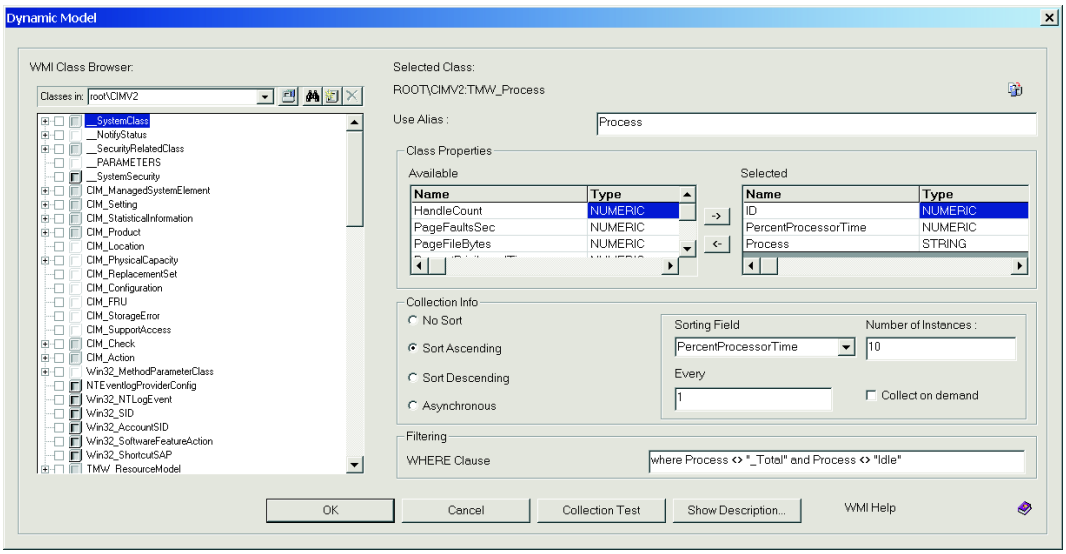
To define a dynamic model though CIM classes, perform the following steps:

1. From your resource model, double-click **Dynamic Model**.
2. Double-click **CIM Classes**.
3. In the Connect to namespace dialog, specify the required namespace and click **OK**. The default namespace is root\CIMV2. Use the browse button to locate another namespace, or to connect to a remote workstation. Connecting to a remote workstation enables you to browse and select one of the CIM classes of that workstation and run a data collection test remotely.

When you connect to a namespace, you are connecting to WMI and accessing all the available classes.



4. In the WMI DM Workbench Login dialog, type your user name and password, if necessary.
5. Click **OK**. The Dynamic Model dialog opens.



6. To change the namespace, select one from the **Classes in** drop-down list, or use the browse button to locate one.
7. From the class list, select the required class. The corresponding properties are displayed in the **Class Properties** group.
8. In the **Class Properties** group, from the **Available** box, select the properties you want to list in your collection and move them in the **Selected** box.
9. In the **Use Alias** text box, type a simple alias for your class name to use in your script.
10. In the **Collection Info** group, select the required sort option for the collected data:

No Sort Implies no sequence order.

Sort Ascending
Sorts from the lowest value to the highest.

a. In the **Sorting Field** text box, type the key element for the sort.

b. For Windows resource models only: in the **Number of Instances** text box, type the number of instances to be in your data collection.

Sort Descending
Sorts from the highest value to the lowest, on the basis of the sorting key element you choose.

Asynchronous
For Windows resource models only: starts data collection at a cycle and

carries it on until the following cycle, when the collection is made available. Specify the type of information you want to obtain, from the following:

- Instance creation
- Instance modification
- Instance deletion

Note: Asynchronous collection can be selected only for those classes whose providers support __InstanceOperationEvent. For more information, see WMI Web site.

11. In the **Every** text box, type the number of cycles you want to elapse between two successive data collections.

    - **Every** = 0, means that the collection is executed only once
    - **Every** = 1, means that the collection is executed at every cycle
    - **Every** = n, means that the collection is executed at every ″n ″ cycles
    - **Every** = -1, means that the collection is invoked and controlled by the monitoring algorithm of the **Visit Tree** section of the Decision Tree Script. This value is automatically set if you have selected the **Collect on demand** check box.

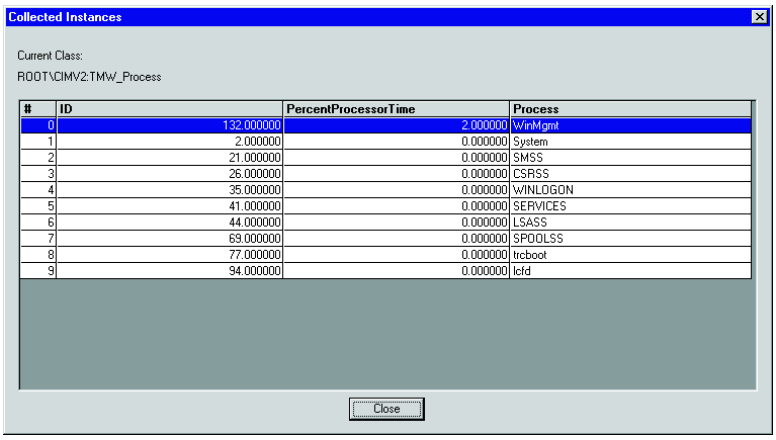
    This text box is not displayed if you select the asynchronous collection.

12. For Windows resource models only: in the **WHERE Clause** text box type a WQL where statement (query) to define filtering criteria for data collection if required. For more information, see “Data Collection” on page 6.

13. For Windows resource models only: click **Collection Test** to see an example of the collected instances of the class you have defined.

    If you are performing an asynchronous collection, data collection starts when you click **Collection Test**. Next, a pop-up dialog opens and you can click **OK** to stop the data collection. In this way you are simulating a cycle time interval.

    The Collected Instances dialog opens and shows the results of data collection.



#	ID	PercentProcessorTime	Process
0	132.000000	2.000000	WinMgmt
1	2.000000	0.000000	System
2	21.000000	0.000000	SMSS
3	26.000000	0.000000	CSRSS
4	35.000000	0.000000	WINLOGON
5	41.000000	0.000000	SERVICES
6	44.000000	0.000000	LSASS
7	69.000000	0.000000	SPOOLSS
8	77.000000	0.000000	trcboot
9	94.000000	0.000000	lcfd

14. Click **Close** to return to Dynamic Model dialog.

15. Click **OK** to apply your changes and close your dynamic model.

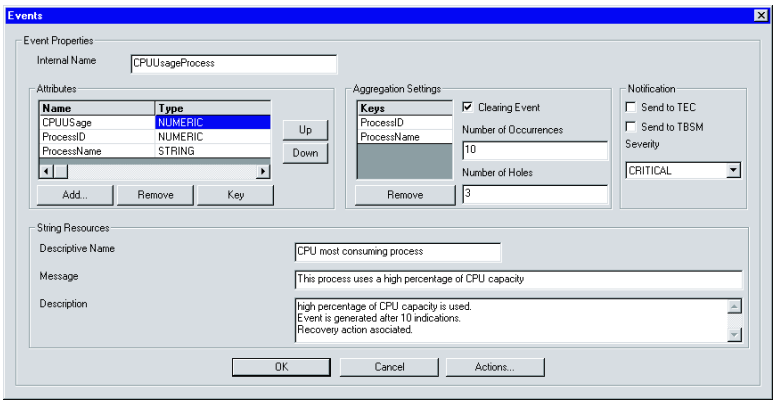
Task 3: Defining Events

When you define an event, you have to specify the attributes that qualify it. In particular, you have to select the most significant attribute for the aggregation process. For more information regarding events, see “Events” on page 7.

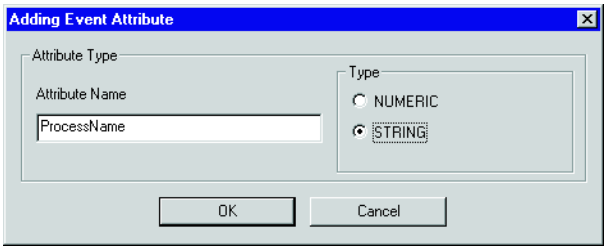
While defining an event, you can also indicate if you want the system to notify the Tivoli Enterprise Console server, or the Tivoli Business Systems Manager, that an event was generated. The Tivoli Monitoring operator can change these selections later, when the resource model is included in a Tivoli Monitoring profile. You can also define the degree of severity and the message that will specify the notification of the event.

To define an event, perform the following steps:

1. From your resource model, double-click **Events**. The Events dialog opens.
2. Type an internal name in the **Internal Name** text box, as shown in the following figure.



3. In the **Attributes** group, click **Add**, to define a new attribute.
4. In the Adding Event Attribute dialog, specify the name and type of the attribute to insert, as shown in the following figure. Click **OK** to return to the Events dialog.



5. Click the **Up** and **Down** buttons to change the attribute order in the list. It must be the same order you follow when you specify the attributes in the SendEvent method in the monitoring algorithm. For more information about the monitoring algorithm, see “Decision Tree Script” on page 12**.**
6. In the **Attributes** box, dialog, click **Key** to move the selected attribute to **Keys** box. Move here the key attributes for your data collection. For more information about key attributes, see “Attributes” on page 7.
7. In the **Number of Occurrences** text box, specify the number of indications that must occur before an event is generated. For more information about indications, see “Events and Indications” on page 7. This will be the default value that the user can change at runtime in the Tivoli Monitoring application.

8. In the **Number of Holes** text box, type the maximum number of monitoring cycles allowed with no indications, for an event to be generated. This will be the default value that the user can change at runtime in the Tivoli Monitoring application.
9. Select **Clearing Event** if you want the system to send a clearing event when the circumstances that generated the event have passed. The Tivoli Enterprise Console server and the Tivoli Business Systems Manager use the clearing event to close the corresponding error event.

   For example, if the resource being monitored is a service, the event would be sent if the service was not available. The event might have an action associated with it to restart the service (or the restart might be done manually), and when the engine detects that the service is available again, a clearing event could be sent, to close the original event. The clearing event itself is not displayed at the Tivoli Enterprise Console server or at the Tivoli Business Systems Manager Java Console.
10. Select **Send to TEC** if you want to send the event notification to the Tivoli Enterprise Console server as a default.
11. Select **Send to TBSM** if you want to send the event notification to the Tivoli Business Systems Manager as a default.
12. In the **Message** text box, type the message you want to send when the event is notified to the Tivoli Enterprise Console server or to the Tivoli Business Systems Manager. When typing the message, specify one or more event attributes between @ symbols. For example: ″This process @ProcessName@, @ProcessID@ is consuming too much CPU″.
13. From the **Severity** drop-down list, select the degree of severity for notification to the Tivoli Enterprise Console server or to the Tivoli Business Systems Manager.
14. Type a descriptive name and a description in the corresponding text boxes.
15. Click **OK** to apply your changes and close the Events dialog.

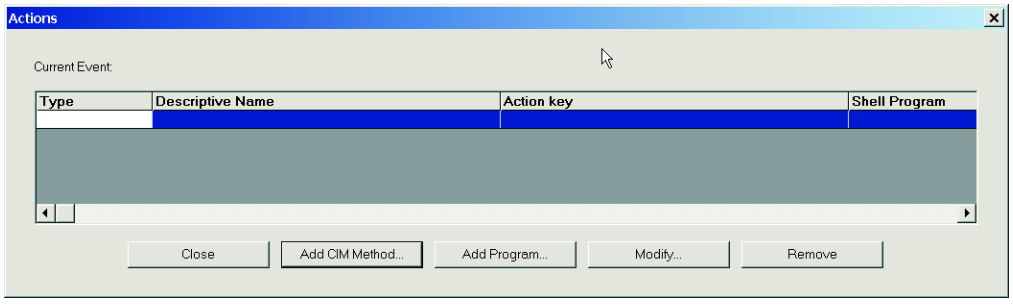
To associate a recovery action with the event, see “Adding Actions”.

Adding Actions

If you want your system to perform some recovery actions on the occurrence of an event, you can associate actions with the event. For more information regarding actions, see “Actions” on page 7.

You can define actions as either the execution of a CIM method or the execution of a program.

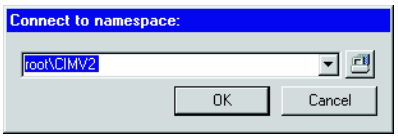
To define an action, click **Actions** on the Events dialog. The Actions dialog opens.



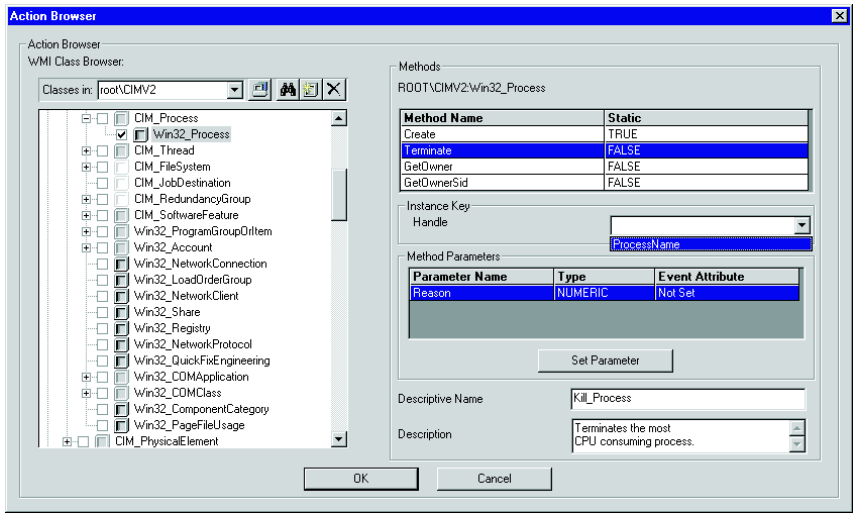
CIM Methods

To associate an action with the execution of a CIM method, perform the following steps:

1. From the Actions dialog, click **Add CIM Method**. The Connect to namespace dialog opens, connecting you to WMI.

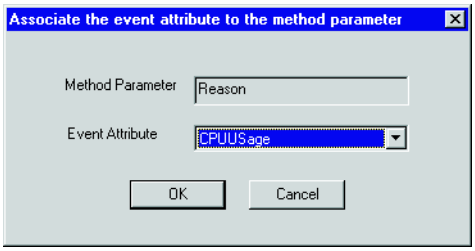


2. Specify the required namespace and click **OK**. The default namespace is root\CIMV2. Use the browse button to locate another namespace.
3. In the WMI DM Workbench Login dialog, type your user name and password, if necessary.
4. Click **OK**. The Action Browser dialog opens, which lists the available classes, and for each class its methods and their parameters. For more information about actions and methods, see “Actions” on page 7.



5. From the **WMI Class Browser** list, select the class that includes the required action.

   The list in the upper part of the **Methods** group shows all the methods of the selected class, and specifies whether the methods are static or non static. For more information about the differences between the two, see “CIM Methods” on page 8.
6. The **Instance Key** group-box is displayed only if you select non-static methods. If you have selected a static method, ignore this step. In the **Instance Key** group-box, select an attribute from the **Handle** drop down list. The selected attribute is a key to associate the action to the required instance. The list displays the string attributes that are associated to the event and compatible with the instance key property. For more information, about event attributes, see “Attributes” on page 7.
7. In the **Method Parameters** group-box, click **Set Parameter** to associate an event attribute to the selected method parameter. The following dialog opens.

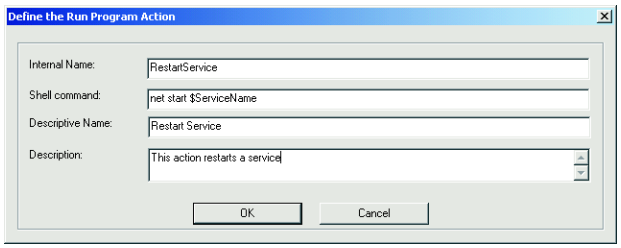


8. From the **Event Attribute** drop down list, select the attribute you want to associate with your method parameter.
9. Click **OK** to return to the Action Browser dialog.
10. Type a descriptive name and a description in the corresponding text boxes.
11. Click **OK** to apply this method.
12. In the Actions dialog, click **Close** to return to the Events dialog. To define additional actions as CIM methods, click **Add CIM Method** and repeat the procedure.

Programs

To associate an action with the execution of a program, perform the following steps:

1. From the Actions dialog, click **Add Program**. The Define the Run Program Action opens.



2. Type the internal name (an identifier for the Run Program Action), the Shell command (the command line which launches a process), a descriptive name and a description, in the corresponding text boxes.

   Note: Make sure you always indicate a slash ("/") in the Shell command path (for example, `c:/action.bat`). This is true also for a Windows endpoint.

3. Click **OK** to complete the definition of this action.
4. In the Actions dialog, click **Close** to return to the Events dialog. To define additional actions as the execution of a program, click **Add Program** and repeat the procedure.

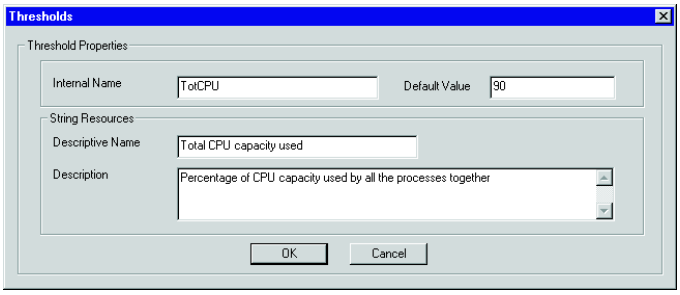
Task 4: Defining Thresholds

For each resource model, you can define one or more thresholds. For more information about thresholds, see "Thresholds" on page 8. Usually, the numeric value you enter in this dialog represents a limit above or below which you do not want your resource to perform. Alternatively, you can use this value as a numeric parameter whose function you must specify in the monitoring algorithm. The default values that you define here can be modified later when this resource model is included in a profile on the Tivoli Monitoring server.

To define a threshold, perform the following steps:

1. From your resource model, double-click **Thresholds**. The Thresholds dialog opens.

2. Type an internal name in the corresponding text box, as shown in the following figure.



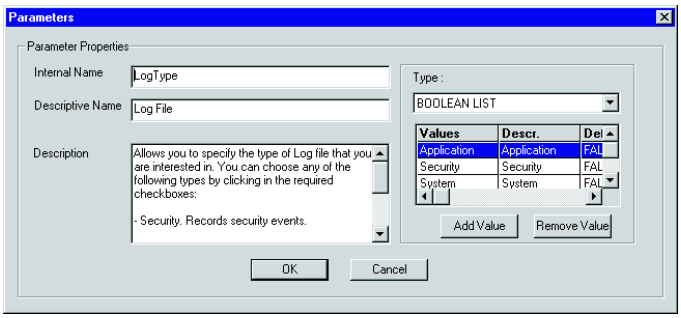
3. In the **Default Value** text box, type a numeric value.
4. Type a descriptive name and a description in the corresponding text boxes.
5. Click **OK** to close and apply this threshold.

Task 5: Defining Parameters

For each resource model, you can define one or more lists of parameters. For more information regarding parameters, see “Parameters” on page 8. To define parameters, you must select the most appropriate type of list and then type the required information. The default values that you define here can be modified later when this resource model is included in a profile on the Tivoli Monitoring server.

To define parameters, perform the following steps:

1. From your resource model, double-click **Parameters**. The Parameters dialog opens.
2. Type an internal name in the corresponding text box as shown in the following figure.



3. Type a descriptive name and a description in the corresponding text boxes.
4. From the **Type** drop-down list, select the required type of list. For more details on the types of lists, see “Parameters” on page 8.
5. Click **Add Value** and specify the parameter as required.
6. Click **OK** to apply the parameters.

Task 6: Defining Data to Log

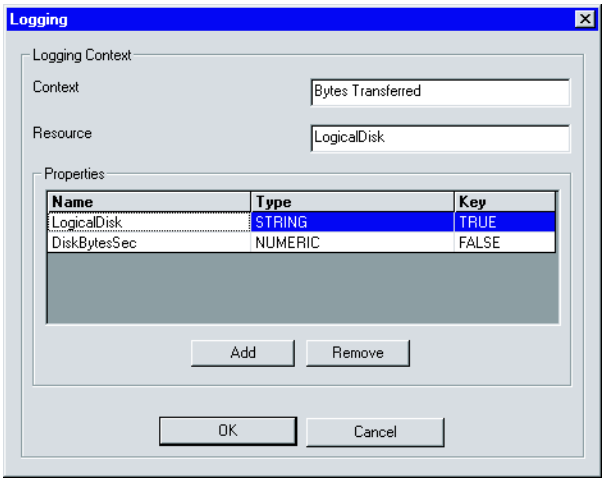
If you want to store data about the attributes of the resource you are monitoring, you can use the logging function. To implement logging in a resource model, first you have to define what data you want to log, then you have to call the DefineLogInst method in the decision tree script. For more information about this method, see “Method **DefineLogInst**” on page 76.

In the resource models you create, logging is disabled by default, but the operator can enable it from the Tivoli Monitoring dialogs.

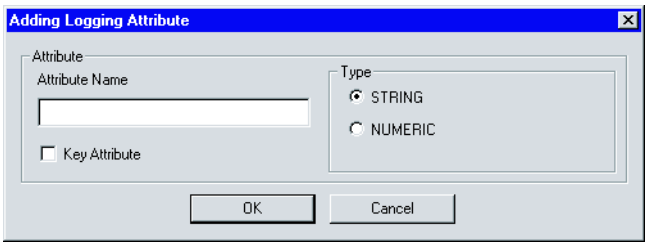
Logged data is stored in a database that can be accessed through the Tivoli Monitoring Web Health Console. For more information, see “Logging” on page 11.

To define the resource attributes that the resource model can log, with reference to the specified context, perform the following steps:

1. From your resource model, double-click **Logging**.
2. In the Logging dialog, specify the required information, as shown in the following figure:



3. In the **Context** text box, type the name of a general problem that the resource logging relates to. **Example**: the bytes transferred to a resource.
4. In the **Resource** text box, type the name of the resource whose state you want to log in relation to the specified context. **Example**: the logical disk.
5. Click **Add**. The Adding Logging Attribute dialog opens, which you use to include the specific properties you want to log. These are attributes of the resource you have specified. **Example**: the name of the disk and the number of bytes transferred per second.



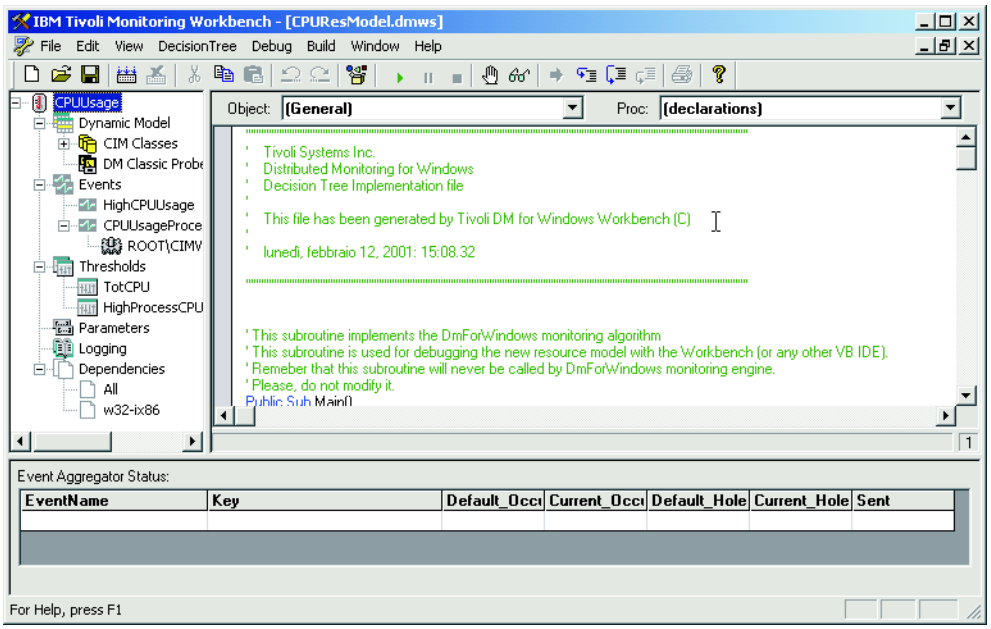
6. In the **Attribute Name** text box, type the specific attribute you wan to log. Often, but not always, these attributes are properties of the class to which the monitored resource belongs.
7. Select either **STRING** or **NUMERIC** to specify the attribute type.
8. Select the **Key Attribute** check-box if you want this attribute to identify the resource instance. Each logging context can have more than one key attribute.

9. Click **OK** to return to the Logging dialog. For each attribute you want to add, repeat the procedure from step 5 on page 35.
10. Click **OK** to save your settings and close Logging dialog.

Task 7: Defining the Decision Tree Script

The decision tree script implements the settings defined during the creation of the resource model. In the decision tree script, you have to write a monitoring algorithm that controls the whole process. The monitoring algorithm is written in Visual Basic or Java Script and appears on the right of the main window when you open a resource model, as shown in the following figure. For more information about the decision tree script, see "Decision Tree Script" on page 12.

Note: The workbench implements the Sax interpreter, which differs slightly from other Visual Basic interpreters, such as Microsoft Visual Studio«. For more information, refer to the following web site: http://www.saxsoft.com/Basic/Details



In the script, you can edit the predefined functions. You can add additional code, but you cannot modify the settings specified in the previous dialogs. If you try to modify them from the script, the new settings will not be saved. However, you can add additional steps to the functions, as long as you are careful writing the new lines of code.

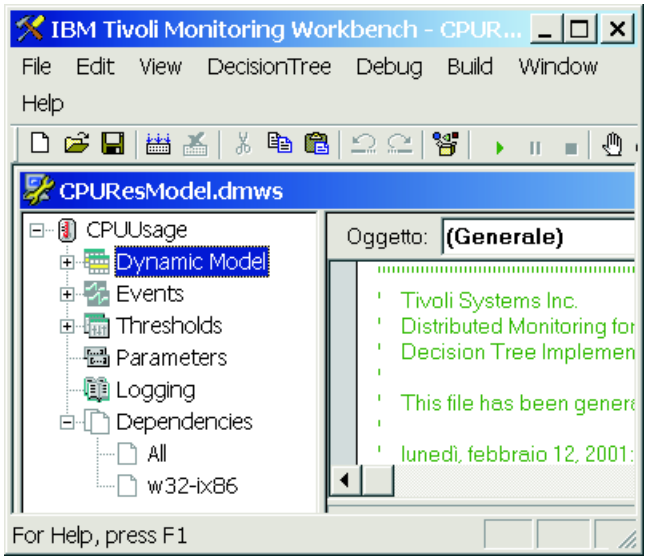
Note: Do not modify remarks contained in tags like <<....>> ... <<\...>>, otherwise the workbench can produce unpredictable results.

When you write the monitoring script, you must specify how your resource model works. When you specify the event attributes, you must list them in the same order as they appear in the Events dialog. For more information on writing the monitoring algorithm in the decision tree script, refer to the syntax descriptions in Appendix A, "Service Object Method Library" on page 57.

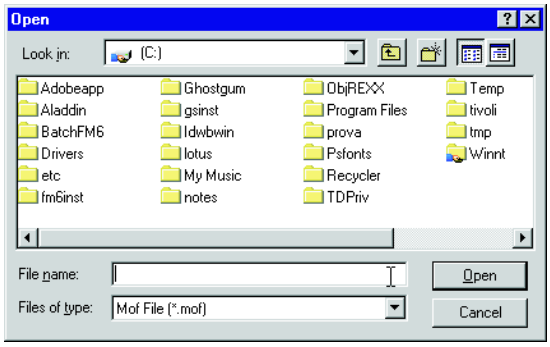
Task 8: Adding Dependencies

If you need to transfer additional files with the resource model, you can use the dependency function. For more information about adding dependencies, see "Dependencies" on page 13. To include additional files to your resource model, perform the following steps:

1. From your resource model, click **Dependencies** to see all the platforms supported by the resource model, as shown in the following figure.



2. Double-click on the platform for which you want to add dependencies.
3. The Open dialog is displayed.



4. In the Open dialog, browse to the file you want to transfer and click **Open**. The file is automatically copied into the Dependencies folder of your resource model.
5. To remove a file from the Dependencies folder, right-click on it and select **Remove**.
6. To view a file added to the Dependencies folder, right-click on it and select **Extract**.

Task 9: Debugging a Resource Model

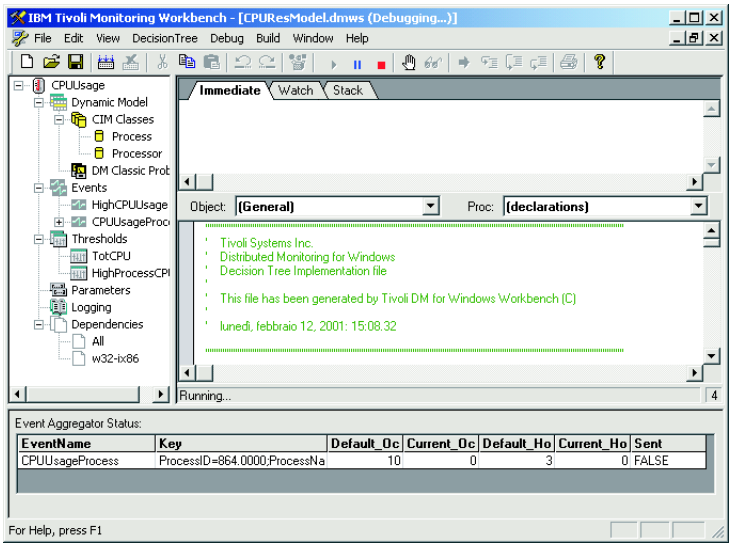
After creating your resource model, you can test it. To debug a resource model, run the script to make sure it does not generate any errors. If a pop-up error dialog appears, see “Exceptions” on page 78 and correct the error. Also, in the decision tree script, the lines containing the error are highlighted. When you have fixed all the errors, run the resource model again. To make sure that the resource model actually detects the problems that you have specified, simulate those problems on your workstation when you run the resource model.

To debug a resource model, perform the following steps:

1. Open your resource model.

2. From the menu bar, click the **Run** button to start debugging:

The Event Aggregator Status is displayed at the bottom of the main dialog, as shown in the following figure:



3. In the Event Aggregator Status, data is updated dynamically and provides you with the following information:

EventName The name of the event that is being generated.

Key The key attribute selected for the indication aggregation process.

Default_Occurrences
The number of occurrences that you specified in the Events dialog for the indication aggregation process.

Current_Occurrences
The number of currently registered occurrences.

Default_Holes
The number of holes that you specified in the Events dialog.

Current_Holes
The number of currently registered holes.

Sent Typically, this reads FALSE, but becomes TRUE when an event is generated.

Note: For more information about the upper-right pane in this dialog, refer to the Sax Basic Help online.

Remote Debugging

If you want, you can also debug your resource model on a remote workstation (only for CIM data collections). This can be useful, for example, if you want to test a resource model on a system without installing the workbench on it. To debug a resource model remotely, perform the following steps:

1. Open your resource model.

2. From the **Debug** menu, select **Remote WMI data collection**.
3. In the Login dialog, type the name of the host you are connecting to, your user name and password.

Now, you are connected to a remote system and if you run a debug of your resource model, it will automatically collect data from the remote system. To run remote debugging, it is necessary that the remote system has either Windows 2000, or Windows NT with WMI installed, while the workbench is not necessary.

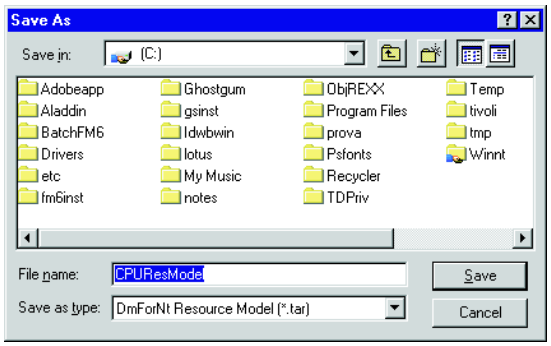
Task 10: Building a Resource Model

After testing your resource model, you have to build it into a TAR file and then install it on the Tivoli Monitoring server.

Building the Package

To build your resource model into a complete package that can later be installed on the Tivoli Monitoring server, perform the following steps:

1. Open your resource model.
2. From the **Build** menu, select **Build package**. The Save As dialog opens.



3. Save the file with a .tar extension.
4. To install the resource model on your Tivoli Monitoring server, from the command line type the following command:

```
wdmrrm -add filename.tar
```

This command adds a new resource model to the list of available resource models on Tivoli Monitoring.

Building the TEC BAROC

To see your resource model events on the Tivoli Enterprise Console, you have to build and export a TEC BAROC file. This file contains all the event definitions specified in the BAROC language. This file can be installed on a TEC Rule base and allows the Tivoli Enterprise Console to display the events of your resource model.

To build a TEC BAROC file, perform the following steps:

1. Open your resource model.
2. From the **Build** menu, select Build TEC BAROC. The Save As dialog opens.
3. Save the file with a .baroc extension.

Exporting the Message Catalog

Optionally, you can build a message file containing all the information that will be displayed in Tivoli Monitoring dialogs. This can be useful if you want to translate your text information into other languages. The workbench is designed to generate resource model packages that contain all the strings for the English Locale, therefore DBCS characters are not supported. All localized strings should be handled separately from the workbench.

The recommended way to work is to export the English message catalog, then translate it into the desired language, compile it using Gencat tool (included in the Application Development Environment), and finally install it manually in the message bundle on the TME server.

Note: Before compiling Japanese message catalogs with Gencat, you need to convert the original files into UTF8 files issuing the following Framework command:

```
wiconv
```

To export a message catalog, perform the following steps:

1. Open your resource model.
2. From the **Build** menu, select **Export Message Catalog**. The Save As dialog opens.
3. Save the file with a .msg extension.

Building an HTML File

You can build an HTML document that contains all the information related to your resource model and included in the descriptive sections of the resource model components.

To build an HTML file, perform the following steps:

1. Open your resource model.
2. From the **Build** menu, select **Build .Html Documentation**. The Save As dialog opens.
3. Save the file with a .html extension.

Creating a Resource Model from a Monitoring Source

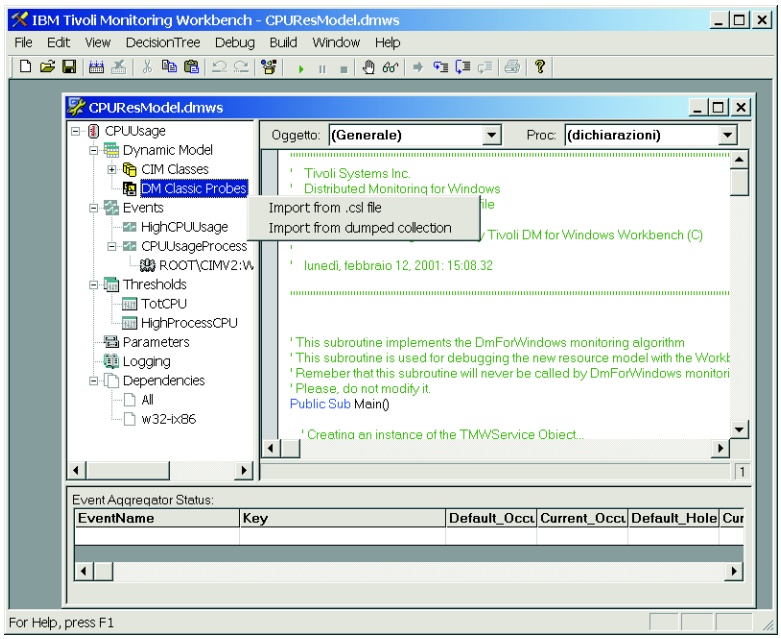
You can import a monitoring source (or a whole monitoring collection) also in a resource model that already exists, by opening the resource model tree structure. You can also import, in the same resource model, monitoring sources that belong to different monitoring collections.

In this procedure, only the definitions of the imported sources are included in the decision tree script automatically. However, you must write the monitoring algorithm that governs the resource model.

To import a monitoring source into a resource model, perform the following steps:

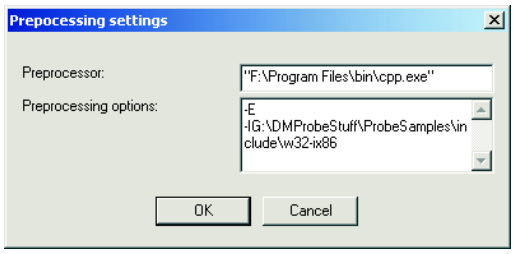
1. From the workbench main dialog, open the resource model tree structure.

2. Under the **Dynamic Model** element, right-click **DM Classic Probes**, and select the required file format:



3. Select the required file format to import and browse to the file location.

   If you are importing a .csl file, specify the preprocessing settings in the displayed dialog:



   a. In the **Preprocessor** text-box, specify the preprocessor you want to use to resolve the preprocessing guidelines used in the .csl file. The specified preprocessor must be in the system path. The workbench automatically installs a default preprocessor and points to it.

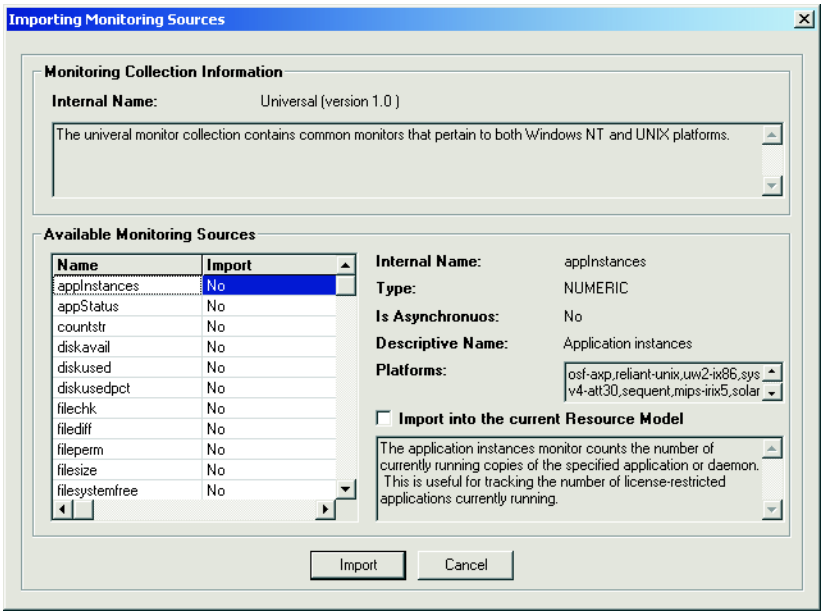
   b. In the **Preprocessing options** text-box, specify the same options used by the mcsl command with Tivoli Distributed Monitoring (Classic Edition).

      These options are passed verbatim to the C preprocessor. In addition to these preprocessor arguments, the workbench also passes the value of the MCSLCPPARGS environment variable. Arguments from the environment variable follow on the preprocessor command line those arguments that are given explicitly as preprocessor options. For more information, see the *Tivoli Distributed Monitoring User's Guide, Version 3.6.2*.

   c. Click **OK**, and, if prompted, save the preprocessing options so that they are available the next time you import a .csl file.

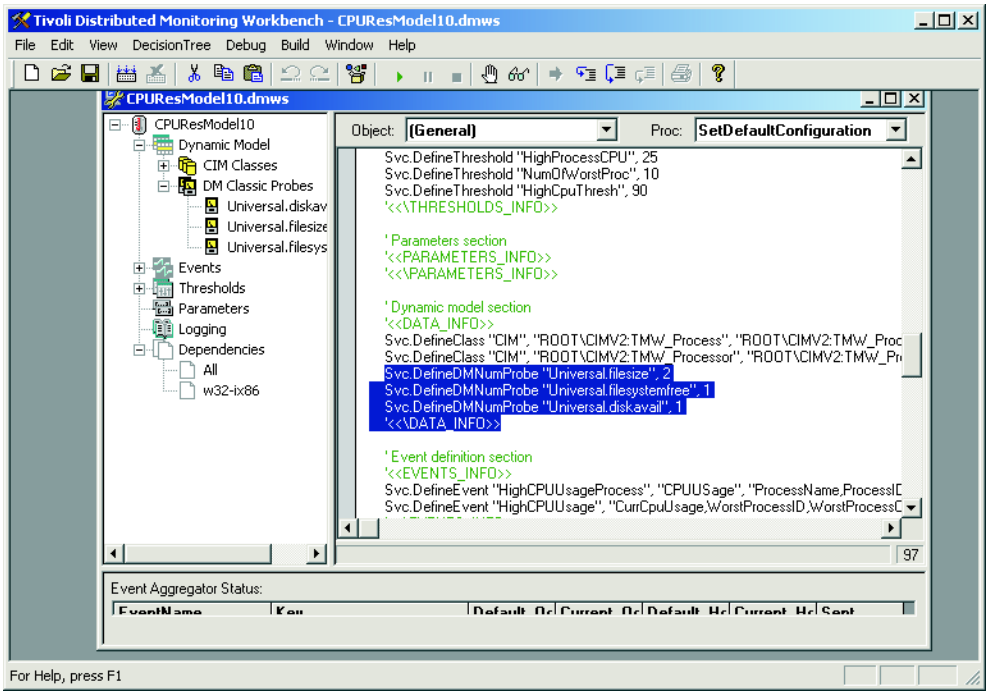
   After the operation is completed successfully, some information about the selected monitoring collection appear in the displayed dialog.

4. In the Importing Monitoring Sources dialog, from the **Available Monitoring Sources** list, double-click the sources you want to import, switching **No** to **Yes**.



5. Select the required sources, and click **Import**.

   In the decision tree script, the definitions of the imported sources are automatically specified under the dynamic model section, as shown in the following window:



The selected monitoring sources are automatically added to the resource model. Now, you just need to write the monitoring algorithm in the decision tree script, as explained before in this chapter.

Resource Model Wizard

An alternate way to create a resource model is by using a wizard that guides you through a series of dialogs. Following this method you don't have to write any scripts or code: you can produce WQL queries and decision tree scripts by means of GUI buttons and options.

The resource model you create in this way is simple and provides you with a limited range of possibilities during resource model creation. However, a resource model created using the wizard is just like any other resource model. This means that, after creating it, you can modify and customize it by editing the decision tree script, adding parameters, or writing filtering queries, just as you would do with any other resource model.

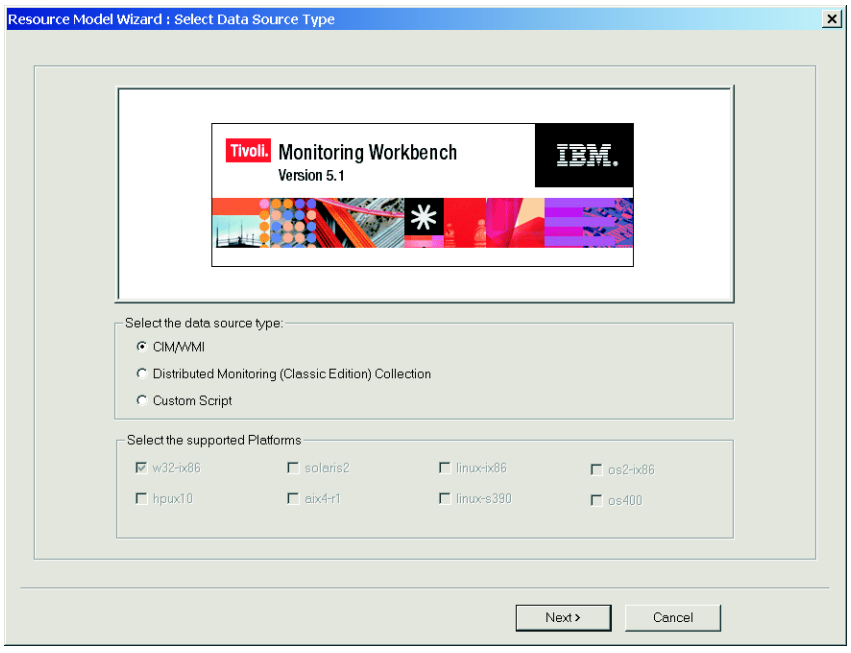
The Resource Model Wizard enables you to create a resource model starting from either:

- a selected CIM class taken from the WMI repository
- or a monitoring collection
- or a custom script

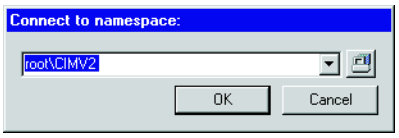
Creating a Resource Model from a CIM class

To create a resource model (from a CIM class) with the wizard, perform the following steps:

1. Open the **File** menu and select **New** to create a new resource model.
2. From the displayed dialog, select the language and click **OK**.
3. From the New Resource Model Workspace, select **Resource Model Wizard** and click **OK**.
4. The Select Data Source Type dialog opens. Select **CIM/WMI** and click **Next**.



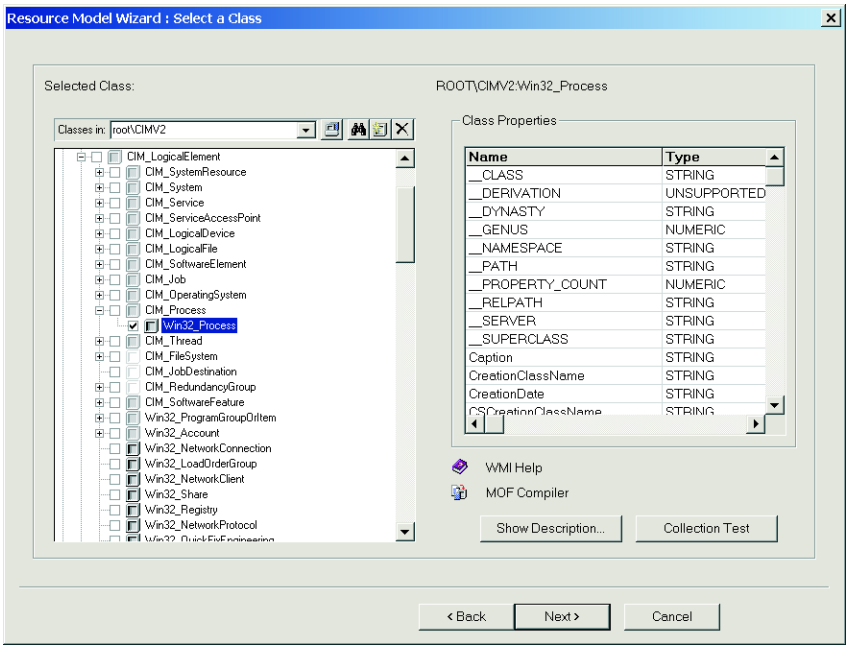
5. In the Connect to namespace dialog, specify the required namespace and click **OK**. The default namespace is root\CIMV2. Use the browse button to locate another namespace.



6. In the WMI DM Workbench Login dialog, type your user name and password, if necessary, and click **OK**. The Select a Class dialog opens.

Selecting a Class

In the **Select a Class** dialog you can choose the class and the specific properties of the resource you want to monitor.



The class list shows all the available classes. When you select a class, all corresponding properties corresponding to the selected class are listed in the **Class Properties** table.

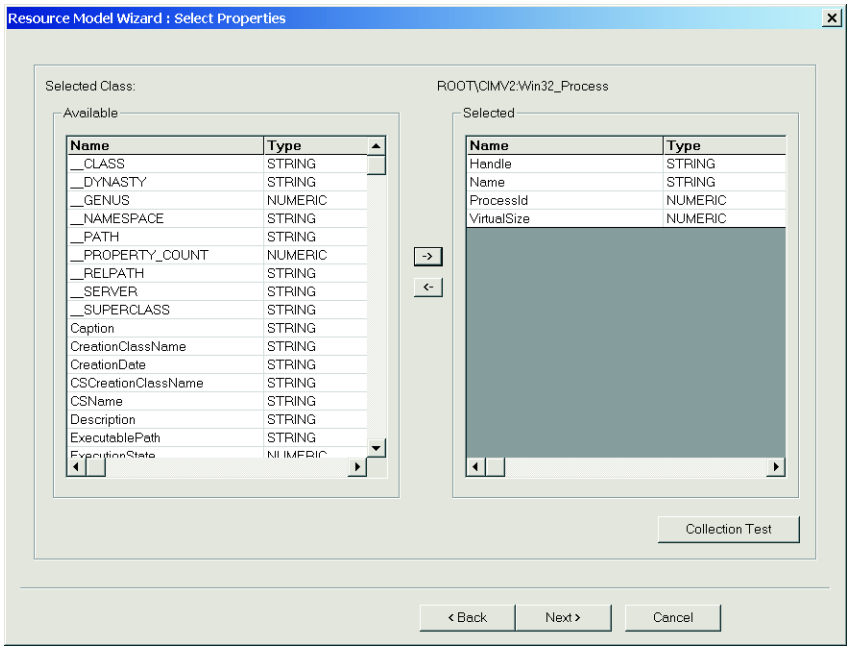
Note: If you select a class that is not included in WMI repository, you have to add to the dependencies the corresponding MOF file and, possibly, the provider.

If you run a collection test at this point, the resource model collects data about all the properties of the selected class.

After selecting a class, click **Next** to open the dialog to select the specific class properties you want to monitor.

Selecting Properties

In the Select Properties dialog you can create a list of properties you want to monitor, that are relevant to the class you have selected in the previous dialog.



The **Available** table lists all the properties available for the selected class. In turn, double-click each of the specific properties you want to monitor, and move it to the **Selected** table.

In this way you include a specific set of properties in your resource model. If you run a collection test at this point, the resource model collects instances of only the selected properties.

If you click the **Back** button and run a collection test from the previous dialog, the resource model still limits the data collection to the selected properties.

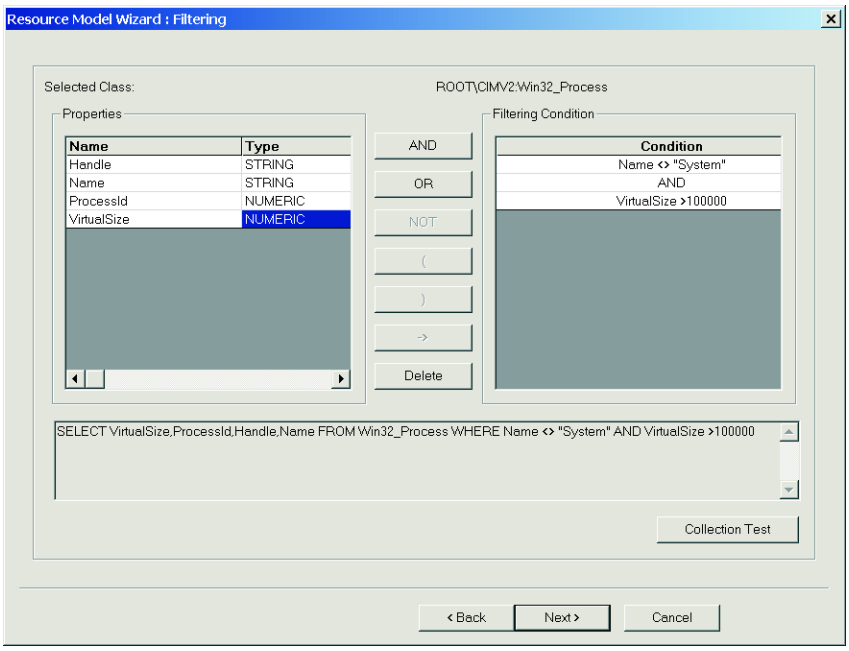
Click **Next** to proceed with the wizard.

Applying Filtering Conditions

You can omit filtering if you do not want to apply filtering criteria to data collection. When you create a resource model following the standard menus, you can write a query in WQL language, specifying how to limit the data collection. For more information about filtering, see “Data Filtering” on page 6.

From the Filtering dialog you can obtain the same result without writing any code. You can build the conditions that make up your query by clicking buttons and selecting items from

lists, as shown in the Filtering dialog:



The **Properties** table lists the properties you have selected for the specified class. You can use each of them to create a filter and define the instances you want to collect.

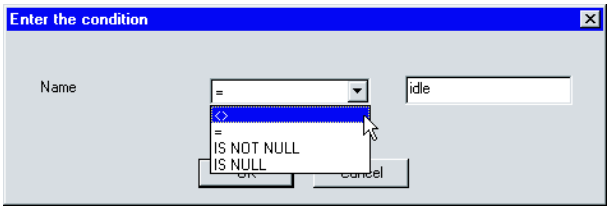
Building a Filtering Query

Building a filtering query means creating a logical expression by means of the logical operators displayed in the buttons located between the tables. The expression associates the properties listed in the **Properties** table with certain conditions. Buttons are enabled or disabled, complying with the logic of the expression.

The filtering query you are building using the GUI is shown in WQL language in the lower pane of the dialog.

To build a filtering query using the GUI, perform the following steps:

1. From the **Properties** table, double-click the property whose instances you want to filter. The Enter the condition dialog opens, enabling you to enter the condition you want to apply to that property. The possible conditions you can set vary, depending on whether the property is a string or a numeric value.



2. In the Enter the condition dialog, select an item from the drop down list and enter a value in the text box to complete the filtering condition.
3. Click **OK** to return to the Filtering dialog.
4. If you want to add one or more conditions, click one of the enabled buttons between the tables and repeat the procedure from step 1.
5. Click **Delete** if you want to delete the last condition displayed in the **Condition** table.

6. After completing your filtering query, click **Collection Test** if you want to see the result of the filtered data collection.
7. Click **Next** to proceed with the resource model wizard.

Specifying Event Triggering Conditions

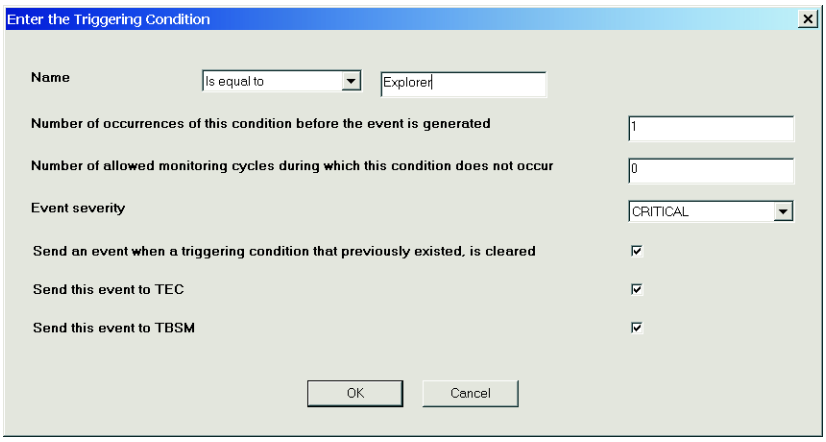
To complete your resource model you have to specify when you want an event to be generated and whether you want it to be notified to the Tivoli Enterprise Console server or to the Tivoli Business Systems Manager.

In the following dialog you select one or more properties out of all monitored properties. Then you specify the conditions that must be satisfied to generate an event.

Example. An event is generated when Process Name is equal to Explorer, and another is generated when the process Status is not equal to Running.

To specify this information, perform the following steps:

1. From the Specify the event triggering condition dialog, double-click the property you want to associate with event triggering. The Enter the Triggering Condition dialog opens:



2. From the drop-down list on the top of the dialog, select the logical operator to associate with the selected property.

   The options displayed in the list for a numeric property are as follows:

   > Records the value of the selected property, compares it to the value you specify in the text box, and triggers an event if the current value is greater than the specified value.

   < Records the value of the selected property, compares it to the value you specify in the text box, and triggers an event if the current value is less than the specified value.

   = Records the value of the selected property, compares it to the value you specify in the text box, and triggers an event if the current value is equal to the specified value.

   <> Records the value the value of the selected property, compares it to the value you specify in the text box, and triggers if the current value is not equal to the specified value.

   >= Records the value of the selected property, compares it to the value you

specify in the text box and triggers an event if the current value is larger than or equal to the specified value.

<= Records the value of the selected property, compares it to the value you specify in the text box and triggers an event if the current value is smaller than or equal to the specified value.

Increases at least by
Checks the current value of the selected property against the previous value and triggers an event if the current value is greater than the previous one by a value equal to or greater than the value you specify in the text box.

% increases at least by
Checks the current value of the selected property as a percentage of the previous value and triggers an event if current value is greater than the previous one by a value equal to or greater than the value you specify in the text box.

Decreases at least by
Checks the current value of the selected property against the previous value and triggers an event if the current value is smaller than the previous one by a value equal to or greater than the value you specify in the text box.

% decreases at least by
Checks the current value of the selected property as a percentage of the previous value and triggers an event if current value is smaller than the previous one by a value equal to or greater than the value you specify in the text box.

Changes by Compares the previous and current values of the selected property and triggers an event if between these two values there is a difference equal to the value you specify in the text box.

Changes at least by
Compares the previous and current values of the selected property and triggers an event if between these two values there is a difference equal to or greater than the value you specify in the text box.

Is out of the range
Compares the current value of the selected property against the defined range limits, and triggers an event if the current value falls outside the specified range.

The options displayed in the list for a string property are as follows:

Is equal to Records the value of the selected property, compares it to the value you specify in the text box, and triggers an event if the current value is equal to the specified value.

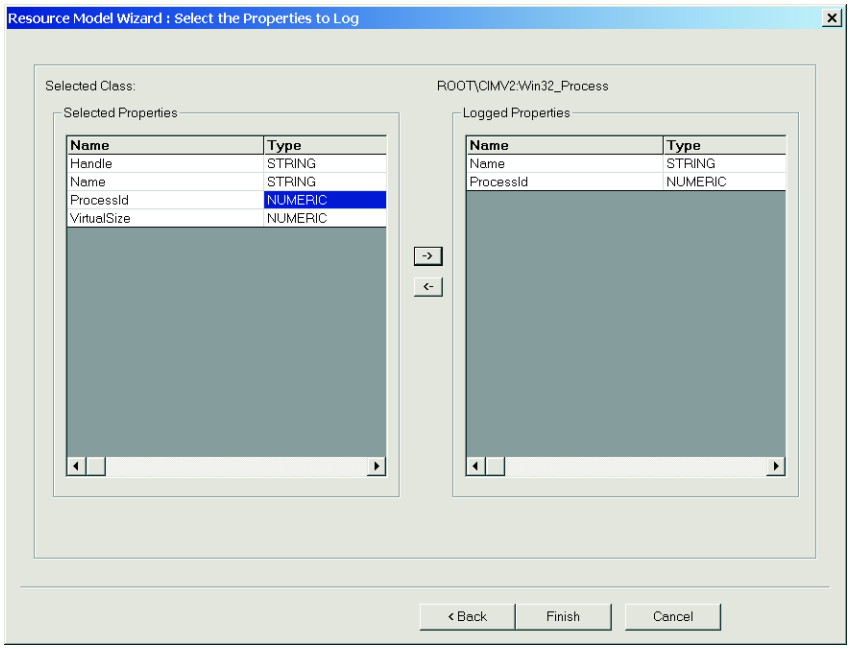
Is not equal to
Records the value of the selected property, compares it to the value you specify in the text box, and triggers an event if the current value is different from the specified value.

Contains Records the value of the selected property, compares it to the value you specify in the text box, and triggers an event if the current value contains the specified value.

3. Enter either a string or numeric value in the text box next to the drop-down list. In this way, you build a logical expression, specifying a value and the logical operator that associates it with the selected property.
4. In the upper-right text box, enter a number that represents the number of occurrences of the specified condition that must be detected before the resource model generates an event.
5. In the second text box on the right, enter a number that represents the number of holes allowed before the resource model generates an event. For more information about holes, see “Events and Indications” on page 7.
6. From the **Event severity** drop-down list, select the severity degree you want to apply to this event.
7. Select the first the check box if you want an event to be generated when the conditions that had previously caused an event have resolved.
8. Select the remaining two check boxes if you want to notify this event to the Tivoli Enterprise Console server or to the Tivoli Business Systems Manager.
9. Click **OK** to return to Specify the triggering conditions dialog.
10. Repeat the procedure for each event you want to generate.
11. Click **Next** to proceed with the wizard.

Selecting the Properties to Log

The Select the properties to Log dialog provides an optional step. That is, you can click **Finish** to end the creation of the resource model, or you can use the dialog to log the monitoring results of one or more properties.



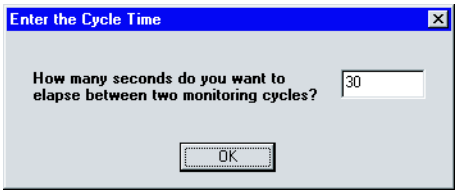
To perform logging, select the required properties in the **Selected Properties** list box and move them to the **Logged Properties** list box using the upper arrow button.

Click **Finish** to proceed with the resource model creation.

Enter a Cycle Time

The last step you have to perform to complete the creation of a resource model through the wizard is to specify the time interval you want to elapse between two successive monitoring cycles.

This value is expressed in seconds and must be entered in the following dialog:



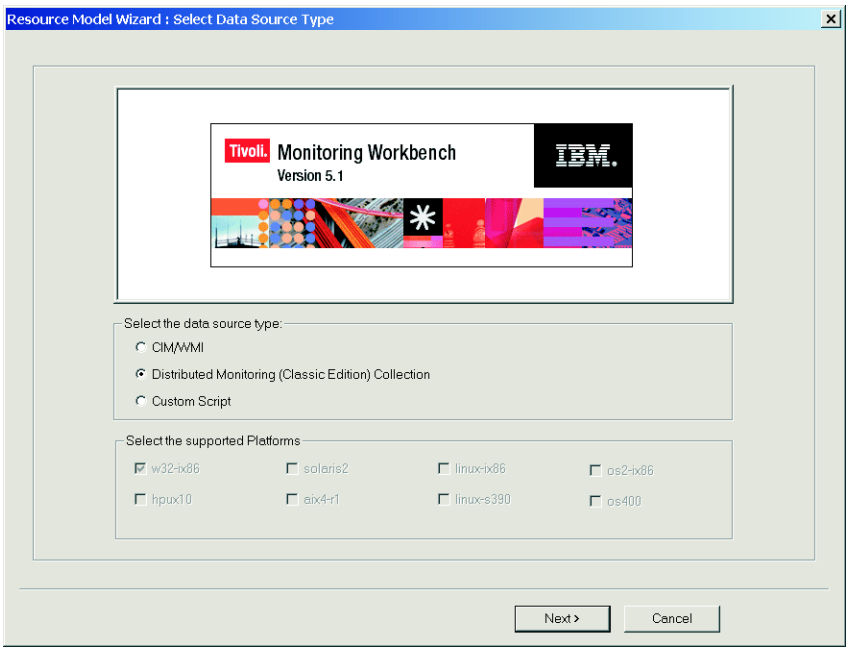
After typing the required value, click **OK** to close the dialog and complete the resource model creation.

Creating a Resource Model from a Monitoring Source

To import using the wizard a monitoring collection created with Tivoli Distributed Monitoring (Classic Edition), perform the following steps:

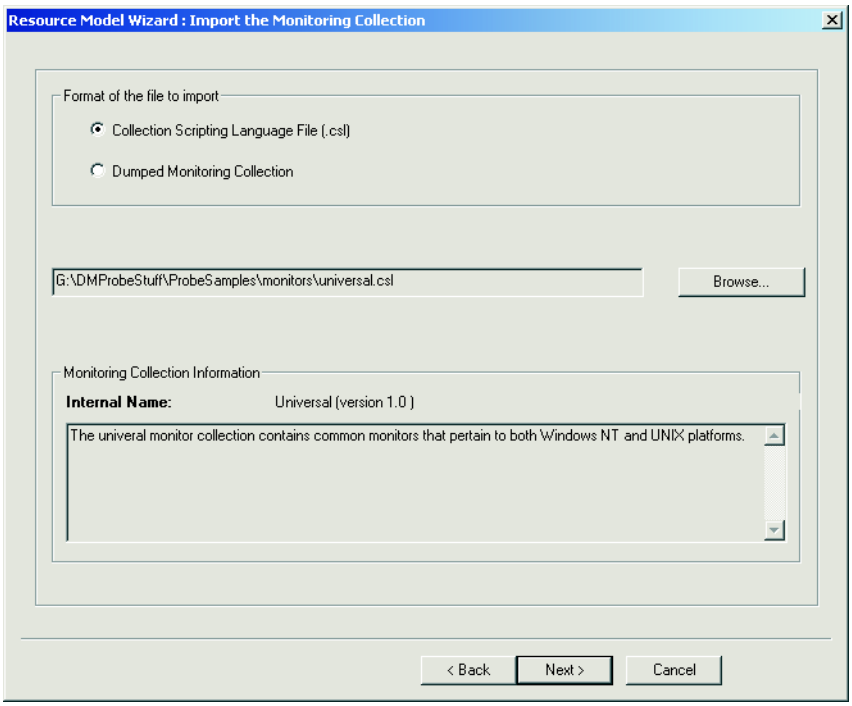
1. Open the **File** menu and select **New** to create a new resource model.
2. From the displayed dialog, select the language of your resource model and click **OK**.
3. From the New Resource Model Workspace, select **Resource Model Wizard** and click **OK**.

   The Select Data Source Type dialog opens:



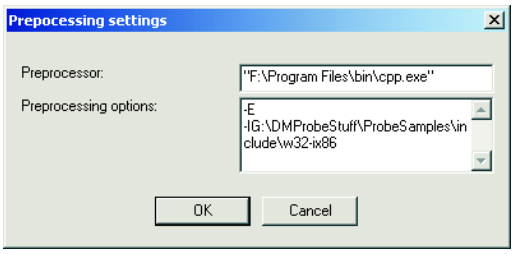
4. Select **Distributed Monitoring (Classic Edition) collection** and click **Next**.

The Import the Monitoring Collection dialog opens.



5. Select the required file format to import, and browse to the file location.

   If you are importing a .csl file, specify the preprocessing settings in the displayed dialog:



   a. In the **Preprocessor** text-box, specify the preprocessor you want to use to resolve the preprocessing guidelines used in the .csl file. The specified preprocessor must be in the system path. The workbench automatically installs a default preprocessor and points to it.

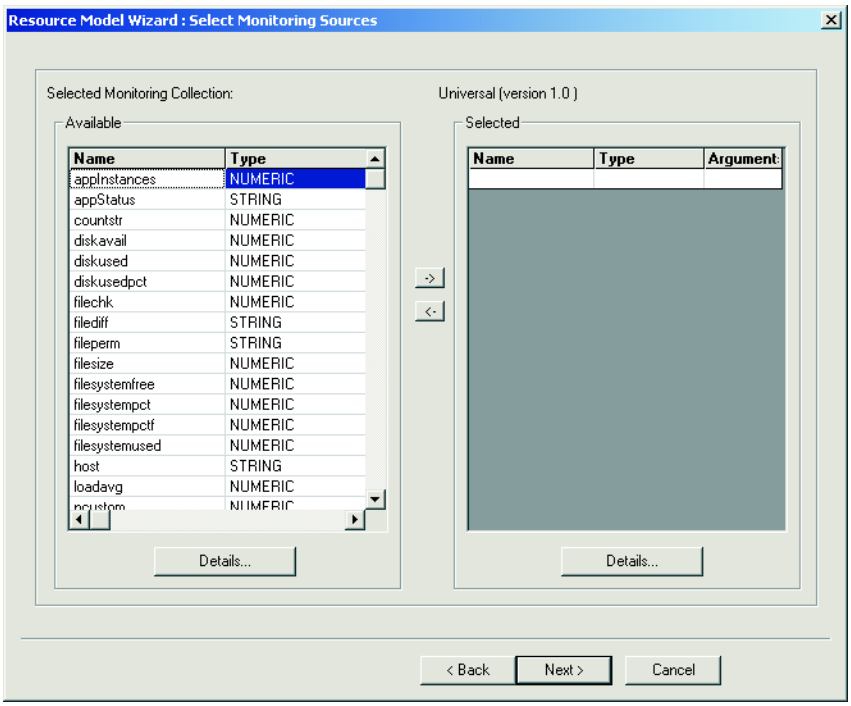
   b. In the **Preprocessing options** text-box, specify the same options used by the mcsl command with Tivoli Distributed Monitoring (Classic Edition).

      These options are passed verbatim to the C preprocessor. In addition to these preprocessor arguments, the workbench also passes the value of the MCSLCPPARGS environment variable. Arguments from the environment variable follow on the preprocessor command line those arguments that are given explicitly as preprocessor options. For more information, see the *Tivoli Distributed Monitoring User's Guide, Version 3.6.2*.

   c. Click **OK**, and, if prompted, save the preprocessing options so that they are available the next time you import a .csl file.

After the operation is completed successfully, some information about the selected monitoring collection appears in the displayed dialog.

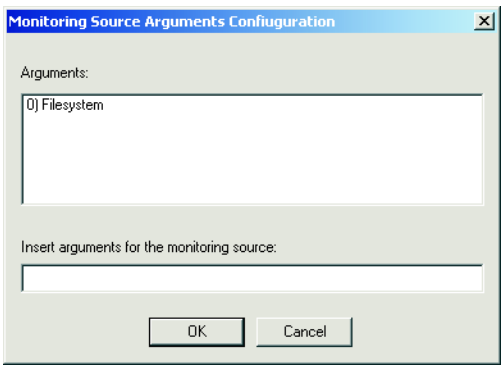
6. Click **Next**. The Select Monitoring Sources dialog opens.



7. To see the details about a particular monitoring source, select it in the **Available** list and click **Details**.

8. From the **Available** list, double-click the monitoring sources you want to import in your resource model. This list displays all sources that belong to the selected monitoring collection. You can import all sources, or a subset of them.

   If you select a source that has arguments, the Monitoring Source Argument Configuration dialog opens.



9. The **Arguments** text box displays a list of all mandatory arguments to be specified for the selected source.

10. In the **Insert arguments for the monitoring source** text box, specify the required arguments, separated by blanks.

    The arguments you specify here identify the monitored resource. They are included in the resource model structure as parameters of the resource model itself, therefore they

represent instances of the monitored resource. So, you can modify a resource model at a later time, adding new parameters to it to monitor more instances of the same resource.

11. Click **Next** and follow the remaining dialogs of the wizard to configure triggering conditions, logging, and cycle time. These steps are the same as those described in the creation through the wizard of a resource model from a CIM class.

When you have completed the procedure, the workbench produces a resource model that contains both the definition of the imported monitoring source, and the code necessary to implement the logic of the analysis.

Creating a Resource Model from a Custom Script

Now that it is more fully integrated with Tivoli Distributed Monitoring (Classic Edition), the workbench enables you to use resource models to launch shell commands or scripts and retrieve the output.

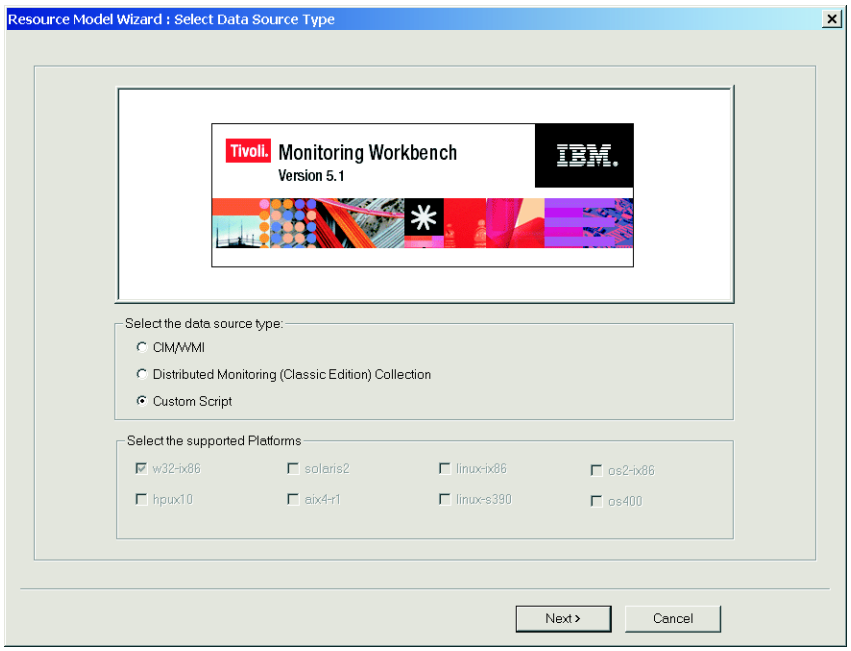
You need only to include in the resource model the commands or the scripts you want to launch, and they will be embedded in the resource model, together with its other components.

To Launch a Shell Command Using the Wizard

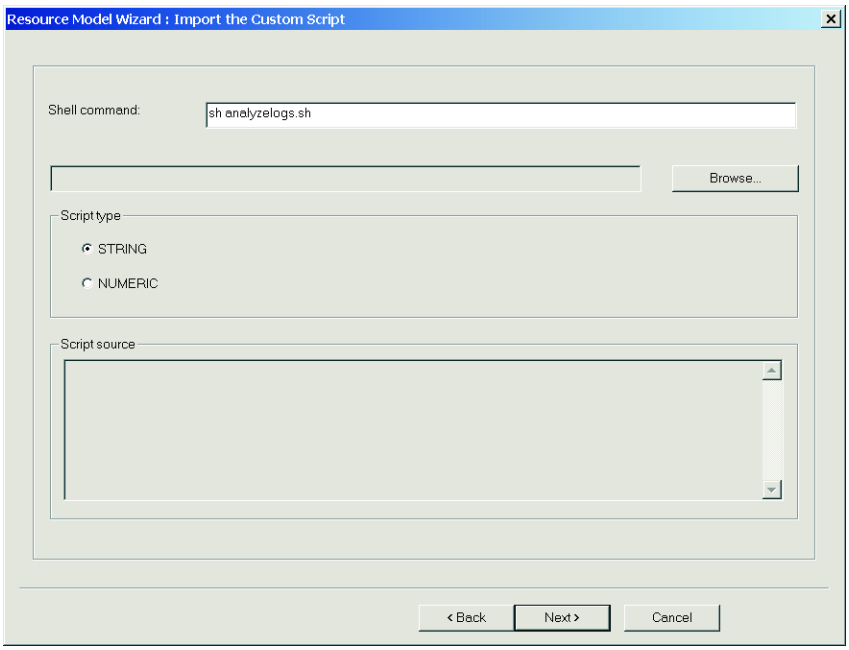
To include a shell command in a resource model, perform the following steps:

1. Open the **File** menu and select **New** to create a new resource model.
2. From the New dialog, select the language of your resource model and click **OK**.
3. From the New Resource Model Workspace dialog, select **Resource Model Wizard** and click **OK**.

   The Select Data Source Type dialog opens:



4. Select **Custom Script**. The Import the Custom Script dialog opens.



5. In the **Shell command** text box, type the command you want to launch when you run this resource model. This could be, for example, a command to determine whether a file is contained in the specified directory.
6. In the **Script type** group-box, select the kind of output you expect from the command.
7. Click **Next** and follow the remaining dialogs of the wizard to configure triggering conditions, logging, and cycle time. These steps are the same as those described in the resource model wizard creation, included in this chapter.

   In the triggering dialog, for example, you can specify that you want an event to be generated if the command output is equal to the given string, that is, if the file is found in the directory.

To Import a Script with the Wizard

With the workbench, you can also import a custom script (like a bash, or perl script) into a resource model, so that the script is launched when you run the resource model.

To import a custom script, follow the procedure described in “To Launch a Shell Command Using the Wizard” on page 53, from step 1, to step 3.

From the Import the Custom Script dialog, browse to the file that contains your script.

The workbench automatically imports that file and adds it to the dependencies of the resource model. Therefore, the file containing the script is automatically installed on the endpoint, together with the resource model.

6

Resource Model Troubleshooting

When a resource model is run through the Tivoli Monitoring application, the action the engine can take depends on the state of the resource model. The various possible states are either assigned automatically by the engine, or specified by the user in the decision tree script. The resource model states are displayed together with the monitoring results, by the command line, Tivoli Monitoring Web Health Console, or Tivoli Business Systems Manager, when enabled.

The following table shows the states automatically assigned by the engine, the external causes from which the states result, and the related actions taken by the engine.

Resource Model State	Cause of the State	Action Taken by Engine
Running	Resource model running successfully.	None
Stopped	User stopped the resource mode, or it has successfully finished running.	None
Disabled	User action.	None
Scheduled	User has scheduled resource model for a later time.	None
Error	An error is preventing the resource model from running.	Every 3 minutes the engine automatically tries to rerun it.
Missing prerequisites	Required prerequisites are missing.	None
Not compiled	An error in Visual Basic or Java Script code.	None

You can program resource model states so that they derive from the return codes generated by the monitoring algorithm you write. You can use the VisitTree, in the Init, or in the SetDefaultConfiguration functions to specify what return codes are given for particular states of the resource models.

All resource model states listed in the following table are caused by return codes as specified in the monitoring algorithm. Therefore when you write the monitoring algorithm, you can foresee specific situations that could cause the resource model to fail. You can specify a return code that must be returned in these situations. Return codes match with specific resource model states and cause the engine to take some actions.

Example: You have created a resource model that monitors some files. You know that sometimes one of these file can be temporarily locked, This would prevent the resource model from running successfully. You can bypass this problem adding a few lines to the VisitTree function, specifying that if one file is not accessible, the VisitTree must return a

specific code included between 601 and 800. This return code means that the resource model state is Retrying and the engine will keep on trying to run it until it finds the file unlocked.

In the following table, the first column lists the resource model states. The second column lists the return code ranges. The third column lists how Tivoli Monitoring interprets resource model states and the consequent actions it takes.

Resource Model State	Cause of the State	Action Taken by the Engine
Failed 1-200. The resource model has failed	VisitTree return code: 201-400	None
Failing 1-200. The resource model has an error.	VisitTree return code: 401-600	The engine automatically retries every three minutes to run the resource model.
Retrying 1-200. The resource model is running	VisitTree return code: 601-800	Retries 3 times in each cycle time, indefinitely, to run the resource model.
Unable to start 1-200. The resource model is unable to start. Assumed missing prerequisites.	Init, or SetDefaultConfiguration return code: 801-1000	None
Recovering 1-200. The resource model is running	VistTree return code: 1001-1100	Once per cycle, for three cycles only, tries to rerun the resource model. After three unsuccessful attempts, interprets the resource model as failed.
Failed after recovery	The resource model failed in three successive cycles. Interpreted as failed.	None

Note: If in the monitoring algorithm you specify a return code that is not included in the supported ranges, the workbench assumes this return code to be 0 (successful).

The return codes that are displayed by the Tivoli Monitoring Web Health Console are not the same return codes you enter in the monitoring algorithm. Those displayed by the Tivoli Monitoring Web Health Console are the result of the following formula:

modulus of return code / 200

That means the remainder of the return code divided by 200. **Example**: if the VisitTree returns **1051**, the Tivoli Monitoring Web Health Console displays: **Recovering 51.**

You can use different return codes to better specify different causes for the same resource model state. For example, a resource model that monitors some files can be in Retrying state because it finds a file locked (one return code), or because this file does not exist (another return code).

A

Service Object Method Library

This Appendix, describes the Class Object library. This library contains the syntax required to express all main functions needed to create an object in a resource model.

The chapter also contains a list of possible errors users can get when they run a resource model, as well as guidelines for producing the main and most useful scripts.

The workbench also allows you to program your own functions, to create customized scripts. If you do this, the results are less predictable, so it is essential to test your scripts thoroughly before distributing them.

Note: All methods described in this appendix are written in Visual Basic. The same methods can also be written in Java Script, by modifying the syntax and complying with the code rules. The following table lists some examples of the same methods expressed in both languages:

Visual Basic	Java Script
GetNumOfInst (ClassName As String) As Long	int GetNumOfInst (String className)
GetNumProperty (ClassName As String, idxAs Long, PropName As String) As Double	double GetNumProperty (String className, int index, String propertyName)
DefineClass (Source As String, AliasName As String, RealClassName As String, WhereClause As String,NumProps As String, StrProps As String, SortType As String, SortField As String, Top as long, Every as long)	DefineClass (String source, String aliasName, String realName, String whereClause, String numProperties, String strProperties, String sortType, String sortField, int Top, int every)

The main differences between the two languages are pointed out in the following table:

Visual Basic	Java Script
Parameter name **As** Parameter Type.	Parameter Type Parameter name.
The type of value returned by the function is expressed at the end of the function. **Example**: GetNumProperty(...) As Double	The type of value returned by the function is expressed at the beginning of the function. **Example**: double GetNumProperty(...)
-	Instructions end with a semicolon.

Basic Object Methods

This section describes basic methods for the TMWService Object. You can refer to it for the syntax required to call the specific methods.

General Settings

The following methods help you to express functions for resource model configuration.

Method GetModelName

Syntax

```
Object.GetModelName() As String
```

Description

Returns the name of the resource model.

Error code

S_OK.

Method GetCycleTime

Syntax

```
Object.GetCycleTime() As Double
```

Description

Returns the cycle time value.

Error code

S_OK.

Dynamic Model

The following methods help you to express functions for dynamic model configuration.

Method GetNumOfInst

Syntax

```
Object.GetNumOfInst(ClassName As String) As Long
```

Parameters

ClassName

The name of the class.

Description

Returns the number of the collected instances of the class named *ClassName* in the current monitoring cycle.

Remarks

The *ClassName* must be previously defined through in the Dynamic model dialog.

Error codes

S_OK; TMWSERVICE_E_CLASS_NOT_FOUND.

Method GetNumProperty

Syntax

```
Object.GetNumProperty(ClassName As String,
idx As Long, PropName As String) As Double
```

Parameters

ClassName

The name of the class.

idx The instance index.

PropName
The name of a NUMERIC property.

Description
Returns the value of the property named *PropName* of the instance number *idx* of the class named *ClassName*.

Remarks
idx counter goes from 0 to NumOfInst -1.

Error codes
S_OK; TMWSERVICE_E_PROPERTY_NOT_FOUND.

Method GetStrProperty

Syntax

```
Object.GetStrProperty(ClassName As String,
idx As Long, PropName As String) As String
```

Parameters

ClassName
The name of the class.

idx The instance index.

PropName
The name of a STRING property.

Description
Returns the value of the property named *PropName* of the instance number *idx* of the class named *ClassName*.

Remarks
idx counter goes from 0 to NumOfInst -1.

Error codes
S_OK; TMWSERVICE_E_PROPERTY_NOT_FOUND.

Method AssociateParameterToClass

Syntax

```
Object.AssociateParameterToClass (ParameterName As String,
ClassName As String)
```

Parameters

ParameterName
The name of the parameter to use as the argument for the provider operations on the specified class.

ClassName
The name of the class whose provider operations are to be executed using the specified parameter as an argument.

Description
Defines an existing parameter as the argument for the provider operations on the class named *ClassName*.

Remarks
This method is for UNIX platforms, only.

Method AssociateParameterToClassProperty

Syntax

```
Object.AssociateParameterToClassProperty (ParameterName As String, ClassName
As String, Property As String)
```

ParameterName
The name of the parameter to use as the argument for the provider operations on the specified class property.

ClassName
The name of the class which the property belongs to.

PropertyName
The name of the property whose provider operations are to be executed using the specified parameter as an argument.

Description
Defines an existing parameter as the argument for the provider operations on the class property named *ClassProperty*.

Remarks
This method is for UNIX platforms, only.

Method CallDMNumProbe

Syntax

```
Object.CallDMNumProbe(ProbeKey As String, Arguments As String) As Double
```

Parameters

Probekey
A key that uniquely identifies the monitoring source; is always in the form *CollectionName.MonitorName.*

Arguments
A string that contains the monitor arguments separated by blanks.

Returns
The numeric value resulting from the monitor call.

Description
This method runs the monitoring source identified by ProbeKey and returns its output.

Remarks
The monitoring source activation must return within 60 seconds, otherwise an error is generated.

Error codes
TMWSERVICE_E_PROBE_WRONG_ARGS_NUM
TMWSERVICE_E_PROBE_NOT_LOAD
TMWSERVICE_E_PROBE_NOT_FOUND
TMWSERVICE_E_NO_INTERP_SUPPORT TMWSERVICE_E_NO_DATA
TMWSERVICE_E_IMPLIED_ERROR
TMWSERVICE_E_ERRORVALUE_SCRIPT_ERROR S_OK.

Method CallDMStrProbe

Syntax

```
Object.CallDMStrProbe(ProbeKey As String, Arguments As String) As String
```

Parameters

Probekey
A key that uniquely identifies the monitoring source; is always in the form *CollectionName.MonitorName*.

Arguments
A string that contains the monitor arguments separated by blanks.

Returns
The string value resulting from the monitor call.

Description
This method runs the monitoring source identified by ProbeKey and returns its output.

Remarks
The monitoring source activation must return within 60 seconds, otherwise an error is generated.

Error codes
TMWSERVICE_E_PROBE_WRONG_ARGS_NUM
TMWSERVICE_E_PROBE_NOT_LOAD
TMWSERVICE_E_PROBE_NOT_FOUND
TMWSERVICE_E_NO_INTERP_SUPPORT TMWSERVICE_E_NO_DATA
TMWSERVICE_E_IMPLIED_ERROR
TMWSERVICE_E_ERRORVALUE_SCRIPT_ERROR S_OK.

Thresholds

The following methods help you to express functions for threshold configuration.

Method GetThreshold

Syntax

```
Object.GetThreshold(ThName As String) As Double
```

Parameters

ThName
The name of the threshold.

Description
Returns the value of the threshold named *ThName*.

Error codes
S_OK; TMWSERVICE_E_THRESHOLD_NAME_NOT_DEFINED.

Parameters

The following methods help you to express functions for parameters configuration.

Method GetNumParameterCount

Syntax

```
Object.GetNumParameterCount(ParamName As String) As Long
```

Parameters

ParamName
The name of a NUMERIC parameter.

Description
Returns the number of values contained by the parameter named *ParamName*.

Error codes
S_OK; TMWSERVICE_E_PARAMETER_NOT_DEFINED.

Method GetStrParameterCount

This method is used to retrieve the number of items of string, boolean, and choice list types. For the string lists, the returned value is the number of elements. For the boolean lists, the returned value is the number of elements set to true. For the choice lists, the returned value is 1.

Syntax

```
Object.GetStrParameterCount(ParamName As String)_
As Long
```

Parameters

ParamName
The name of a STRING parameter.

Description
Returns the number of values contained by the parameter named *ParamName*.

Error codes
S_OK; TMWSERVICE_E_PARAMETER_NOT_DEFINED.

Method GetNumParameter

Syntax

```
Object.GetNumParameter(ParamName As String,_
idx As Long) As Double
```

Parameters

ParamName
The name of a NUMERIC parameter.

idx The index of the parameter value.

Remarks
idx counter goes from 0 to NumOfInst -1.

Description
Returns the NUMERIC value contained at the index idx of the parameter named *ParamName*.

Error codes
S_OK; TMWSERVICE_E_PARAMETER_NOT_DEFINED; TMWSERVICE_E_PARAMETER_BAD_INDEX.

Method GetStrParameter

This method is used to retrieve the values of string, boolean, and choice list types. For the string lists, the returned values are the element value. For the boolean lists, the returned values are the values of the elements set to true. For the choice lists, the returned value is the selected value.

Syntax

```
Object.GetStrParameter(ParamName As String, idx As Long) As String
```

Parameters

ParamName
The name of a STRING parameter.

idx The index of the parameter value.

Description

Returns the STRING value contained at the index idx of the parameter named *ParamName*.

Remarks

idx counter goes from 0 to NumOfInst -1.

Error codes

S_OK; TMWSERVICE_E_PARAMETER_NOT_DEFINED; TMWSERVICE_E_PARAMETER_BAD_INDEX.

Events

The following methods help you to express functions for events configuration.

Method SendEventEx

Syntax

```
Object.SendEventEx(EventName As String, mapHndl As Integer)
```

Parameters

EventName
The name of the event to send.

mapHndl
The handle of a mapping table returned by a call to CreateMap that contains all the keys required to set the event attributes.

Description

This method sends the event named EventName and specifies its attributes.

Remarks

The event named EventName must be previously defined through a DefineEvent in the SetDefaultConfiguration Subroutine. The mapping table associated to the handle mapHndl must contain, as keys, all the attributes defined for the given event.

Error codes

TMWSERVICE_E_MAP_KEY_NOT_FOUND
TMWSERVICE_E_ANALYZER_EVENT_NOT_GOT
TMWSERVICE_E_SPAWN_EVENT_FAILED
TMWSERVICE_E_EVENT_NOT_DECLARED
TMWSERVICE_E_EVENT_PROP_NOT_FOUND
TMWSERVICE_E_EVENT_INDICATION_FAILED S_OK.

Logging

The following methods help you to express functions for data logging.

Method LogInstEx

Syntax

```
Object.LogInstEx(contextName As String, Resource As String, mapHndl As Integer)
```

Parameters

context
The logging context.

pResource
The resource that the attributes refer to.

mapHndl
The handle of a mapping table returned by a call to CreateMap that contains all the keys required to set the logging attributes.

Description
This method logs the attributes of the resource named Resource in the given context.

Remarks
The context and the resource must be previously defined through a DefineLogInst in the SetDefaultConfiguration Subroutine. The mapping table associated to the handle mapHndl must contain as keys all the attributes defined for the given logging context.

Error codes
TMWSERVICE_E_MAP_KEY_NOT_FOUND
TMWSERVICE_E_ANALYZER_EVENT_NOT_GOT
TMWSERVICE_E_SPAWN_EVENT_FAILED
TMWSERVICE_E_EVENT_NOT_DECLARED
TMWSERVICE_E_EVENT_PROP_NOT_FOUND
TMWSERVICE_E_EVENT_INDICATION_FAILED S_OK.

Utilities

The following methods help you to express functions for tracing activity.

MethodTrace

Syntax

```
Object.Trace(LogLevel As Integer, Message As String)
```

Parameters

LogLevel
The log level of the message (0..3, 0 is the highest).

Message
The Message to Trace.

Description
Writes the Message in the DM for Window Trace if the LogLevel is greater than or equal to the DMW Trace level.

Remarks

Call this method for debugging. To change the trace level from the Tivoli Monitoring server, type the following command:

```
wdmtrceng
```

For more information, see *Tivoli Monitoring User's Guide*.

Error code

S_OK.

Method GetInterp

Syntax

```
Object.GetInterp()As String
```

Description

Returns the interp of the workstation on which the resource model is running.

Error codes

w32-x86 on Windows NT and Windows 2000 platforms; *aix-r1; hpux10;linux-ix86;linux-s390;solaris2* on UNIX platforms.

Method Shell

Syntax

```
Object.Shell(shell As String) As String
```

Parameters

shell The command line to launch the new process.

Returns

The standard output of the launched process.

Description

The Shell method is used to run a new program on the monitored host and to retrieve its output.

Remarks

It is possible to launch customized scripts that are available on the endpoint by issuing, for example,

```
perl myperl.pl
```

The launched process must return within 60 seconds, otherwise an error is generated.

Error codes

TMWSERVICE_E_NO_INTERP_SUPPORT TMWSERVICE_E_NO_DATA TMWSERVICE_E_IMPLIED_ERROR TMWSERVICE_E_ERRORVALUE_SCRIPT_ERROR S_OK.

Mapping Tables

Method CreateMap

Syntax

```
Object.CreateMap( ) As Integer
```

Returns

The handle of a new mapping table.

Description

This method creates a new mapping table and returns a handle to it.

Remarks

To avoid memory leaks, call the DestroyMap method when the mapping table is no longer required.

Error code

S_OK.

Method SetMapNumElement

Syntax

```
Object.SetMapNumElement(hndl As Integer, key As String, val As Double)
```

Parameters

hndl The handle of a mapping table returned by a call to CreateMap.

key The key to store.

val The numeric value to store.

Description

This method inserts the *key-val* pair in the mapping table associated to the given handle.

Error codes

TMWSERVICE_E_MAP_HANDLE_NOT_FOUND S_OK.

Method SetMapStrElement

Syntax

```
Object.SetMapStrElement(hndl As Integer, key As String, val As String)
```

Parameters

hndl The handle of a mapping table returned by a call to CreateMap.

key The key to store.

val The string value to store.

Description

This method inserts the *key-val* pair in the mapping table associated to the given handle.

Error codes

TMWSERVICE_E_MAP_HANDLE_NOT_FOUND S_OK.

Method GetMapNumValue

Syntax

```
Object.GetMapNumValue (hndl As Integer, key As String) As Double
```

Parameters

hndl The handle of a mapping table returned by a call to CreateMap.

key The key to lookup.

Returns

The numeric value associated to the given key.

Description

This method retrieves the value associated to the key in the mapping table linked with the given handle.

Error codes

TMWSERVICE_E_MAP_HANDLE_NOT_FOUND
TMWSERVICE_E_MAP_KEY_NOT_FOUND S_OK.

Method GetMapStrValue

Syntax

```
Object.GetMapStrValue(hndl As Integer, key As String) As String
```

Parameters

hndl The handle of a mapping table returned by a call to CreateMap.

key The key to lookup.

Returns

The string value associated to the given key.

Description

This method retrieves the value associated to the key in the mapping table linked to the given handle.

Error codes

TMWSERVICE_E_MAP_HANDLE_NOT_FOUND
TMWSERVICE_E_MAP_KEY_NOT_FOUND S_OK.

Method RemoveMapElement

Syntax

```
Object.RemoveMapElement(hndl As Integer, key As String)
```

Parameters

hndl The handle of a mapping table returned by a call to CreateMap.

key The key to remove from the mapping table.

Description

This method removes the value associated to the key in the mapping table linked to the given handle.

Error codes

TMWSERVICE_E_MAP_HANDLE_NOT_FOUND S_OK.

Method RemoveMapAll

Syntax

```
Object.RemoveMapAll(hndl As Integer)
```

Parameters

hndl The handle of a mapping table returned by a call to CreateMap.

Description

This method removes all the elements contained in the mapping table associated to the handle hndl, but it does not destroy the mapping table.

Remarks

This method does not free all the resources used by a mapping table. If you want to make them available, call the DestroyMap method.

Error codes

TMWSERVICE_E_MAP_HANDLE_NOT_FOUND S_OK.

Method ExistsMapElement

Syntax

```
Object.ExistsMapElement(hndl As Integer, key As String) As Boolean
```

Parameters

hndl The handle of a mapping table returned by a call to CreateMap.

key The key to check whether an element exists.

Returns

TRUE if the key exists in the given mapping table, or FALSE if it does not exist.

Description

This method checks whether the given key is contained in the mapping table associated to the given handle.

Error codes

TMWSERVICE_E_MAP_HANDLE_NOT_FOUND S_OK.

Method DestroyMap

Syntax

```
Object.DestroyMap(hndl As Integer)
```

Parameters

hndl The handle of a mapping table returned by a call to CreateMap.

Remarks

Call the DestroyMap to free the resources used by a mapping table.

Description

This method destroys a the mapping table associated to the handle hndl.

Error codes

TMWSERVICE_E_MAP_HANDLE_NOT_FOUND S_OK.

Advanced Object Methods

This section contains a description of the advanced methods of the TMWService Object. You should never call these methods directly, because the calls to these methods are always generated by the workbench

General Settings

The following methods help you to express functions for resource model configuration.

Method SetModelName

Syntax

```
Object.SetModelName(ModelName As String)
```

Parameters

ModelName
The name of the resource model.

Description
Sets the resource model name.

Remarks
Do not call this method outside the SetDefaultConfiguration(...) function.

Error code
S_OK.

Method SetCycleTime

Syntax

```
Object.SetCycleTime(CycleTime As Double)
```

Parameters

CycleTime
The cycle time of the resource model.

Description
Sets the resource model cycle time.

Remarks
Do not call this method outside the SetDefaultConfiguration(...) function.

Error code
S_OK.

Dynamic Model

The following methods help you to express functions for dynamic model configuration.

Method DefineClass

Syntax

```
Object.DefineClass(Source As String, AliasName As String,_
RealClassName As String, WhereClause As String,_
NumProps As String, StrProps As String, SortType As String,_
SortField As String, Top as long, Every as long)
```

Parameters

Source
The data source name (It can be only ″CIM″.)

AliasName
An alias for the class name.

RealClassName
The class full path.

WhereClause
A where clause used to filter the instances.

NumProps
Comma-separated values with the names of class NUMERIC attributes.

StrProps
Comma-separated values with the names of class STRING attributes.

SortType
Used to sort the collected instances. It can be set to **Ascending**, **Descending**, or **None**.

SortField
A NUMERIC field used for sorting the collected instances.

Top Used only when the instances are sorted, to specify the maximum number of instances to collect. Set to 0 to get all the instances.

Every
Specifies the number of cycle times that have to elapse between two successive data collections. The default is 1.

Description
Adds a new data class to the resource model dynamic model.

Remarks
The *ClassName* must be previously defined in the Dynamic Model dialog.

Error codes
S_OK; TMWSERVICE_E_CLASS_NAME_NOT_VALID; TMWSERVICE_E_SOURCE_NOT_SUPPORTED.

Method DefineCimClassAsync

Syntax

```
Object.DefineClassAsync(AliasName As String,_
RealClassName As String,_
EvType As String, NumProps As String, StrProps As String)
```

Parameters

AliasName
An alias for the class name.

RealClassName
The class full path.

EvType
Used to specify the kind of asynchronous collection. It can be set to **InstanceCreation**, **InstanceModification**, or **InstanceDeletion**.

NumProps
Comma-separated values with the names of class NUMERIC attributes.

StrProps
Comma-separated values with the names of class STRING attributes.

Description
Adds a new data class to the resource model dynamic model.

Error codes
S_OK; TMWSERVICE_E_CLASS_NAME_NOT_VALID.

Method RemoveClass

Syntax

```
Object.RemoveClass(ClassName As String)
```

Parameters

ClassName
The name of the class.

Description
Removes the data class named *ClassName* from the resource model dynamic model.

Error codes
S_OK; TMWSERVICE_E_PROPERTY_NOT_FOUND.

Method CollectData

Syntax

```
Object.CollectData()
```

Description
Collects data for all the classes belonging to the resource model dynamic model.

Remarks
This method is called from Tivoli Monitoring engine before the VisitTree function is called. By default, it is called every cycle time, unless otherwise specified in the **Every** option of the Dynamic Model dialog. Do not call this method. If you invoke this method, a new collection of the data defined in the resource model starts. So if, for example, you are in a loop, you risk changing data during the analysis

Error codes
S_OK; TMWSERVICE_E_COLLECTOR_UNABLE_TO_USE_PROV; TMWSERVICE_E_COLLECTOR_CRITICAL_ERROR; TMWSERVICE_E_COLLECTOR_PROV_NOT_READY.

Method CollectClassData

Syntax

```
Object.CollectClassData(ClassName As String)
```

Parameters

ClassName
The name of the class.

Description
Collects data only for the specified class.

Remarks
Do not call this method if you are analyzing the defined class. Invoking this method, a new collection of the data defined in the resource model starts. So if, for example, you are in a loop, you risk changing data during the analysis.

Error codes
S_OK; TMWSERVICE_E_COLLECTOR_UNABLE_TO_USE_PROV; TMWSERVICE_E_COLLECTOR_CRITICAL_ERROR; TMWSERVICE_E_COLLECTOR_PROV_NOT_READY.

Method DefineDMNumProbe

Syntax

```
Object.DefineDMNumProbe(key As String, numOfArgs As Long)
```

Parameters

key A key that uniquely identifies the monitoring source, always in the form *CollectionName.MonitorName.*

numOfArgs
The number of arguments required by the monitor.

Description
This method adds to the dynamic model a Tivoli Distributed Monitoring (Classic Edition) monitoring source that returns a numeric value.

Error code
S_OK.

Method DefineDMStrProbe

Syntax

```
Object.DefineDMStrProbe(key As String, numOfArgs As Long)
```

Parameters

key A key that uniquely identifies the monitoring source, always in the form *CollectionName.MonitorName*

numOfArgs
The number of arguments required by the monitor.

Description
This method adds to the dynamic model a Tivoli Distributed Monitoring (Classic Edition) monitoring source that returns a string value.

Error code
S_OK.

Thresholds

The following methods help you to express functions for threshold configuration.

Method DefineThreshold

Syntax

```
Object.DefineThreshold(ThName As String, value As Double)
```

Parameters

ThName
The name of a new threshold.

value The threshold default value.

Description
Defines a new threshold.

Remarks
This method is called inside the <<THRESHOLDS_INFO>>...<<\THRESHOLDS_INFO>> tags after you have defined a threshold in the Thresholds dialog.

Error codes

S_OK; TMWSERVICE_E_THRESHOLD_NAME_NOT_VALID.

Parameters

The following methods help you to express functions for parameters configuration.

Method DefineStrParameter

Syntax

```
Object.DefineStrParameter(ParamName As String, values As String)
```

Parameters

ParamName
The name of a new STRING parameter.

values Comma-separated values of the parameter default values.

Description

Defines a new STRING parameter.

Remarks

This method is called inside the <<PARAMETERS_INFO>>...<<\PARAMETERS_INFO>> tags after you have defined a parameter in the Parameters dialog.

Error codes

S_OK; TMWSERVICE_E_PARAMETER_NAME_NOT_VALID.

Method DefineNumParameter

Syntax

```
Object.DefineNumParameter(ParamName As String, values As String)
```

Parameters

ParamName
The name of a new NUMERIC parameter

values Comma-separated values of the parameter default values.

Description

Defines a new NUMERIC parameter.

Remarks

This method is called inside the <<PARAMETERS_INFO>>...<<\PARAMETERS_INFO>> tags after you have defined a parameter in the Parameters dialog.

Error codes

S_OK; TMWSERVICE_E_PARAMETER_NAME_NOT_VALID.

Method AddStrParameter

Syntax

```
Object.AddStrParameter(ParamName As String, Value As String)
```

Parameters

ParamName
The name of a STRING parameter.

Value A STRING value.

Description

Adds the given Value at the end of the parameter named *ParamName*.

Remarks

This method is called after you have defined a new parameter in the Parameters dialog.

Error codes

S_OK; TMWSERVICE_E_PARAMETER_NOT_DEFINED; TMWSERVICE_E_PARAMETER_BAD_INDEX.

Method AddNumParameter

Syntax

```
Object.AddNumParameter(ParamName As String, Value As Double)
```

Parameters

ParamName

The name of a NUMERIC parameter.

Value A NUMERIC value.

Description

Adds the given Value at the end of the parameter named *ParamName*.

Remarks

This method is called after you have defined a new parameter in the Parameters dialog.

Error codes

S_OK; TMWSERVICE_E_PARAMETER_NOT_DEFINED; TMWSERVICE_E_PARAMETER_BAD_INDEX.

Method RemoveStrParameter

Syntax

```
Object.RemoveStrParameter(ParamName As String, idx As Long)
```

Parameters

ParamName

The name of a STRING parameter.

idx The index of a parameter value.

Description

Removes the value contained at the given index and shifts down all the values after the removed one.

Remarks

idx counter goes from 0 to NumOfInst -1.

Error codes

S_OK; TMWSERVICE_E_PARAMETER_NOT_DEFINED; TMWSERVICE_E_PARAMETER_BAD_INDEX.

Method RemoveNumParameter

Syntax

```
Object.RemoveNumParameter(ParamName As String, idx As Long)
```

Parameters

ParamName
The name of a NUMERIC parameter.

idx The index of a parameter value.

Description
Removes the value contained at the given index and shifts down all the values after the removed one.

Remarks
idx counter goes from 0 to NumOfInst -1.

Error codes
S_OK; TMWSERVICE_E_PARAMETER_NOT_DEFINED;
TMWSERVICE_E_PARAMETER_BAD_INDEX.

Method ReplaceStrParameter

Syntax

```
Object.ReplaceStrParameter(ParamName As String,_
idx As Long, Value As String)
```

Parameters

ParamName
The name of a STRING parameter.

idx The index of a parameter value.

Value A STRING value.

Description
Replaces the value contained at the given index with the new value.

Remarks
idx counter goes from 0 to NumOfInst -1.

Error codes
S_OK; TMWSERVICE_E_PARAMETER_NOT_DEFINED
TMWSERVICE_E_PARAMETER_BAD_INDEX

Method ReplaceNumParameter

Syntax

```
Object.ReplaceNumParameter(ParamName As String,_
idx As Long, Value As Double)
```

Parameters

ParamName
The name of a NUMERIC parameter.

idx The index of a parameter value.

Value A NUMERIC value.

Description

Replaces the value contained at the given index with the new value.

Remarks

idx counter goes from 0 to NumOfInst -1.

Error codes

S_OK; TMWSERVICE_E_PARAMETER_NOT_DEFINED; TMWSERVICE_E_PARAMETER_BAD_INDEX.

Events

The following methods help you to express functions for events configuration

Method DefineEvent

Syntax

```
Object.DefineEvent(EventName As String,_
NumAttrs As String, StrAttrs As String)
```

Parameters

EventName

The name of a new event.

NumAttrs

Comma-separated values with the names of the NUMERIC attributes of the event.

StrAttrs

Comma-separated values with the names of the STRING attributes of the event.

Description

Defines a new Event.

Remarks

This method is called inside the <<EVENTS_INFO>>...<<\EVENTS_INFO>> tags after you have defined an event in the Events dialog.

Error codes

S_OK; TMWSERVICE_E_EVENT_REDEFINITION

Logging

The following methods help you to express functions for data logging.

Method DefineLogInst

Syntax

```
Object.DefineLogInst(Context As String,_
Resource As String_
Keys As String, NumAttrs As String, StrAttrs As String)
```

Parameters

Context

The name of a new logging context.

Resource

The resource name.

Keys The resource key properties.

NumAttrs
Comma-separated values with the names of resource NUMERIC attributes.

StrAttrs
Comma-separated values with the names of resource STRING attributes.

Description
Defines a logging context.

Remarks
This method is called inside the <<LOGGING_INFO>>...<<\LOGGING_INFO>> tags after you have defined a context in the Logging dialog.

Error codes
S_OK; TMWSERVICE_E_INST_EVENT_REDEFINITION

Generic Functions

The following methods help you to express functions for generic activities.

Method EndVisit

Syntax

```
Object.EndVisit()
```

Description
Is called at the end of each cycle, to perform clean up operations.

Error code
S_OK;

Method Dispose

Syntax

```
Object.Dispose()
```

Description
Is called at the end of the monitoring loop, to perform clean up operations.

Error code
S_OK.

Method GetProfileName

Syntax

```
Object.GetProfileName() As String
```

Returns
The profile name.

Description
Returns the name of the profile to which the resource model belongs.

Error code
S_OK.

Method GetGlobalCounter

Syntax

```
Object.GetGlobalCounter() As String
```

Returns

The value of the current monitoring loop.

Description

Returns the value of the current monitoring loop. When a resource model starts its monitoring this value is set to 0, and it is increased by one after each successful call to the VisitTree function.

Error code

S_OK.

Deprecated Methods

Some methods used with Tivoli Distributed Monitoring (Advanced Edition), Version 4.1 are being replaced by new ones.

The SendEvent method is deprecated. To send an event, use the SendEventEx method.

The LogInst method is deprecated. To log the attributes of a resource, use the LogInstEx method, instead.

Exceptions

This table contains a list of exceptions you may encounter, with the corresponding message.

Exceptions	Description	Remarks
0x80041001	TMWSERVICE_E_FAILED	Generic Error.
0x80041002	TMWSERVICE_E_UNABLE_TO_CONNECT_WMI	It is not possible to get access to WMI Service.
0x80041003	TMWSERVICE_E_ANLYZER_EVENT_NOT_FOUND	The TMW_AnalyzerEvent cannot be found, the MOF file compilation may have failed.
0x80041004	TMWSERVICE_E_CANNOT_GET_SINK	It is not possible to get the Event Sink.
0x80041005	TMWSERVICE_E_CANNOT_RESTART_COLLECTOR	The data collector cannot be restarted.
0x80041006	TMWSERVICE_E_ANLYZER_EVENT_NOT_GOT	The definition of the TMW_AnalyzerEvent was not found in the CIM Repository.
0x80041007	TMWSERVICE_E_SPAWN_EVENT_FAILED	It is not possible to spawn a new instance of the event.
0x80041008	TMWSERVICE_E_EVENT_NOT_DECLARED	The event or the logging context has not been defined.
0x80041009	TMWSERVICE_E_EVENT_INDICATION_FAILED	WMI errors occurred while the TMW_AnalyzerEvent, or the TMW_InstEvent was being sent.
0x8004100a	TMWSERVICE_E_EVENT_WRONG_NUM_OF_ARGS	Wrong number of arguments.
0x8004100b	TMWSERVICE_E_CIM_CLASS_NOT_FOUND	The class has not been defined.
0x8004100c	TMWSERVICE_E_THRESHOLD_NAME_NOT_VALID	The name for the threshold is not allowed.
0x8004100d	TMWSERVICE_E_THRESHOLD_NAME_NOT_DEFINED	The threshold has not been defined.
0x8004100e	TMWSERVICE_E_EVENT_REDEFINITION	The event was already defined.

Exceptions	Description	Remarks
0x8004100f	TMWSERVICE_E_PARAMETER_NOT_DEFINED	The parameter has not been defined.
0x80041010	TMWSERVICE_E_PARAMETER_NAME_NOT_VALID	The name for the threshold is not allowed.
0x80041011	TMWSERVICE_E_CLASS_NAME_NOT_VALID	The name for data class is not allowed.
0x80041012	TMWSERVICE_E_CIM_PROPERTY_NOT_FOUND	The property does not belong to the given class.
0x80041013	TMWSERVICE_E_PARAMETER_BAD_INDEX	The parameter index is out of the bound.
0x80041014	TMWSERVICE_E_COLLECTOR_UNABLE_TO_USE_PROV	The provider is not able to collect data.
0x80041015	TMWSERVICE_E_COLLECTOR_CRITICAL_ERROR	It is not possible to collect data. The class name might be wrong, or the query invalid.
0x80041016	TMWSERVICE_E_COLLECTOR_PROV_NOT_READY	The provider is not able to collect data; it may be busy.
0x80041017	TMWSERVICE_E_DEPENDENCY_NOT_FOUND	It is not possible to find the dependency file.
0x80041018	TMWSERVICE_E_CANNOT_SAVE_WORKSPACE	It is not possible to save the workspace.
0x80041019	TMWSERVICE_E_CANNOT_LOAD_WORKSPACE	It is not possible to load the workspace.
0x8004101a	TMWSERVICE_E_INST_EVENT_NOT_GOT	The definition of the TMW_InstEvent was not found in the CIM Repository.
0x8004101b	TMWSERVICE_E_INST_EVENT_NOT_FOUND	The TMW_InstEvent cannot be found. The MOF file compilation might have failed.
0x8004101e	TMWSERVICE_E_INST_EVENT_REDEFINITION	The logging context was already defined.
0x80041020	TMWSERVICE_E_NOT_VALID_WORKSPACE_FILE	It is not possible to load the workspace because of a bad workspace file format.
0x80041021	TMWSERVICE_E_ACTION_RULE_NOT_FOUND	The action rule was not found.
0x80041022	TMWSERVICE_E_ACTION_NOT_FOUND	The action was not found.
0x80041023	TMWSERVICE_E_SOURCE_NOT_SUPPORTED	The data source is not allowed.
0x80041024	TMWSERVICE_E_PROBE_NOT_FOUND	The Tivoli Distributed Monitoring (Classic Edition) monitoring source has not been defined.
0x80041025	TMWSERVICE_E_PROBE_NOT_LOAD	The Tivoli Distributed Monitoring (Classic Edition) monitoring source implementation is not available.
0x80041026	TMWSERVICE_E_NO_INTERP_SUPPORT	The Tivoli Distributed Monitoring (Classic Edition) monitoring source implementation is not available for the given platform.
0x80041027	TMWSERVICE_E_NO_DATA	The Tivoli Distributed Monitoring (Classic Edition) monitoring source implementation has returned an empty output.
0x80041028	TMWSERVICE_E_IMPLIED_ERROR	The Tivoli Distributed Monitoring (Classic Edition) monitoring source implementation is getting an error due to bad arguments.

Exceptions	Description	Remarks
0x80041029	TMWSERVICE_E_ERRORVALUE_SCRIPT_ERROR	The Tivoli Distributed Monitoring (Classic Edition) monitoring source implementation contains an internal script error.
0x8004102a	TMWSERVICE_E_PROBE_WRONG_ARGS_NUM	A wrong number of arguments has been specified in a call to a Tivoli Distributed Monitoring (Classic Edition) monitoring source.
0x8004102b	TMWSERVICE_E_MAP_HANDLE_NOT_FOUND	The handle of the mapping table is invalid.
0x8004102c	TMWSERVICE_E_MAP_KEY_NOT_FOUND	The given key is not contained in the mapping table associated to the given handle.
0x8004102d	TMWSERVICE_E_EVENT_PROP_NOT_FOUND	A property required for sending an event or for logging a call is not contained in the mapping table associated to the given handle.
0x8004102e	TMWSERVICE_E_CONTEX_PROP_NOT_FOUND	The context property is not available.
0x8004102f	TMWSERVICE_E_PARAMETER_IS_EMPTY	The parameter doesn't contain any value.
0x80041030	TMWSERVICE_E_NOT_WORKSPACE_TOO_NEW	The resource model workspace is too new for the current version of the workbench.

B

Examples of Resource Model Creation

This Appendix describes examples related to the creation of resource models.

Note: The resource models considered in this appendix are intended for Windows platforms and the decision tree scripts are written in Visual Basic.

Aspects Considered

The following examples of resource models are presented with focus on specific settings and options. Each resource model shows some specific functions:

In the **Processor Monitor** example, the following items are treated

- Dynamic Model
  - Sorting
  - Use of the WHERE clause
  - Selective data collection (number of instances)
- Events
  - Use of attributes
  - Aggregation process
  - Clearing event option
  - Use of actions (non static method)
- Thresholds
- Code example (Decision Tree Script)

In the **Parametric Event Log** resource model, the following items are treated

- Dynamic Model
- Asynchronous collection
  - Instance creation
- Events
  - Use of attributes
  - Aggregation process
  - Send to TEC option
- Parameters

- Boolean list
- Choice list
- Numeric list
- String list

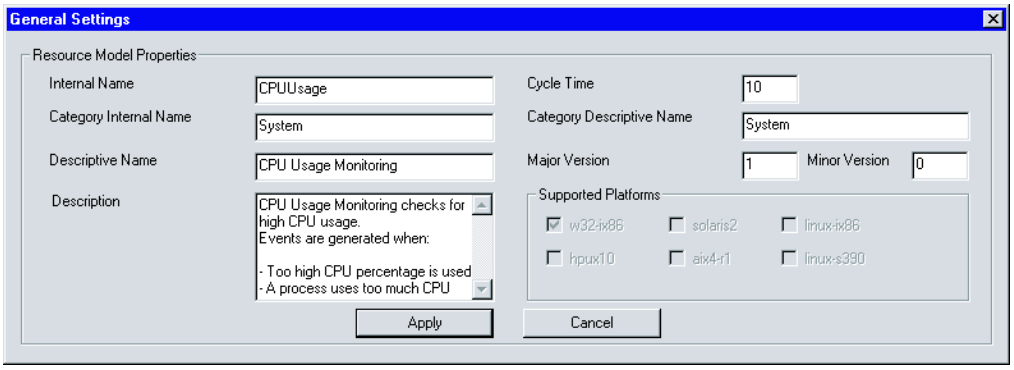
- Use of logging
- Code example (Decision Tree Script)

Processor Monitor

The processor resource model monitors CPU usage, by collecting data about the most CPU-consuming process, and CPU percentage used by the most consuming processes. This resource model focuses mainly on dynamic model, threshold and event configurations, and includes a recovery action.

General Settings Dialog

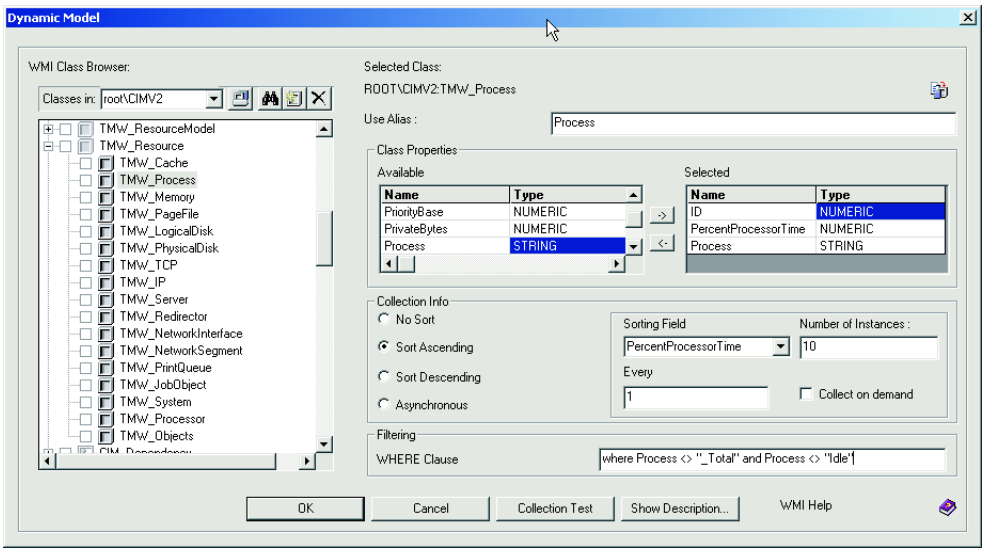
The following panel shows the General Settings dialog.



It specifies that this resource model collects data about CPU usage considering both processor and processes. Data collection is performed every 10 seconds.

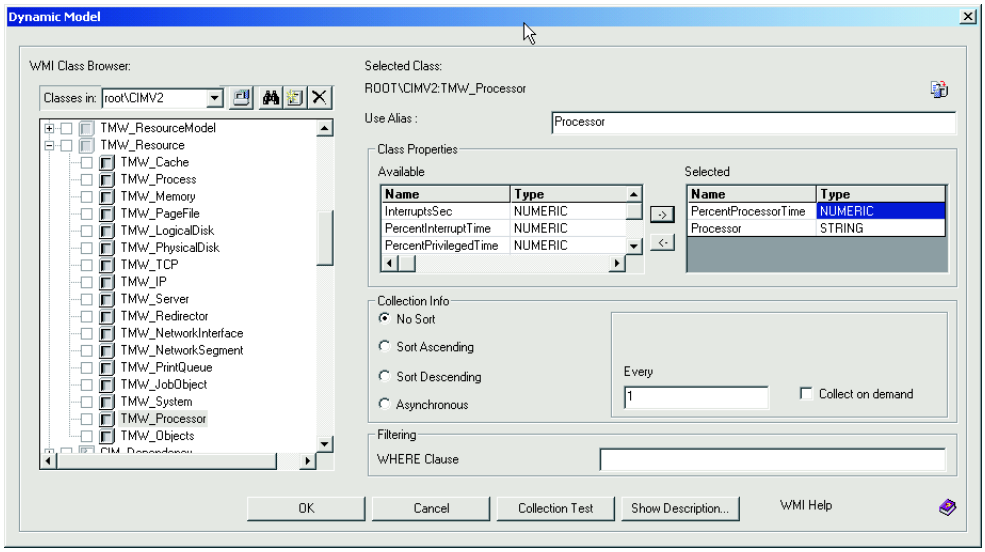
Dynamic Model Dialogs

The Dynamic Model dialog of the following panel shows that the selected class is the *CIMV2:TMW_Process class*.



The properties associated with the selected class contain both numeric and string values. They provide information about the process ID, name and CPU usage. Collected data is sorted in ascending order, on the basis of CPU usage percentage. Only the ten most consuming processes are considered. Data collection is performed at every cycle time. To filter the collected data, a **WHERE Clause** specifies that the application does not have to monitor Total and Idle processes. They are Windows automatic processes whose CPU usage is of no interest in our resource model. For more information about filtering, see “Data Filtering” on page 6.

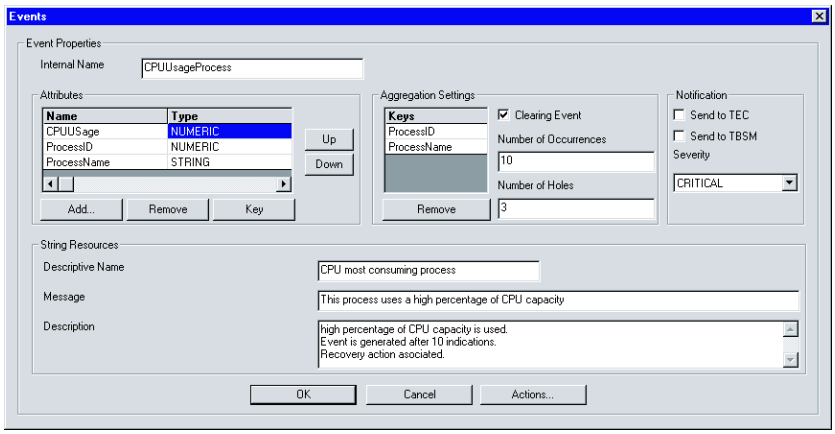
The Dynamic Model dialog in the following figure shows that the selected class is *CIMV2:TMW_Processor class*.



The properties associated with the selected class contain both numeric and string values. They provide information about processor type and usage percentage. Collected data is not sorted. Data collection is performed at every cycle time.

Events Dialog

The Events dialog of the following figure is associated with process data collection.

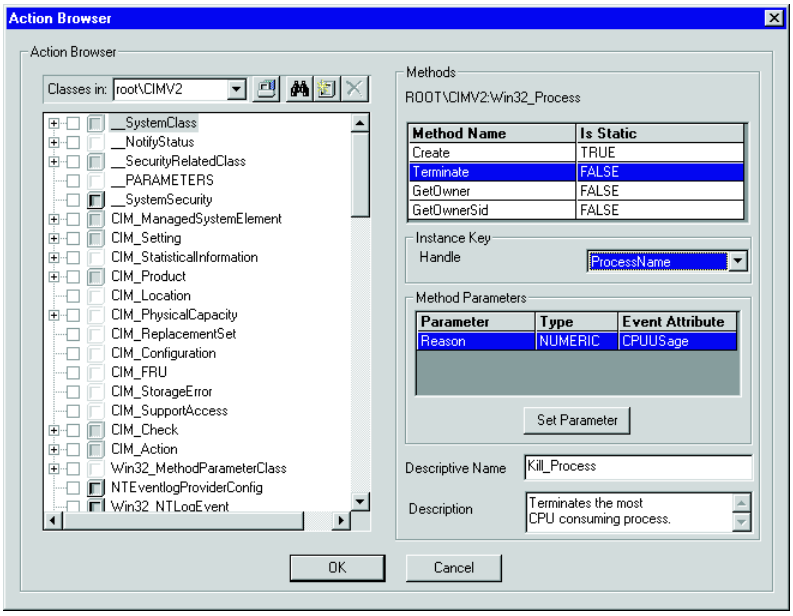


The attributes considered include string and numeric values. They provide information about CPU usage percentage, process name, and process ID. Process ID, is the key attribute that drives the aggregation process, therefore it appears also in the **Aggregation Settings** text box. An event is generated after ten indications (number of occurrences) and three holes.

The event notification is not sent to the Tivoli Enterprise Console server. The **Clearing Event** check box is selected, so a clearing event is sent when the conditions that caused the event are no longer present.

Actions Dialog

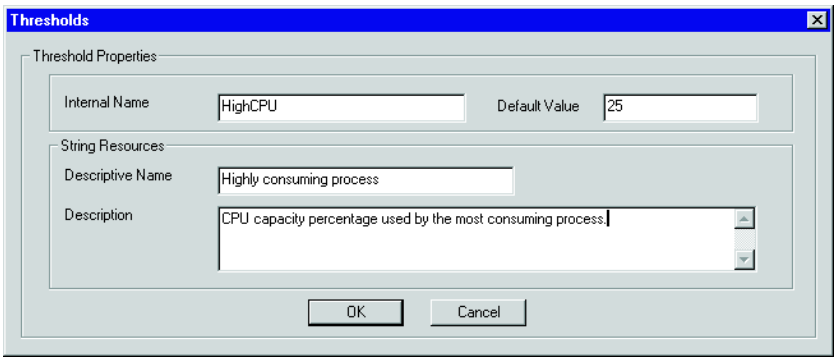
The Action Browser dialog of the following figure shows the action associated to the CPU Usage Process event. The action is automatically performed when the event is generated.



The selected method is not static and it terminates the process that is using most of the CPU capacity, according to the values specified in the Events and Thresholds dialogs. The Process name is the key that associates the terminate method to the actual process that has to be terminated. The information contained in the **Method Parameters** group specifies that this method is triggered on the basis of the numeric value represented by the CPU usage of the considered process. For more information about actions and methods, see “Actions” on page 7.

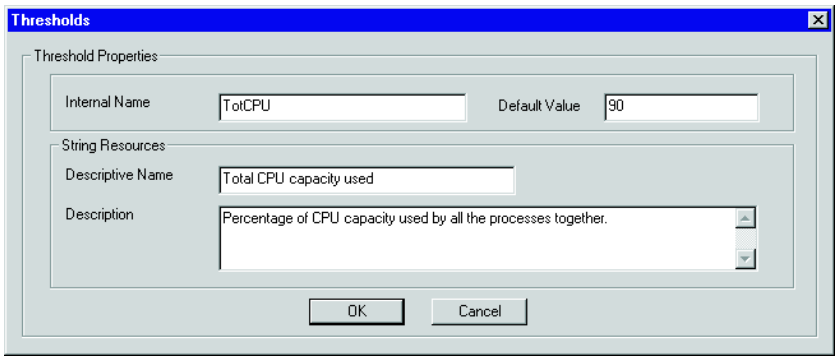
Thresholds Dialogs

The Thresholds dialog of the following figure shows the threshold set for the *Process* dynamic model.



The **Default Value** represents a maximum limit of CPU capacity that can be used by a single process. If a process uses more than 25% of CPU capacity, an indication is triggered. The Thresholds dialog of the following figure shows the threshold set for the *Processor*

dynamic model.



The **Default Value** represents the maximum limit of CPU that can be used by all the processes together. If overall CPU usage exceeds 90%, an indication is triggered.

Decision Tree Script

All the settings implemented in the previous dialogs appear automatically in the first parts of the decision tree script, as shown in the following code sample:

```
Public Sub Main ()
.....................
Public Function SetDefaultConfiguration (Svc As Object) As Long

' General info section
'<<GENERAL_INFO>>
Svc.SetModelName "CPUUsage"
Svc.SetProfileName "277210437"
Svc.SetCycleTime 10
'<<\GENERAL_INFO>>

' Thresholds section
'<<THRESHOLDS_INFO>>
Svc.DefineThreshold "TotCPU", 90
Svc.DefineThreshold "HighProcessCPU", 25
'<<\THRESHOLDS_INFO>>

' Parameters section
'<<PARAMETERS_INFO>>
'<<\PARAMETERS_INFO>>

' Dynamic model section
'<<DATA_INFO>>
Svc.DefineClass "CIM", "Process", "ROOT\CIMV2:TMW_Process",_
"where Process <> ""_Total"_
" and Process <> ""Idle""",_
"ID,PercentProcessorTime", "Process", _
"Ascending", "PercentProcessorTime", 10, 1
Svc.DefineClass "CIM", "Processor", "ROOT\CIMV2:TMW_Processor", "", _
"PercentProcessorTime", "Processor",_
"None", "", 0, 1
'<<\DATA_INFO>>

' Event definition section
'<<EVENTS_INFO>>
Svc.DefineEvent "HighCPUUsage",_
"CurrCpuUsage,WorstProcessID,WorstProcessCPUUsage",_
"WorstProcessName"
Svc.DefineEvent "CPUUsageProcess",_
```

```
"CPUUSage,ProcessID", "ProcessName"
'<<\EVENTS_INFO>>

' Logging definition section
'<<LOGGING_INFO>>
'<<\LOGGING_INFO>>

' Place your additional intializing code below


SetDefaultConfiguration = 0

End Function
```

Visit Tree Function

The following code example contains the visit tree function that is called cyclically every cycle time. This function implements the monitoring algorithm.

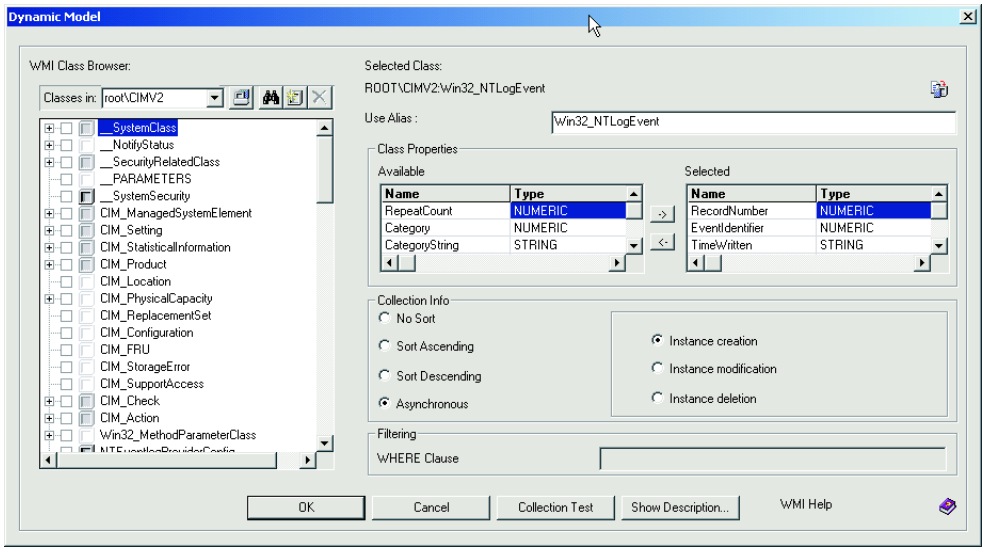
```
Public Function VisitTree(Svc As Object) As Long
Dim i As Long, n As Long
If Svc.GetNumProperty _
"Processor", 0 , "PercentProcessorTime" ) > Svc.GetThreshold ("TotCPU" )  Then

Svc.SendEvent "HighCPUUsage", _
Svc.GetNumProperty ("Processor", 0 , "PercentProcessorTime") , _
Svc.GetNumProperty ("Process", 0 , "ID"), _
Svc.GetNumProperty ("Process", 0 ,_
"PercentProcessorTime"), _
Svc.GetStrProperty ("Process", 0 , "Process")

End If

i=0
n= Svc.GetNumOfInst ("Process")
Do While i < n
If Svc.GetNumProperty ("Process", i ,_
"PercentProcessorTime" )  > Svc.GetThreshold_
("HighProcessCPU" )  Then
Svc.SendEvent "HighCPUUsageProcess", _
Svc.GetNumProperty ("Process", i ,_
"PercentProcessorTime"), _
Svc.GetNumProperty ("Process", i , "ID"), _
Svc.GetStrProperty ("Process", i , "Process")

i=i+1

Else
i=n

End If
Loop

VisitTree = 0

End Function
```

Parametric Event Log

The Parametric Event Log resource model monitors Windows EventLog, and collects data about each new instance created in the log. This resource model focuses mainly on the Parameters settings and shows a sample of all types of available parameter lists. For more details on this topic, see “Parameters” on page 8.

Dynamic Model Dialog

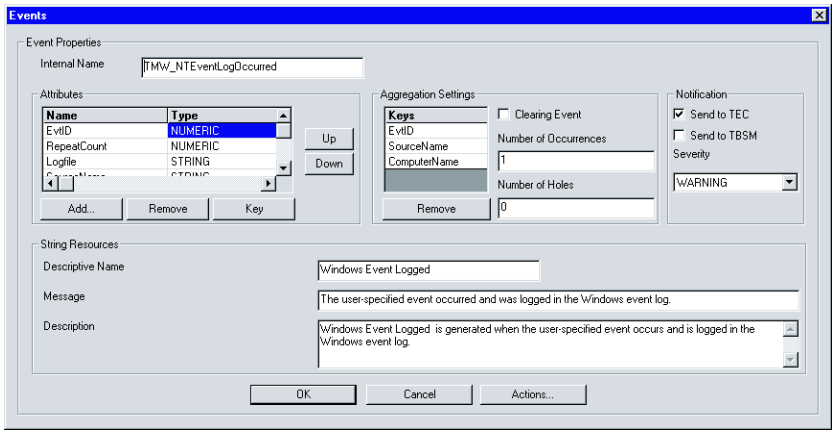
The Dynamic Model of the following figure shows that the selected class is *Win32_NT LogEvent*.



The properties related to the selected class are both numeric and string values. The kind of data collection selected is asynchronous. For more information on asynchronous data collection, see “Asynchronous Data Collection” on page 6.

Events Dialog

The Events dialog is depicted in the following figure.

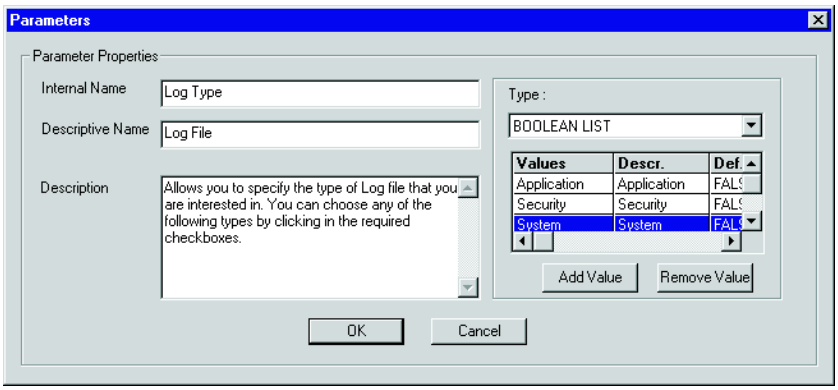


The attributes considered include string and numeric values. They provide information about the time when the events have been generated and written in the log, the degree of severity of the event, the workstation where the events occur, and the event IDs. Event ID, SourceName and ComputerName are also key attributes that drive the aggregation process, therefore they appear also in the **Aggregation Settings** text box. An event is generated after one indication (number of occurrences) and no holes. The event notification is sent to the Tivoli Enterprise Console server, with the corresponding message and a severity level indication (WARNING). The **Clearing Event** check box is not selected, so a clearing event is not sent when the conditions that caused the event are no longer present.

Parameters Dialogs

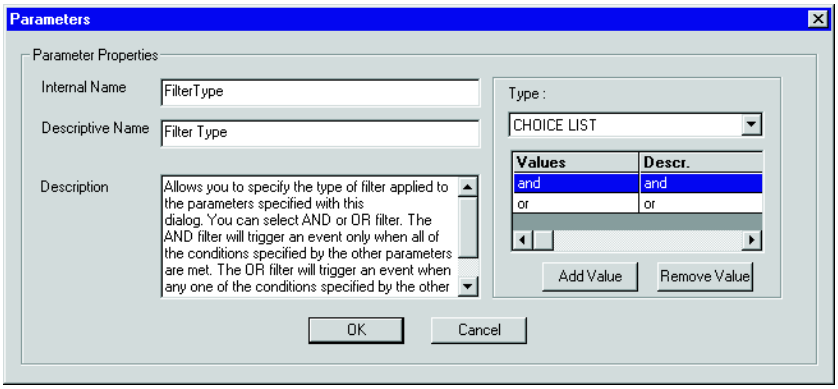
The following figures show several Parameters dialogs, providing an example for each type of list. Depending on the kind of list selected, the collected results will be displayed differently on Tivoli Monitoring dialogs. For more information about parameter lists, see “Parameters” on page 8.

The following dialog depicts a boolean list.



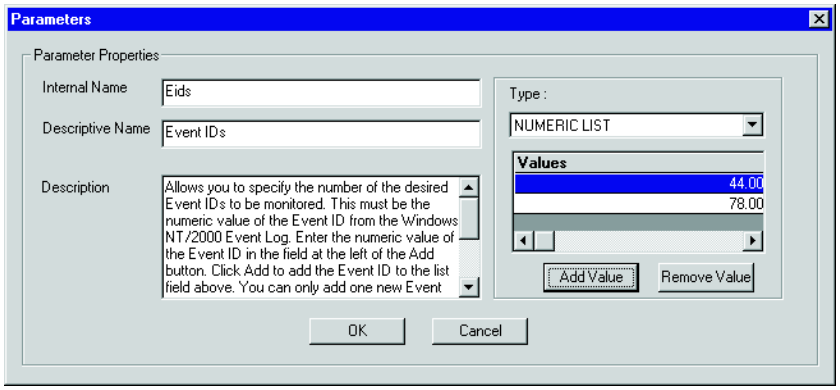
The boolean list contains different types of log files. Application, Security, and System log files are considered, but they are all set to FALSE as default values. Later, if required, they can be set to TRUE from the Tivoli Monitoring dialog.

The following dialog depicts a choice list.

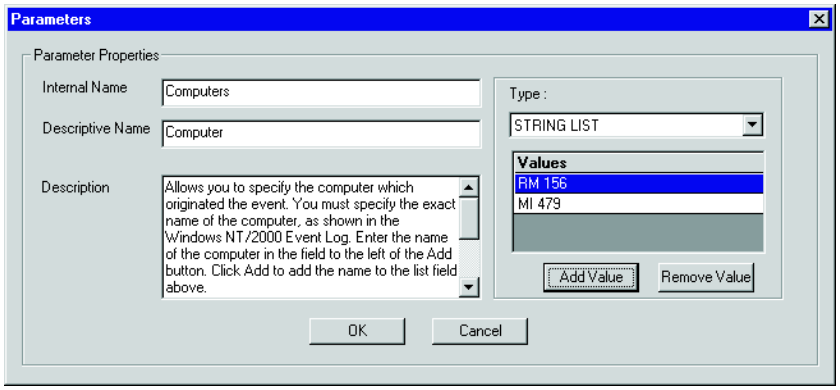


The choice list shows the type of filter applied to the parameters specified with this dialog. You can select **and** or **or** filter. If you select the **and** filter, an event is generated only when *all* of the conditions specified by the other parameters are met. If you select the **or** filter an event is generated when *any* of the conditions specified by the other parameters is met. The default value is **and**, but it can be changed at a later time from the Tivoli Monitoring dialogs. The following figure illustrates the Parameters dialog selecting a numeric list type and showing the numbers of the required Event IDs. The inserted value must be the numeric value of the Event ID indicated in the Windows Event Log. The numeric values of the Event

IDs are displayed in the Values text box.



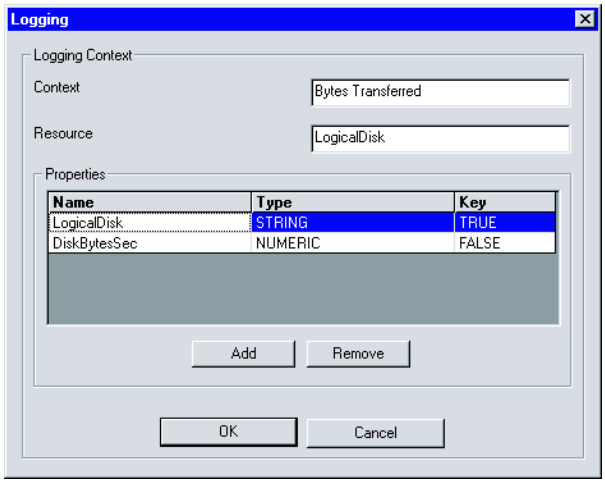
The following dialog depicts a string list.



The string list shows the workstations that originated the event. The **Values** list box shows the names of the workstations, exactly as they are indicated in the Windows EventLog.

Logging Dialog

The Logging dialog is depicted in the following figure.



The context to which the logging activity belongs is the EventLog. Event Log Occurrences are the resources whose activity is logged. Severity, Event ID and the name of the computer where the event occurs are the properties on which data logging is focused.

Decision Tree Script

All settings implemented in the previous dialogs appear automatically in the first parts of the decision tree script, as shown in the following code sample:

```
Public Sub Main()
.......................
End SubPublic Function SetDefaultConfiguration (Svc As Object) As Long

' General info section
'<<GENERAL_INFO>>
Svc.SetModelName "ParamEventLog"
Svc.SetProfileName "281239968"
Svc.SetCycleTime 60
'<<\GENERAL_INFO>>

' Thresholds section
'<<THRESHOLDS_INFO>>
'<<\THRESHOLDS_INFO>>

' Parameters section
'<<PARAMETERS_INFO>>
Svc.DefineNumParameter "Eids",_
"44.00,78.00"
Svc.DefineStrParameter "Computers",_
"RM 156,MI 479"
Svc.DefineStrParameter "LogType", "Application,Security,System"
Svc.DefineStrParameter "Win2kLogs", "Directory Service,_
File Replication Service,DNS Server"
Svc.DefineStrParameter "Severity", "error,information,_
warning,audit failure,audit success"
Svc.DefineStrParameter "Source", ""
Svc.DefineStrParameter "FilterType", "and,or"
'<<\PARAMETERS_INFO>>

' Dynamic model section
'<<DATA_INFO>>
Svc.DefineClassAsync "Win32_NTLogEvent",_
"ROOT\CIMV2:Win32_NTLogEvent","InstanceCreation",_
"RecordNumber,EventIdentifier",_
"TimeWritten,TimeGenerated,_
SourceName,ComputerName,Logfile,Type,Message"
'<<\DATA_INFO>>

' Event definition section
'<<EVENTS_INFO>>
Svc.DefineEvent "TMW_NTEventLogOccurred",_
"EvtID", "Logfile,SourceName,TimeGenerated,_
TimeWritten,ComputerName,Message,severity"
'<<\EVENTS_INFO>>

' Logging definition section
'<<LOGGING_INFO>>
Svc.DefineLogInst "EventLog",_
"Event log Occurrences", "EvtID", "EvtID", "severity,ComputerName"
'<<\LOGGING_INFO>>

' Place your additional intializing code below


SetDefaultConfiguration = 0

End Function
```

Visit Tree Function

The following code sample contains the visit tree function that is called once every cycle time. This function implements the monitoring algorithm.

```
Public Function VisitTree(Svc As Object) As Long
'Variable declaration
.....................
         Eids_Flag = 0
        Severity_Flag = 0
        LogType_Flag = 0
        Source_Flag = 0
        Computers_Flag = 0
        Win2kLogs_Flag = 0
        BoolOper_Flag = 0
        Total_Flag = 0



        numTotal = Svc.GETNUMOFINST("Win32_NTLogEvent")
        i = 0
        Do While i < numTotal
            'get all the property of the instance

            numEvtID = Svc.GetNumProperty_
("Win32_NTLogEvent", i, "EventIdentifier")
            strLogfile = Svc.GetStrProperty("Win32_NTLogEvent",_
i, "Logfile")
            strSourceName = Svc.GetStrProperty("Win32_NTLogEvent",_
i, "SourceName")
            strTimeGenerated = Svc.GetStrProperty("Win32_NTLogEvent",_
i, "TimeGenerated")
            strTimeWritten = Svc.GetStrProperty("Win32_NTLogEvent",_
i, "TimeWritten")
            strComputerName = Svc.GetStrProperty("Win32_NTLogEvent",_
i, "ComputerName")
            strMessage = Svc.GetStrProperty("Win32_NTLogEvent",_
i, "Message")
            numRecord = Svc.GetNumProperty("Win32_NTLogEvent",_
i, "RecordNumber")
            strType = Svc.GetStrProperty_
("Win32_NTLogEvent", i, "Type")

            Svc.LogInst "EventLog",_
 "Event log Occurrences", numEvtID, strType, strComputerName

            i = i + 1

            'verify the parameters;

            numSeverity = Svc.GetStrParameterCount("Severity")

            If numSeverity = 0 Then
                Total_Flag = Total_Flag + 1
            Else
                j = 0
                Do While j < numSeverity
                    SevElem = Svc.GetStrParameter_
("Severity", j)

                    If strType = SevElem Then
                        Severity_Flag = 1
                        Total_Flag = Total_Flag + 1
```

```
                        j = numSeverity
                    End If
                    j = j + 1
                Loop
            End If

            numEvs = Svc.GetNumParameterCount("Eids")
            If numEvs = 0 Then
                Total_Flag = Total_Flag + 1
            Else
                j = 0
                Do While j < numEvs
                    EvtID = Svc.GetNumParameter_
("Eids", j)

                    If numEvtID = EvtID Then
                        Eids_Flag = 1
                        Total_Flag = Total_Flag + 1
                        j = numEvs
                    End If
                    j = j + 1
                Loop
            End If


            numLogType = Svc.GetStrParameterCount_
("LogType")
            If numLogType = 0 Then
                Total_Flag = Total_Flag + 1
            Else
                j = 0
                Do While j < numLogType
                    LogTypeElem = Svc.GetStrParameter("LogType", j)
                    If strLogfile = LogTypeElem Then
                        LogType_Flag = 1
                        Total_Flag = Total_Flag + 1
                        j = numLogType
                    End If
                    j = j + 1
                Loop
            End If

            numSource = Svc.GetStrParameterCount_
("Source")
            If numSource = 0 Then
                Total_Flag = Total_Flag + 1
            Else
                j = 0
                Do While j < numSource
                    SourceElem = Svc.GetStrParameter_
("Source", j)
                    If strSourceName = SourceElem_
Then
                        Source_Flag = 1
                        Total_Flag = Total_Flag + 1
                        j = numSource
                    End If
                    j = j + 1
                Loop
            End If

            numComputers = Svc.GetStrParameterCount("Computers")
            If numComputers = 0 Then
                Total_Flag = Total_Flag + 1
            Else
```

```
                j = 0
                Do While j < numComputers
                    CompElem = Svc.GetStrParameter_
("Computers", j)
                    If strComputerName = CompElem_
Then
                        Computers_Flag = 1
                        Total_Flag = Total_Flag + 1
                        j = numComputers
                    End If
                    j = j + 1
                Loop
            End If

            numWin2kLogs = Svc.GetStrParameterCount("Win2kLogs")
            If numWin2kLogs = 0 Then
                Total_Flag = Total_Flag + 1
            Else
                j = 0
                Do While j < numWin2kLogs
                    Win2kLogsElem = Svc.GetStrParameter("Win2kLogs", j)
                    If strLogfile = Win2kLogsElem_
Then
                        Win2kLogs_Flag = 1
                        Total_Flag = Total_Flag + 1
                        j = numWin2kLogs
                    End If
                    j = j + 1
                Loop
            End If

            'Verify if and or or
            BoolOperElem = Svc.GetStrParameter_
("FilterType", 0)
            If BoolOperElem = "and" Then
                BoolOper_Flag = 1
                Total_Flag = Total_Flag + 1
            End If


            'Process the flags

            Select Case strType
            Case Is = "warning"
                strTecSeverity = "WARNING"
            Case Is = "information"
                strTecSeverity = "HARMLESS"
            Case Is = "error"
                strTecSeverity = "MINOR"
            Case Is = "audit success"
                strTecSeverity = "HARMLESS"
            Case Is = "audit failure"
                strTecSeverity = "MINOR"
            Case Else
                strTecSeverity = "HARMLESS"
            End Select



            If Total_Flag = 7 Then

                Svc.SENDEVENT _
"TMW_NTEventLogOccurred", _
numEvtID, _
                strLogfile, _
```

```
                strSourceName, _
                strTimeGenerated, _
                strTimeWritten, _
                strComputerName, _
                strMessage, _
strTecSeverity

            Else
                If BoolOper_Flag = 0 Then
                    If (Eids_Flag
                + Severity_Flag
                + LogType_Flag + Source_Flag
+ Computers_Flag + Win2kLogs_Flag)_
             > 0 Then


  Svc.SENDEVENT "TMW_NTEventLogOccurred", _
                        numEvtID, _
                strLogfile, _
                strSourceName, _
                strTimeGenerated, _
                strTimeWritten, _
                strComputerName, _
                strMessage, _
strTecSeverity

                    End If
                End If
            End If

            'Reset the flags
            Eids_Flag = 0
            Severity_Flag = 0
            LogType_Flag = 0
            Source_Flag = 0
            Computers_Flag = 0
            Win2kLogs_Flag = 0
            BoolOper_Flag = 0
            Total_Flag = 0

        'End  while
        Loop

      'End DECISIONTREE

VisitTree = 0

End Function
```

C

Resource Models for Microsoft Exchange Server

The workbench comes with a set of sample, default resource models. These samples can be used to produce other resource models that can be installed on Tivoli Monitoring. All samples provided correspond to resource models installed with Tivoli Monitoring. For more information on using the samples, see *Tivoli Monitoring User's Guide*.

In this chapter you will find a description of the Microsoft Exchange Server resource models and some instructions to use them with Tivoli Monitoring.

Microsoft Exchange Server Resource Models

The following models are specifically tailored to detect problems with Microsoft Exchange Server resources and services:

- Microsoft Exchange Server services
- Microsoft Exchange Server ports availability
- Microsoft Exchange Server performance
- Microsoft Exchange Server diagnostic logging

Microsoft Exchange Server Services

The Microsoft Exchange Services resource model checks that Microsoft Exchange Server 5.5 components are available.

All components of Exchange Server are implemented as Windows services. The basic components supply services for managing the Microsoft Exchange Directory, storing and forwarding messages, and other essential functions.

Services, like any other installed software, can become corrupted or unstable. The resource model checks that these services are not stopped or unstable to ensure that Microsoft Exchange is working correctly. Unstable services must be stopped to ensure they do not harm other functions of Microsoft Exchange.

You can also configure the resource model to monitor any other, optional, components of Microsoft Exchange Server that you have installed.

Prerequisites

If you want to monitor Microsoft Exchange Server 5.5 resources, Tivoli Monitoring (Resource Models for Microsoft Exchange Server 5.5) and Microsoft Exchange server 5.5. must be installed on the Tivoli management region server.

Problems Highlighted

The Microsoft Exchange Services resource model highlights the following problems:

- Key services

  All components of Microsoft Exchange Server are implemented as Windows services.

  The basic components supply services for managing the Microsoft Exchange Directory, storing and forwarding messages, and other essential functions.

- Unstable services

  Services, like any other installed software, can become corrupted or unstable. The resource model checks that Microsoft Exchange services are stable. Unstable services must be stopped to ensure they do not harm other functions of Microsoft Exchange.

Parameters

The Microsoft Exchange Services resource model has the following parameters:

- Microsoft Exchange core components
- Optional Microsoft Exchange components

Microsoft Exchange Core Components

The following services are key to Microsoft Exchange Server:

- Microsoft Exchange System Attendant
- Microsoft Exchange Directory Service
- Microsoft Exchange Information Store
- Microsoft Exchange Message Transfer Agent

Optional Microsoft Exchange Components

The following services are optional to Microsoft Exchange Server:

- Microsoft Exchange Connector for Lotus® cc:Mail
- Microsoft Exchange Directory Synchronization
- Microsoft Exchange Event Service
- Microsoft Schedule Free+Busy Connector
- Microsoft Mail Connector Interchange
- Microsoft Exchange Key Management Server

Events

The Microsoft Exchange Services resource model generates the following events:

- Service failing
- Service stopped

Service Failing

The associated string is TMW_ExcServicesFailingService. This event is generated when any Microsoft Exchange Server service is not in a stable state. Services that are not stable can cause problems for the local machine and for connected machines over the network.

Services that are not stable should be stopped to prevent them from causing any bottlenecks or damage.

TMW_ExcServicesFailingService has the following event properties:

ServiceStatus
The current status of the service.

Name
The name of the service being examined.

StartMode
The start mode of the service.

State The current state of the service.

Service Stopped

The associated string is TMW_ExcServicesStoppedService. This event is generated when any Microsoft Exchange Server service is stopped. If a key service is stopped, it must be restarted to ensure Microsoft Exchange is working properly.

TMW_ExcServicesStoppedService has the following event properties:

ServiceStatus
The current status of the service.

Name
The name of the service being examined.

StartMode
The start mode of the service.

Built-in Actions

The Microsoft Exchange Services resource model generates the following built-in action:

Restart Service
If a key service is in a stopped or paused state, this action restarts the service. The system administrator is also notified.

Microsoft Exchange Ports Availability

The Microsoft Exchange Ports Availability resource model checks the availability of all ports used in the Microsoft Exchange Server organization.

Prerequisites

The Microsoft Exchange Services resource model requires Microsoft Exchange Server 5.5.

Problems highlighted

The Microsoft Exchange Ports Availability resource model highlights when one or more Microsoft Exchange ports are not available.

Parameters

The Microsoft Exchange Ports Availability resource model has the following parameters, which you can specify to monitor the ports on which Microsoft Exchange Server listens for specific types of communications:

Port Number	Microsoft Exchange Server Listens for
Port 110	Incoming connection requests from POP3 clients for message download
Port 143	Incoming connection requests from IMAP4 clients, for message download and retrieval

Port Number	Microsoft Exchange Server Listens for
Port 25	Incoming SMTP messages to Internet Mail Connector and Internet Mail Service
Port 80	Incoming connections from Microsoft Outlook Web Access Server
Port 389	Incoming connections from LDAP clients

Events

The Microsoft Exchange Ports Availability resource model generates the following event:

- A Microsoft Exchange port is not available

A Microsoft Exchange Port is Not Available

The associated string is TMW_ExcPortProblem.

The event is generated when one or more of the Microsoft Exchange ports that are being monitored by the resource model is not available.

Built-in actions

The Microsoft Exchange Ports Availability resource model contains no built-in actions.

Microsoft Exchange Server Performance

The Microsoft Exchange Server Performance resource model monitors a set of Microsoft Exchange Server performance counters.

Prerequisites

The Microsoft Exchange Services resource model requires Microsoft Exchange Server 5.5.

Problems Highlighted

The Microsoft Exchange Services resource model highlights performance problems of the following components of Microsoft Exchange Server:

- Microsoft Exchange Internet Mail Connector
- Microsoft Exchange Directory Service
- Microsoft Exchange Information Store
- Microsoft Exchange Message Transfer Agent

Events

The Microsoft Exchange Services resource model generates the following events:

- Microsoft Exchange Message Transfer Agent queue length is high
- Non-zero queue size for Information Store
- Failing Directory Synchronization
- Slow Message Transfer Agent message delivery
- Slow Information System Delivery
- Long Internet Mail Connector outbound queue
- Slow Information System local delivery
- High queue length
- Long Internet Mail Connector inbound queue

Microsoft Exchange Message Transfer Agent Queue Length Is High

The associated string is TMW_HighPersistentWorkQueueMessage.

This event is generated when the queue length for the Message Transfer Agent, either for inbound and outbound messages, is persistently high.

Non zero queue size for Information Store

The associated string is TMW_PossibleMTADown.

The size of the queue of messages outbound from the Information Store to the Message Transfer Agent is zero under normal conditions. This event is generated if the following conditions apply for a significant duration:

- The size of the queue stays at a non-zero value
- The Message Transfer Agent is down or its performance is reduced.

Failing Directory Synchronization

The associated string is TMW_FailedSync.

The counter of directory synchronization requests not answered by other servers in this Microsoft Exchange site, should be zero. This event is generated if this counter remains at a non-zero value. This indicates that a server missed the synchronization. Scan the Event Log to find the name of the server.

Slow Message Transfer Agent message delivery

The associated string is TMW_SlowMTADelivery.

This event is generated to show how long a message can expect to remain in the Message Transfer Agent queue before being delivered.

Slow Information System Delivery

The associated string is TMW_SlowISDelivery.

This event is generated when the message delivery rate is low. A high value could indicate a performance problem with the Message Transfer Agent.

Long Internet Mail Connector Outbound Queue

The associated string is TMW_HighIMCQueueOut.

This event is generated when a large number of messages has been queued from Microsoft Exchange Server for delivery to the Internet.

Slow Information System local delivery

The associated string is TMW_SlowISLocalDelivery.

This event is generated when local IS delivery is slow. A high average value of the time required for the last ten local deliveries inside the information system could indicate a performance problem within the information store.

High queue length

The associated string is TMW_HighQueueLength.

This event is generated when the queue length is high.

Long Internet Mail Connector inbound queue

The associated string is TMW_HighIMCQueueIn.

This event is generated if the number of messages received from the Internet destined for Microsoft Exchange Server is high.

Thresholds

The Microsoft Exchange Services resource model has the following thresholds:

- Internet Mail Connector inbound queue
- Average time for delivery
- Time Message Transfer Agent delivery
- Internet Mail Connector outbound queue
- MTA work queue length
- Message Transfer Agent connections queue length
- Average time for local delivery

Internet Mail Connector Inbound Queue

This is the count of messages received from the Internet destined for Microsoft Exchange Server. The default threshold setting is 20.

Average Time for Delivery

This is the average time taken for the last ten messages to be submitted to the Message Transfer Agent for remote delivery. The default threshold setting is 5.

Time Message Transfer Agent Delivery

This is an estimate of the delay in the Message Transfer Agent queue before messages are delivered or sent. A high value indicates a problem either in performance or in transmitting to other servers. The default threshold setting is 60.

Internet Mail Connector Outbound Queue

This is the count of messages from Microsoft Exchange Server that are queued for delivery to the internet. The default threshold setting is 20.

MTA Work Queue Length

This is the queue length for the whole of the Message Transfer Agent. A high number indicates a problem either in performance or in transmitting to other servers. The default threshold setting is 30.

Message Transfer Agent Connections Queue Length

This value is related to each work queue within the Message Transfer Agent. If a large queue is detected in the Message Transfer Agent, this counter pinpoints the exact connection responsible. The default threshold setting is 20.

Average Time for Local Delivery

This is the average length of time the last ten local delivery messages waited for transport to a mailbox in the same information store. The default threshold setting is 5.

Microsoft Exchange Server Diagnostic Logging

Microsoft Exchange logs events in the Windows event log. You can use the Microsoft Exchange Server Diagnostic Logging resource model to filter events for specific Microsoft Exchange components by event type and forward the filtered events to the Tivoli Enterprise Console.

Prerequisites

The Microsoft Exchange Services resource model requires Microsoft Exchange Server 5.5.

Parameters

The Microsoft Exchange Server Diagnostic Logging resource model has the following parameters:

- Event type
- Event source

Event Type

This parameter allows you to specify the severity of the events to be monitored. You can choose one or more of the following severity levels:

- Information
- Warning
- Error
- Success audit
- Failure audit

These severities are mapped to the following Tivoli Enterprise Console server severities:

- Information -> Harmless
- Warning -> Warning
- Error -> Minor
- Success audit -> Harmless
- Failure audit -> Minor

Event Source

Each Microsoft Exchange Server component generates different kinds of events. This parameter allows you to specify the source of the event you want to monitor.

You can specify one or more of the following sources:

- Microsoft Exchange Directory Service
- Microsoft Exchange Directory Synchronization
- Microsoft Exchange Information Store
- Microsoft Exchange Private Information Store
- Microsoft Exchange Public Information Store
- Microsoft Exchange Internet Mail Service
- Microsoft Exchange KM Server
- Microsoft Exchange Message Transfer Agent
- Microsoft Exchange Connector for Lotus cc:Mail
- Microsoft Exchange Microsoft Mail Connector
- Microsoft Exchange Microsoft Free+Busy Connector

You can set the required level of detail of the events to be logged in the Configuring Diagnostic Logging property page on Microsoft Exchange Server components.

Events

The Microsoft Exchange Server Diagnostic Logging resource model generates the following event:

- Microsoft Exchange logged an event

Microsoft Exchange Logged An Event

This event is generated whenever the Microsoft Exchange Server Diagnostic Logging resource model detects that one of the event types you specified has been generated by one of the event source components you specified.

D

Instrumentation Library Type Interface

This appendix describes the Instrumentation Library Type (ILT) interface and provides indications to use it. It is composed of three sections: the first section documents the public operations of the ILT interface, the second section documents the support classes, the third and last section provides guidelines and samples for creating a UNIX provider.

ILT Public Operations

This section describes the Public Operations available with the ILT interface.

Note: some public operation are not supported by Tivoli Monitoring . When that is the case, the operation is indicated as not supported.

enumerateInstances

Supported: YES

Syntax

```
public java.util.Enumeration enumerateInstances (M12ClassPath classPath,
java.lang.String mappingString,
ParameterSet parms)
throws M12Exception
```

Parameters

classPath
The M12ClassPath identifying the class whose instances have to be enumerated.

mappingString
Any string that has been specified in the M12_Instrumentation qualifier for the ENUM operation type for this class.

parms A ParameterSet object filled by the client with parameters for ILT.

Description
Returns all M12ObjectIdentity objects that identify all the instances belonging to the class specified in the classPath.

Returns
Enumeration of instances identity (M12ObjectIdentity).

Exceptions Thrown
M12Exception

getProperty

Supported: YES

Syntax

```
public java.lang.String getProperty (M12ObjectIdentity targetInstance,
java.lang.String propertyName,
java.lang.String mappingString,
ParameterSet parms)
throws M12Exception
```

Parameters

targetInstance
M12ObjectIdentity that identifies the instance of the resource to be accessed.

propertyName
The property whose value is required.

mappingString
Any string that has been specified in the M12_Instrumentation qualifier for the GET operation type for this property.

parms A ParameterSet object filled by the client with parameters associated to this property.

Description
Gets the value (in String format) of the specified property for the identified object.

Returns
String - the value for property propertyName. Property values have to be CIM standard types and ILT converts them to string format according to the CIM standards.

Exceptions Thrown
M12Exception

getMultipleProperties

Supported: YES

Syntax

```
public M12PropertySet getMultipleProperties (M12ObjectIdentity targetInstance,
java.util.Vector propertyList,
java.lang.String mappingString,
ParameterSet parms)
throws M12Exception
```

Parameters

targetInstance
M12ObjectIdentity that identifies the instance of the resource to be accessed.

propertyList
The list of properties whose value is required.

mappingString
Any string that has been specified in the M12_Instrumentation qualifier for the GET operation type for the class which the specified instance belongs to.

parms A ParameterSet object filled by the client with parameters associated to the class which the specified instance belongs to.

Description
Gets the value (in String format) of the specified properties for the identified object.

Returns
M12PropertySet - the values of the requested properties.

Exceptions Thrown
M12Exception

setProperty

Supported: NO

Syntax

```
public java.lang.String setProperty (M12ObjectIdentity targetInstance,
java.lang.String propertyName,
java.lang.String propertyValue,
java.lang.String mappingString,
ParameterSet parms)
throws M12Exception
```

Parameters

targetInstance
M12ObjectIdentity that identifies the instance of the resource to be accessed.

propertyName
The property whose value is to be set.

propertyValue
The property value to be set.

mappingString
Any string that has been specified in the M12_Instrumentation qualifier for the SET operation type for this property.

parms A ParameterSet object filled by the client with parameters associated to this property.

Description
Sets the value (in String format) of the specified property for the identified object.

Returns
String - the new value of the specified property.

Exceptions Thrown
M12Exception

invokeMethod

Supported: YES

Syntax

```
public java.lang.String invokeMethod (M12ClassPath classPath,
java.lang.String methodName,
java.lang.String mappingString,
ParameterSet parms,
ParameterSet inParms,
ParameterSet outParms,
throws M12Exception
```

Parameters

classPath
The M12ClassPath that identifies the class whose method has to be called.

methodName
The name of the method to be called.

mappingString
Any string that has been specified in the M12_Instrumentation qualifier for the INVOKE operation type for this method.

parms A ParameterSet object filled by the client with parameters for this method.

inParms
A ParameterSet object filled by the client with parameters to be passed to the method.

outParms
A ParameterSet object created by the client and filled by the method with output results.

Description
Invokes the specified method on the CIM instance belonging to the class specified in the classPath.

Returns
String - the result of the method. Result values have to be CIM standard types and ILT converts them to string format according to the CIM standards.

Exceptions Thrown
M12Exception

invokeMethod

Supported: YES

Syntax

```
public java.lang.String invokeMethod (M12ObjectIdentity targetInstance,
java.lang.String methodName,
java.lang.String mappingString,
ParameterSet parms,
ParameterSet inParms,
```

```
ParameterSet outParms,
throws M12Exception
```

Parameters

targetInstance
M12ObjectIdentity that identifies the instance whose method has to be called.

methodName
The name of the method to be called.

mappingString
Any string that has been specified in the M12_Instrumentation qualifier for the INVOKE operation type for this method.

parms A ParameterSet object filled by the client with parameters for this method.

inParms
A ParameterSet object filled by the client with parameters to be passed to the method.

outParms
A ParameterSet object created by the client and filled by the method with output results.

Description
Invokes the specified method on the identified CIM instance.

Returns
String - the result of the method. Result values have to be CIM standard types and ILT converts them to string format according to the CIM standards.

Exceptions Thrown
M12Exception

create

Supported: NO

Syntax

```
public void create (M12ObjectIdentity targetInstance,
java.lang.String mappingString,
ParameterSet parms)
throws M12Exception
```

Parameters

targetInstance
M12ObjectIdentity that identifies the instance of the resource to be created.

mappingString
Any string that has been specified in the M12_Instrumentation qualifier for the CREATE operation type for the class whose instance to be created will belong to.

parms A ParameterSet object filled by the client with parameters for the ILT.

Description

Creates an instance of the resource that will be identified by the specified targetInstance.

Returns

void

Exceptions Thrown

M12Exception

destroy

Supported: NO

Syntax

```
public void destroy (M12ObjectIdentity targetInstance,
java.lang.String mappingString,
ParameterSet parms)
throws M12Exception
```

Parameters

targetInstance
M12ObjectIdentity that identifies the instance of the resource to be deleted.

mappingString
Any string that has been specified in the M12_Instrumentation qualifier for the DESTROY operation type for the class whose instance to be deleted belongs to.

parms A ParameterSet object filled by the client with parameters for the ILT.

Description

Deletes an instance of the resource identified by the specified targetInstance.

Returns

void

Exceptions Thrown

M12Exception

ILT Support Classes

This section described the set of classes supported by the ILT interface.

M12ClassPath

Syntax Detail:

```
public M12ClassPath (java.lang.String className)
```

```
public M12ClassPath (java.lang.String className, java.lang.String nameSpace)
```

Method Detail:

```
public java.lang.String getClassName ()

public java.lang.String getNameSpace ()
```

M12IdentityElement

Syntax Detail:

```
public M12IdentityElement (java.lang.String className, M12PropertySet identity)

public M12IdentityElement (java.lang.String className, java.lang.String nameSpace, M12PropertySet
```

Method Detail:

```
public java.lang.String getClassName ()

public java.lang.String getNameSpace ()

public M12PropertySet getIdentity ()
```

M12ObjectIdentity

Syntax Detail:

```
public M12ObjectIdentity (M12IdentityElement[ ] scopingPath)
```

Method Detail:

```
public M12IdentityElement[ ] getScopingPath ()
```

M12PropertySet

Syntax Detail:

```
public M12PropertySet ()

public M12PropertySet (java.util.Properties prop)

public M12PropertySet (M12PropertySet propertySet)
```

Method Detail:

```
public void setProperty (java.lang.String propertyName, java.lang.String propertyValue)

public java.lang.String getProperty (java.lang.String propertyName)

public java.util.Enumeration propertyNames ()
```

```
public int size ()
```

M12Exception

Syntax Detail:

```
public M12Exception (java.lang.Exception e)

public M12Exception (java.lang.Exception e, java.lang.String message)
```

Method Detail:

```
public java.lang.Exception getTargetException ()
```

ParameterSet

Syntax Detail:

```
public ParameterSet ()

public ParameterSet (ParameterSet paramSet)
```

Method Detail:

```
public void setParam (java.lang.String paramName, java.lang.Object paramValue)

public java.lang.Object getParam (java.lang.String paramName)

public void removeParam (java.lang.String paramName)

public java.util.Enumeration parametersNames ()
```

ParameterSetList

Syntax Detail:

```
public ParameterSetList ()
```

Method Detail:

```
public void addParameterSet (java.lang.String parameterSetName, ParameterSet parameterSet)

public ParameterSet getParameterSet (java.lang.String paramName)

public void removeParameterSet (java.lang.String parameterSetName)

public java.util.Enumeration parameterSetNames ()
```

Writing a provider for UNIX

On UNIX platforms the provider environment is Java. The provider code is written as Java classes that implement the ILT interface described in this chapter.

There are two technologies used to create data providers: the first technology is writing the code that interacts with the managed resource, the second is writing the CIM classes that model the managed resource. The CIM classes definitions specify what code to invoke to retrieve the desired data.

The following sections show simplified elements for creating a provider.

Creating a MOF file for UNIX

The first step consists in writing a Mof file for UNIX providers (in other words a textual definiton of a CIM class) and in loading the Mof file to WMI so that the Workbench can be used to create resource models for UNIX. Note that the Mof file must have DOS style CR/LF newlines, otherwise the classes wil not show up in WMI.

The usual namespace in WMI is `root\cimv2`, but for UNIX you load the Mof files into `root\default`:

```
mofcomp -N:root\default Sample01.mof
```

M12_Instrumentation is a non-standard qualifier which applies to Class, Property and Method. Its format is:

[M12_Instrumentation(″Java.<path to ILT>|<mappingstring>|<operation>″].

This qualifier is used to identify which ILT (″path to ILT″) is able to perform the specified operation.

<path to ILT> is the Java class that implements the ILT and that must be specified with the complete package (with no .class extension).

<mappingstring> is a string whose meaning is known to the ILT.

The allowed operations are:

- Enum (only for the qualifier associated with the class)
- Get (for the qualifier associated with class and property)
- Invoke (for the qualifier associated with class and method)

It is required that all classes have an M12_Instrumentation qualifier for the Enum operation. It is also required that an M12_Instrumentation qualifier for the Get operation be present for every non-key property. As an alternative, when the instrumentation string is the same for every property, a single M12_Instrumentation qualifier for the Get operation can be set at class level. It is also required that an M12_Instrumentation qualifier for the Invoke operation be present for every method. As an alternative, when the instrumentation string is the same for every method, a single M12_Instrumentation qualifier for the Invoke operation can be set at class level.

The following is an example of Mof file for UNIX:

```
//************************************************************************
[
 Description ("Unix File Systems info"),
 provider("com.tivoli.dmunix.ep.touchpoint.cimom.ifc.M12JavaProvider"),
 M12_Instrumentation ("Java.com.tivoli.dmunix.ep.ilts.DMXFileSystemIlt | | ENUM"),
 M12_Instrumentation ("Java.com.tivoli.dmunix.ep.ilts.DMXFileSystemIlt | | GET")
]
class DMXFileSystem
```

```
{
[key]string mountPoint;
sint32 usedKBytes;
sint32 availKBytes;

[ provider("com.tivoli.dmunix.ep.touchpoint.cimom.ifc.M12JavaProvider"),
  M12_Instrumentation ("Java.com.tivoli.dmunix.ep.ilts.DMXFileSystemIlt || GET")]
sint32 totalKBytes;

};
```

Properties not associated with any ″provider″ or ″M12_Instrumentation″ qualifier are collected by calling the getMultpleProperties method, while all other properties are collected through a getProperty method.

ILT Sample

The following is a sample of ILT:

```
package com.tivoli.dmunix.ep.ilts;

import java.util.*;
import com.tivoli.javautils.Trace;
import com.tivoli.dmunix.ep.touchpoint.base.*;
import com.tivoli.dmunix.ep.providers.DMXFileSystem;

public class DMXFileSystemIlt implements ILTInterface {

public String getProperty (M12ObjectIdentity targetInstance, String propertyName,
   String mappingString, ParameterSet parms)
throws M12Exception {
try {
M12IdentityElement idElem = (targetInstance.getScopingPath())[0];
M12PropertySet propSet = idElem.getIdentity();
String mountPoint = propSet.getProperty("mountPoint");
if (propertyName.equals("totalKBytes")) {
Integer res = new Integer(DMXFileSystem.getTotalKBytes(mountPoint));
return res.toString();
}
} catch (Exception e) {
...................
}
}

public M12PropertySet getMultipleProperties (M12ObjectIdentity targetInstance, Vector propertyList,  String ma
throws M12Exception {
try {
M12IdentityElement idElem = (targetInstance.getScopingPath())[0];
M12PropertySet propSet = idElem.getIdentity();
String mountPoint = propSet.getProperty("mountPoint");
M12PropertySet result = new M12PropertySet();
for (int i = 0; i < propertyList.size(); i++) {
String propertyName = (String)propertyList.elementAt(i);
if (propertyName.equals("usedKBytes")) {
Integer res = new Integer(DMXFileSystem.getUsedKBytes(mountPoint));
result.setProperty(propertyName,res);
}
if (propertyName.equals("availKBytes")) {
Integer res = new Integer(DMXFileSystem.getAvailKBytes(mountPoint));result.setProperty(propertyName,res);
}
}
return result;
} catch (Exception e) {
...................
}
}

public Enumeration enumerateInstances (M12ClassPath classPath, String mappingString,
```

```
   ParameterSet parms)
throws M12Exception {
try {
Vector result = new Vector();
String[] mountPoints = DMXFileSystem.getMountPoints();
if (mountPoints == null) {
trace.log(1,"DMXFileSystemIlt","enumerateInstances: no instances found");
return null;
}
for (int i=0; i<mountPoints.length; i++) {
M12IdentityElement idElem;
M12PropertySet propSet = new M12PropertySet();
propSet.setProperty("mountPoint",mountPoints[i]);
idElem = new M12IdentityElement(classPath.getClassName(),
classPath.getNameSpace(),
propSet);
result.add(new M12ObjectIdentity(new M12IdentityElement[] {idElem}));
}
return result.elements();
} catch (Exception e) {
.............
}
}
.........................

}
```

If the ILT interfaces with native libraries (through JNI, the Java Native Interface), these libraries must be added to the resource model as a dependency.

ILTs must be packaged into `.jar` files and added to the resource model as dependencies.

E

Error Messages

This appendix lists the messages that can be displayed when creating or debugging a resource model with the workbench.

The messages are listed in ascending numeric order.

Identifying a Message

Messages are of different types but are all identified in the same way. The following example shows a typical message and explains its identifying components.

Identity	Message
AMW0500E	The Action Browser is unable to get the class definition.

AMW This prefix identifies the message as belonging to the workbench.

0XXX The unique serial number of the message.

E Is the type of message and can be:

- **I** **Information messages** provide feedback about something that has happened in the product or system that may be important. These messages also give guidance when you are requesting a specific action from the product.
- **W** **Warning messages** call your attention to an exception condition that is not necessarily an error but may cause problems if not attended to.
- **E** **Error messages** indicate that an action cannot be completed because of a user or system error. These error messages always require user response.

Notation

Some messages, especially information and warning messages, are multi-purpose. The same basic text can contain different strings such as different command names or application names, according to the way the application was behaving when the message was generated. These messages are shown in the following sections with the string identity displayed in italics at the appropriate part of the message.

Messages

The following messages can be displayed.

AMW0500E The Action Browser is unable to get the class definition

Explanation: WMI Service is not working properly.

User Response: Close the workbench, stop and restart WMI Service, then try again. If the problem persists contact the customer support.

AMW0510W There are some invalid fields. List of one or more of the empty fields.

Explanation: Some fields are empty or incorrect.

User Response: Enter the appropriate information in the required fields.

AMW0511E Errors occurred during the creation of the Event Aggregator profile *<profile>*

Explanation: WMI Service is corrupted or some of its components are not properly installed.

User Response: Close the workbench, stop and restart WMI Service, then try again. If the problem persists reinstall the WMI Service, or contact customer support.

AMW0513E An error occurred during data collection

Explanation: Error *<error_number>* occurred during data collection. WMI, or a provider may not be working properly, or you may not have the authorization required to perform data collection.

User Response: Ensure that WMI and the provider are working properly, if they are not, stop and restart them. If you are collecting data remotely, ensure that you have the required authorization.

AMW0514W No instance was found

Explanation: No instance was found for the class *<classname>*.

User Response: If you expect some instances for this class, ensure that the provider is available and the MOF file is correctly compiled.

AMW0515E Unable to find WMI

Explanation: The Windows Management Instrumentation Service is not available on this workstation. The workbench can not run without that service

User Response: Install WMI Service.

AMW0516E Unable to create an instance of TMWService.IAnalyzer interface

Explanation: An internal error occurred, a new workspace cannot be created.

User Response: Uninstall and reinstall the workbench; if the problem persists contact customer support.

AMW0517E An error occurred during general settings modification

Explanation: Error *<error_number>* occurred during general settings modification. Refer to the workbench documentation for more details about the TMWService error codes.

User Response: Make sure that the entered information complies with naming conventions.

AMW0519E An error occurred while opening a saved workspace

Explanation: Error <error_number> occurred. A saved workspace could not be opened. Workspace file may be corrupted, or you are trying to open a workspace created with a higher version of the workbench.

User Response: If the workspace is not corrupted, install the workbench new version.

AMW0520E An error occurred. Workspace could not be saved

Explanation: Error <error_number> occurred. The workspace could not be saved. Unable to write on the target file due to disk full or missing authorization.

User Response: Check that the storage media is accessible.

AMW0521E An error occurred during VBA code running

Explanation: Error occurred: *<Line_number _Column_number>*.

User Response: Fix the VBA error on the given line and column.

AMW0522E An error occurred during Action Manager creation

Explanation: Action Manager profile <profile_name> could not be created. WMI Service is corrupted, or some of its components are not properly installed.

User Response: Close the workbench, stop and restart WMI Service, then try again. If the problem persists, reinstall WMI, or contact customer support.

AMW0523E A required tag has been removed. Unable to update the settings in the code

Explanation: During the editing of the resource model code, the tag <tag_name> has been removed. It is not possible to update the general info.

User Response: Insert the removed tag in the SetDefaultConfiguration function.

AMW0532E Unable to create an instance of TMWService.IAnalyzer interface while scanning for new items

Explanation: An internal error occurred.

User Response: Uninstall and reinstall the workbench, if the problem persists contact customer support.

AMW0533E An error occurred during the scanning for new items

Explanation: The function SetDefaultConfiguration is missing or contains errors.

User Response: Make sure that the function is available, or fix VBA errors.

AMW0537E The attribute name is not specified

Explanation: The attribute name field is empty.

User Response: Specify an attribute name.

AMW0542E The property named <*property_name*> has already been used

Explanation: The name <*property_name*> cannot be used for this property because has already been used.

User Response: Specify a different name.

AMW0545E Parameter string value not specified

Explanation: The parameter field must contain a string value.

User Response: Enter a valid string value.

AMW0546E An error occurred during general setting modification

Explanation: Error <*error_number*> occurred during general setting modification, refer to the workbench documentation for more details about the TMWService error codes.

User Response: Make sure that the entered information complies with naming conventions.

AMW0547E An error occurred. Class could not be added

Explanation: Error <*error_number*> occurred when you tried to add the class, refer to the workbench documentation for more details about the TMWService error codes.

User Response: Ensure that the selected class had not been already added.

AMW0548E An error occurred. Event could not be added

Explanation: Error <*error_number*> occurred when you tried to add the event, refer to the workbench documentation for more details about the TMWService error codes.

User Response: Ensure that there is not any existing event with the same name and that the entered information complies with naming conventions.

AMW0549E An error occurred during the event modification

Explanation: Error <*error_number*> occurred when you tried to modify the event definition, refer to the workbench documentation for more details about the TMWService error codes.

User Response: Ensure that there is not any existing event with the same name and that the entered information complies with naming conventions.

AMW0551E An error occurred. Threshold could not be added

Explanation: Error <*error_number*> occurred when you tried to add the threshold definition refer to the workbench documentation for more details about the TMWService error codes.

User Response: Ensure that there is not any existing threshold with the same name and that the entered information complies with naming conventions.

AMW0553E An error occurred during the threshold modification

Explanation: Error <*error_number*> occurred when you tried to modify the threshold definition refer to the workbench documentation for more details about the TMWService error codes.

User Response: Ensure that there is not any existing threshold with the same name and that the entered information complies with naming conventions.

AMW0554E An error occurred. Class definition could not be modified

Explanation: Error <*error_number*> occurred when you tried to modify the class definition, refer to the workbench documentation for more details about the TMWService error codes.

User Response: Ensure that there is not any existing class with the same name and that the entered information complies with naming conventions.

AMW0556E An error occurred. Parameter could not be added

Explanation: Error <*error_number*> occurred when you tried to add the parameter, refer to the workbench documentation for more details about the TMWService error codes.

User Response: Ensure that there is not any existing parameter with the same name and that the entered information complies with naming conventions.

AMW0557E An error occurred. Parameter definition could not be modified

Explanation: Error <*error_number*> occurred when you tried to modify the parameter definition, refer to the workbench documentation for more details about the TMWService error codes.

User Response: Ensure that there is not any existing parameter with the same name and that the entered information complies with naming conventions.

AMW0559E An error occurred. Dependency file could not be added

Explanation: Error <*error_number*> occurred when you tried to add the dependency file, refer to the workbench documentation for more details about the TMWService error codes.

User Response: Ensure that there is not any existing dependency file with the same name and that the file is accessible.

AMW0560E An error occurred. Dependency file could not be modified

Explanation: Error <*error_number*> occurred when you tried to modify the dependency file, refer to the workbench documentation for more details about the TMWService error codes.

User Response: Ensure that there is not any existing dependency file with the same name and that the file is accessible.

AMW0561E An error occurred. Class definition could not be removed

Explanation: Error <*error_number*> occurred when you tried to remove the class definition, refer to the workbench documentation for more details about the TMWService error codes.

User Response: None.

AMW0562E An error occurred. Event definition could not be removed

Explanation: Error <*error_number*> occurred when you tried to to remove the event definition, refer to the workbench documentation for more details about the TMWService error codes.

User Response: Ensure that there is not any existing event with the same name and that the entered information complies with naming convention.

AMW0563E An error occurred. Action could not be removed

Explanation: Error <*error_number*> occurred when you tried to remove the action, refer to the workbench documentation for more details about the TMWService error codes.

User Response: None.

AMW0564E An error occurred. Threshold definition could not be removed

Explanation: Error <*error_number*> occurred when you tried to remove the threshold definition, refer to the workbench documentation for more details about the TMWService error codes.

User Response: None.

AMW0565E An error occurred. Parameter definition could not be removed

Explanation: Error <*error_number*> occurred when you tried to remove the parameter definition, refer to the workbench documentation for more details about the TMWService error codes.

User Response: None.

AMW0566E Error <*error_number*> occurred. Dependency file could not be removed

Explanation: Error <*error_number*> occurred when you tried to remove the dependency file, refer to the workbench documentation for more details about the TMWService error codes.

User Response: None.

AMW0567E An error occurred. Logging context could not be added

Explanation: Error <*error_number*> occurred when you tried to add the logging context, refer to the workbench documentation for more details about the TMWService error codes.

User Response: Ensure that there is not any context and resource with the same names and that the entered information complies with naming conventions.

AMW0569E An error occurred. Logging context could not be modified

Explanation: Error <*error_number*> occurred when you tried to modify the logging context, refer to the workbench documentation for more details about the TMWService error codes.

User Response: Ensure that there is not any existing context and resource with the same names and that the entered information complies with naming conventions.

AMW0570E An error occurred. Logging context could not be removed

Explanation: Error <*error_number*> occurred when you tried to remove the logging context, refer to the workbench documentation for more details about the TMWService error codes.

User Response: None.

AMW0571E No Resource Model Available

Explanation: You are trying to export files from an empty workspace.

User Response: Configure your workspace.

AMW0594E Unable to remove the event attribute

Explanation: The event attribute cannot be removed because it is associated to the parameter <*paramener_name*>, in the action <*action_name*>.

User Response: None.

AMW0597E Export Package failed

Explanation: The dependencies could not be copied to directory <*directory_name*>.

User Response: Ensure that there is enough disk space and that you have write access.

AMW0598E Export Package failed

Explanation: File <*filename*> cannot be opened. Unable to write to the target file because there is not enough disk space or you have not the required authorization.

User Response: Ensure that the storage media is accessible.

AMW0599E Unable to extract the dependency file <*filename*>

Explanation: The specified file could not be extracted.

User Response: Ensure that there is enough disk space and that you have write access.

AMW0633E Unable to get the class definition

Explanation: WMI Service is not working properly.

User Response: Close the workbench, stop and restart WMI Service, then try again. If the problem persists contact customer support.

AMW0634E Unable to add property: type not supported

Explanation: You are trying to add a parameter of a type that is not supported.

User Response: Specify a valid parameter type.

AMW0635E Unable to collect instances of the class <*class_name*>

Explanation: Error <*error_number*> occurred during data collection.

User Response: Close the workbench, stop and restart WMI Service, then try again. If the problem persists contact customer support.

AMW0641E Macro still active

Explanation: There are some active macros. You cannot exit Windows without closing all the active macros.

User Response: Close active macros before quitting Windows.

AMW0643I It is not possible to remove the class property.

Explanation: The specified property could not be removed, because it is a key for the selected class.

User Response: Do not remove that property.

AMW0644E The monitoring collection cannot be imported.

Explanation: The monitoring collection cannot be imported from the file *FileName* due to the following error:

User Response: Make sure that the preprocessor and the preprocessing options are correctly set. Ensure that the *Path* environment variable contains the folder where the preprocessor is located. Check also if the .csl file contains errors.

AMW0645E Asynchronous monitors cannot be imported.

Explanation: You have tried to import an asynchronous monitor source. The workbench does not asynchronous monitors to be imported.

User Response: Import a synchronous monitor.

AMW0646W The monitor *MonitorName* is not available for the platform *Platform*.

Explanation: The monitor *MonitorName* is not available for the platform *Platform*, therefore it cannot be used by this resource model when running on the given platform. Anyway the package building will proceed.

User Response: Do not use the monitor *MonitorName* on the platform *Platform*, or import a monitor source that supports the given platform.

AMW0647E Probe *ProbeName* collection test failed.

Explanation: Error *ErrorDescription* occurred during probe collection test.

User Response: Ensure that the probe is available on w32-ix86.

AMW0648E No javascript debugger available

Explanation: No javascript debugger has been found on this machine. It is not possible to debug javascript resource models unless you install a javascript debugger

User Response: Install a javascript debugger. To download the latest version of the Microsoft Script debugger please visit: http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=280011

Index

E

F

H

I

K

L

M

N

O

P

R

S

T

V

W

SH19-4571-00