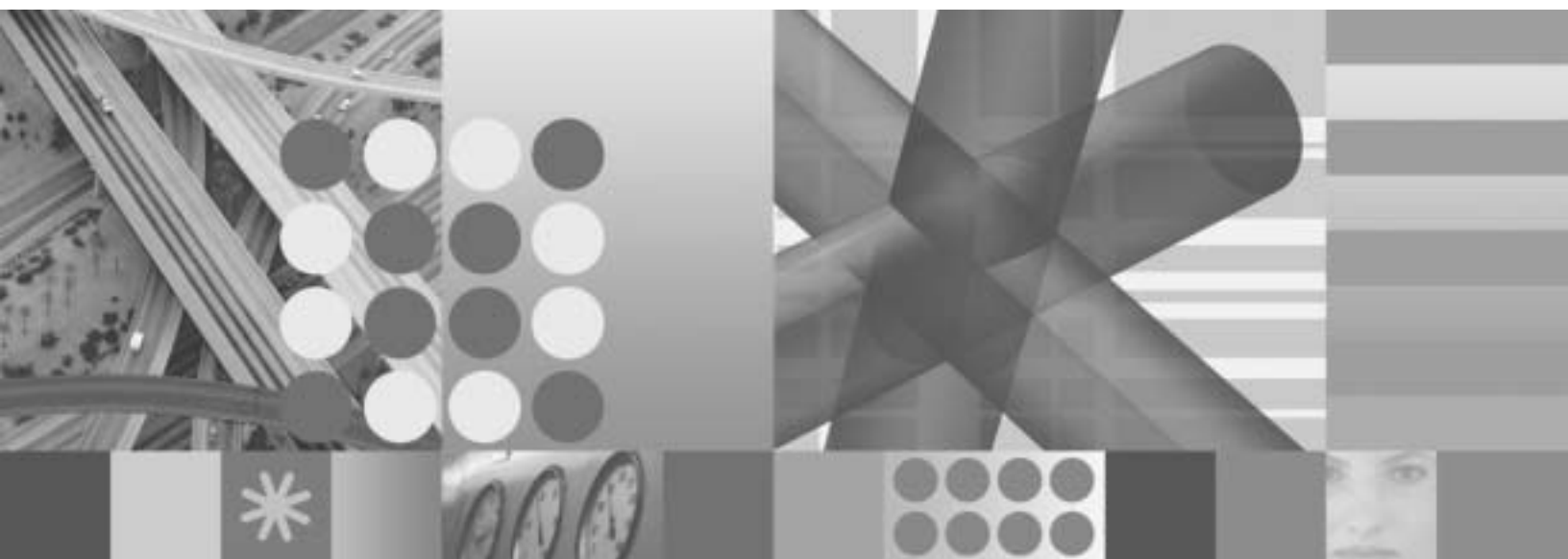


Version 4.3.1



User's Guide for Inventory



User's Guide for Inventory

Note

Before using this information and the product it supports, read the information in "Notices" on page 309.

This edition applies to version 4, release 3, modification level 1 of IBM Tivoli Configuration Manager (program number 5724-C06) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC23-4713-05.

© **Copyright International Business Machines Corporation 2002, 2008.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this guide	vii
What this guide contains	vii
Publications	viii
IBM Tivoli Configuration Manager library	viii
Prerequisite publications	ix
Related publications	ix
Accessing publications online	x
Ordering publications	x
Accessibility	x
Tivoli technical training	xi
Support information	xi
Conventions used in this guide	xi
Typeface conventions	xi
Operating system-dependent variables and paths	xii

Chapter 1. Introduction to Inventory . . . 1

Components used by Inventory	1
How does Inventory work?	3
How scalable collection works	4
How data is collected with SCS	4
Inventory and SCS features	7
Inventory features	7
SCS features	9
Security	9
The computer system ID	9

Chapter 2. Working with SCS 13

Using collectors	13
Stopping and starting a collector	14
Configuring a collector	14
Configuring the run-time directory and depot	14
Configuring the collector log file	15
Configuring threads	15
Configuring checkpoint mode	15
Modifying the collector hierarchy	16
Using the Inventory data handler	16
Configuring the Inventory data handler	16
Moving the Inventory data handler	17
Moving the Inventory callback object	17
Creating and configuring RIM objects for the inventory data handler	17
Creating RIM objects	18
Configuring RIM objects	19
Moving RIM objects	19
Scheduling collections	20
Controlling the flow of network data	21
Obtaining information about SCS	22
Viewing collector settings	23
Viewing collector status	23
Viewing information about queues and CTOCs	23
Viewing inventory data handler settings	24
Viewing information about inventory scans	24
Viewing inventory profile settings	24
Data stored in the configuration repository	24
Configuring inventory status information	24

Chapter 3. Working with Inventory profiles 27

Using inventory profiles	27
Creating an inventory profile	28
Desktop	29
Command line	31
Customizing an inventory profile	32
Customizing global properties	33
Customizing a software scan	34
Software scan options for PC	34
Software scan options for UNIX and OS/400	36
Software scan configuration options	36
Scan Options	38
Files	39
Directories	39
Software patch scan options for PC	39
Customizing a hardware scan	41
PC hardware scans	41
Tivoli hardware scanner	41
DMI scanner	42
UNIX and OS/400 hardware scans	42
Using custom scripts and MIF files	44
Customizing scans of pervasive devices	45
Customizing inventory profiles from the command line	46
Cloning an inventory profile	47
Desktop	47
Command line	48
Deleting an inventory profile	48
Desktop	48
Command line	49
Renaming an inventory profile	49

Chapter 4. Distributing Inventory profiles 51

Distributing inventory profiles	51
Desktop	52
Command line	55
Performing an endpoint-initiated scan	55
Scanning disconnected systems	56
Populating the configuration repository	58
Cancelling scans	58

Chapter 5. Collecting custom information with Inventory 59

Using signatures	59
Modifying signatures	61
Desktop	62
Command Line	62
Defining signatures	62
Using signature packages	64
Desktop	65
Command Line	65
Using custom MIF files	66

Using the custom scanning program	67
Designing custom tables	67
Creating history tables for custom tables.	68
Adding a custom table to the configuration repository	69
Running a query using a custom MIF file	70
Collecting information with UserLink	70
Overview	71
Planning to collect user information	72
Giving users access to UserLink	72
Desktop	72
Command Line	74
Creating MIF groups and attributes with the user data template.	75
Using the userlink.htm file	83
Completing the user data form	83
Retrieving and saving user information	84
Viewing custom information.	85

Chapter 6. Querying inventory information 87

Using the inventory, pervasive, and subscription queries	87
Creating a query.	88
Desktop	88
Command line	92
Running a query	92
Desktop	92
Command line	94
Defining subscribers with queries	94
Desktop	94
Command line	96
Querying for information about one client	96
Desktop	97
Command line	98
Using queries in interconnected Tivoli regions.	98

Appendix A. Authorization roles . . . 101

Inventory roles	101
Inventory operations	101
Query operations	102

Appendix B. Commands. 103

Using Tivoli commands	103
Command line syntax	103
Examples of command syntax.	104
Object references	104
Registered names	104
Object paths.	104
Inventory commands.	105
Command syntax	107
wcancelscan	108
wcollect	110
wcrtinvcb.	116
wcrtinvdh	117
wcstat	120
wdistinv	122
wepscan	127
wgetinvdh	131
wgetinvglobal	133

wgetinvpcfiles	136
wgetinvpchw	138
wgetinvpcsw	140
wgetinvpvdconfig	143
wgetinvpvdhw	144
wgetinvpvdsw	145
wgetinvswd	146
wgetinvunixfiles	147
wgetinvunixhw.	149
wgetinvunixsw	151
wgetscanstat.	154
winvdeps.	156
winvfilter.	157
winviso	159
winvmgr	161
winvmigrate.	166
winvpackage	167
winvrnode	170
winvsig	172
winvupdatesid	175
wloadiso	178
wmvinvcb	180
wmvinvdh	181
wqueryinv	182
wsetinvdh	184
wsetinvglobal	187
wsetinvpcfiles	192
wsetinvpchw	196
wsetinvpcsw	199
wsetinvpvdconfig	203
wsetinvpvdhw	204
wsetinvpvdsw	205
wsetinvswd	206
wsetinvunixfiles	207
wsetinvunixhw	211
wsetinvunixsw	214
wtransfer	217
wwaitscan	219

Appendix C. Policy 221

Default policy methods	221
Policy objects	222
Creating a new policy object	222
Replacing the contents of a policy method.	223
Assigning policy to a policy region	224
Example—Setting a default policy method	224
Policy methods.	227
ic_def_global_action	229
ic_def_global_ep_timeout	230
ic_def_global_logfile_host	231
ic_def_global_logfile_path	232
ic_def_global_notice_interval	233
ic_def_global_notice_location	234
ic_def_global_notice_type	235
ic_def_global_security	236
ic_def_global_update_replace	237
ic_def_pc_custom_before_script	238
ic_def_pc_custom_mifs	239
ic_def_pc_custom_script.	240
ic_def_pc_exclude_dirs	241
ic_def_pc_extensions	242

ic_def_pc_hw_gran	243
ic_def_pc_hw_outfile	244
ic_def_pc_hw_scan	245
ic_def_pc_include_dirs	246
ic_def_pc_sw_crc	247
ic_def_pc_sw_flags	248
ic_def_pc_sw_outfile	249
ic_def_pc_sw_scan.	250
ic_def_pvd_config_scan	251
ic_def_pvd_hw_scan	252
ic_def_pvd_sw_scan	253
ic_def_unix_custom_before_script	254
ic_def_unix_custom_mifs	255
ic_def_unix_custom_script	256
ic_def_unix_exclude_dirs	257
ic_def_unix_extensions	258
ic_def_unix_hw_gran.	259
ic_def_unix_hw_outfile	260
ic_def_unix_hw_scan	261
ic_def_unix_include_dirs	262
ic_def_unix_sw_crc	263
ic_def_unix_sw_flags	264
ic_def_unix_sw_outfile	265
ic_def_unix_sw_scan	266

Appendix D. XML schema definition for signature catalogs. 267

Appendix E. Installing and uninstalling Common Inventory Technology (CIT). 273

Installing Common Inventory Technology (CIT)	273
Uninstalling Common Inventory Technology (CIT)	275
Managing the Inventory bundle dependency set	276
winvdeps.	277
Scanning virtual environments	278
Starting the enabler	278
Enabler return codes	279

Appendix F. Troubleshooting. 281

AutoTrace	281
Configuration repository schema	281
DMI	281
Adding a DMI layer	281
DMI scans	283
DMI table and column names	283
Endpoints	283
Enabling endpoint log files.	283
Installing multiple endpoints on the same workstation	284
Endpoint-initiated scans	285
Graphical user interface	286
System requirements	286

Log files	286
UNIX Systems	286
PC Systems	287
Running the GUI on UNIX systems	287
Logging in	287
Properties window does not open	288
Scalable Collection Service	288
Log and data files	288
Queue Data Files	288
Collector Log File	289
Inventory Data Handler Log File	289
Collection Manager Log File	289
Inventory Status Log File	289
CTOC Completed Status Log File	290
Activity Planner and Web Interface	290
Other Log Files.	291
Depot contents	292
Troubleshooting procedures	292
Common Inventory Technology	295
Common Inventory Technology traces	296
Common Inventory Technology scanners	296
Common Inventory Technology installations	299
Queries	300
RIM objects	301
Scans	301
Cancelling scans	301
Profile distribution failures	302
Software scan with the option "Scan Registry for Product Information".	302

Appendix G. Accessibility 303

Navigating the interface using the keyboard	303
Magnifying what is displayed on the screen	304

Appendix H. Support information . . . 305

Searching knowledge bases.	305
Search the information center on your local system or network.	305
Search the Internet	305
Obtaining fixes	305
Contacting IBM Software Support	306
Determine the business impact of your problem	307
Describe your problem and gather background information	307
Submit your problem to IBM Software Support	307

Notices 309

Trademarks	311
----------------------	-----

Glossary 313

Index 317

About this guide

This guide provides information about the Inventory component of IBM® Tivoli® Configuration Manager, Version 4.3.1.

Inventory enables you to manage hardware and software inventory on the systems in your enterprise. Inventory uses profiles to scan for information and then stores the scan data in the IBM Tivoli Configuration Manager configuration repository. This guide includes an introduction to the Inventory component and information about, procedures for, and examples of the management tasks that you can complete using Inventory.

Throughout the book, the changed or new sections are marked by revision bars.

This guide is for system administrators and other users of Inventory who perform inventory gathering and reporting operations about systems in a distributed enterprise.

Readers should be familiar with the following:

- PC, UNIX®, and OS/400® systems and pervasive devices
- Basic inventory control and system configuration management concepts
- Database and SQL concepts
- Graphical user interfaces

What this guide contains

This guide contains the following sections:

- Chapter 1, “Introduction to Inventory,” on page 1
This chapter provides an overview of the components that make up Inventory, an explanation of how Inventory works, a list of features, and information about configuring security for Inventory tasks and information.
- Chapter 2, “Working with SCS,” on page 13
This chapter provides information about configuring and using Scalable Collection Service (SCS), a Tivoli Management Framework service that enables efficient, asynchronous collection of large amounts of data across complex networks.
- Chapter 3, “Working with Inventory profiles,” on page 27
This chapter describes the steps needed to create, customize, clone, and delete inventory profiles.
- Chapter 4, “Distributing Inventory profiles,” on page 51
This chapter provides the steps that you take to distribute inventory profiles from the Tivoli desktop and from UserLink. It also describes how to initiate a scan from an endpoint and how to scan a machine outside of your Tivoli management region (Tivoli region).
- Chapter 5, “Collecting custom information with Inventory,” on page 59
This chapter provides the procedures for collecting custom and optional information with Inventory. You can collect custom information by adding signatures to the configuration repository. You can also add custom tables, collect custom information with UserLink, and populate the custom tables.

- Chapter 6, “Querying inventory information,” on page 87
This chapter describes the tasks necessary to search for information in the configuration repository. The Tivoli Management Framework query facility enables you to construct SQL queries to retrieve information gathered from inventory profile distributions.
- Appendix A, “Authorization roles,” on page 101
This appendix contains tables that describe the roles you need to perform activities using Inventory.
- Appendix B, “Commands,” on page 103
This appendix lists and documents, in alphabetical order, the Tivoli commands related to Inventory and SCS.
- Appendix C, “Policy,” on page 221
This appendix includes an explanation of policies along with a description of each policy included with Inventory.
- Appendix F, “Troubleshooting,” on page 281
This appendix provides troubleshooting information for Inventory.
- Appendix G, “Accessibility,” on page 303
This appendix provides information about the accessibility features of Inventory.
- Appendix H, “Support information,” on page 305
This appendix provides information for obtaining support for IBM products.

Publications

This section lists publications in the IBM Tivoli Configuration Manager library and any other related documents. It also describes how to access Tivoli publications online and how to order Tivoli publications.

IBM Tivoli Configuration Manager library

The following documents are available in the IBM Tivoli Configuration Manager library:

- *Release Notes*, GI11-0926
Contains the latest information about this release of IBM Tivoli Configuration Manager, including installation and upgrade notes; software limitations, problems, and workarounds; documentation notes; and internationalization notes.
- *Planning and Installation Guide*, GC23-4702
Explains how to plan or upgrade your deployment of IBM Tivoli Configuration Manager in a Tivoli environment as well as how to install, upgrade, and uninstall the components of IBM Tivoli Configuration Manager using the available installation mechanisms.
- *Introducing IBM Tivoli Configuration Manager*, GC23-4703
Explains the concepts of IBM Tivoli Configuration Manager and its components and provides a road map to the IBM Tivoli Configuration Manager documentation.
- *User's Guide for Software Distribution*, SC23-4711
Explains the concepts and procedures necessary to effectively distribute software over networks using the Software Distribution component of IBM Tivoli Configuration Manager.
- *Reference Manual for Software Distribution*, SC23-4712

Provides in-depth information about the IBM Tivoli Configuration Manager commands used by the Software Distribution component and explains advanced features, concepts, and procedures necessary to effectively use the Software Distribution component.

- *Database Schema Reference*, SC23-4783

Describes the schema used by the IBM Tivoli Configuration Manager configuration repository.

- *Messages and Codes*, SC23-4706

Provides details of the messages generated by the IBM Tivoli Configuration Manager components.

- *Patch Management Guide*, SC23-5263

Provides details of the patch management.

- *IBM Tivoli Configuration Manager: Guide for Active Directory Integration*, SC32-2285

Describes the integration of Microsoft Active Directory with your Tivoli environment.

- *IBM Tivoli Configuration Manager: License Management Extension*, SC32-2260

Describes the license management facilities provided in your Configuration Manager environment.

- *IBM Tivoli Configuration Manager: User's Guide for Operating System Deployment Solution*, SC32-2578

Describes how you can implement an operating system deployment solution delivered with Configuration Manager.

- *Inventory Online Help*

Provides related information about using the Inventory graphical user interface (GUI).

Prerequisite publications

The following documents provide information that you need to set up your Tivoli environment and install IBM Tivoli Configuration Manager:

- *Tivoli Management Framework: Planning for Deployment Guide*, GC32-0803

Explains how to plan for deploying your Tivoli environment. It also describes Tivoli Management Framework and its services.

- *Installation Guide*, GC32-0804

Explains how to install and upgrade Tivoli Enterprise™ software within your Tivoli region using the available installation mechanisms provide by Tivoli Software Installation Service and Tivoli Management Framework. Tivoli Enterprise software includes the Tivoli management region server (Tivoli server), managed nodes, gateways, endpoints, and RDBMS interface module (RIM) objects. This guide also provides information about troubleshooting installation problems.

- *Tivoli Management Framework: Release Notes*, GI11-0890

Describes the latest installation information, including supported platforms, defects, and limitations for Tivoli Management Framework.

Related publications

The *Tivoli Management Framework: Reference Manual*, SC32-0806, provides in-depth information about Tivoli Management Framework commands. This manual is helpful when writing scripts that are later run as Tivoli tasks. This manual also documents default and validation policy scripts used by Tivoli Management Framework.

The *Tivoli Software Glossary* includes definitions for many of the technical terms related to Tivoli software. The *Tivoli Software Glossary* is available at the following Tivoli software library Web site:

<http://publib.boulder.ibm.com/tividd/glossary/tivoliglossarymst.htm>

Accessing publications online

The documentation CD contains the publications that are in the product library. The format of the publications is PDF, HTML, or both. Refer to the readme file on the CD for instructions on how to access the documentation.

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli software information center Web site. Access the Tivoli software information center by first going to the Tivoli software library at the following Web address:

<http://www.ibm.com/software/tivoli/library/>

Scroll down and click the **Product manuals** link. In the Tivoli Technical Product Documents Alphabetical Listing window, click the IBM Tivoli Configuration Manager link to access the product library at the Tivoli software information center.

Note: If you print PDF documents on other than letter-sized paper, set the option in the **File** » **Print** window that allows Adobe Reader to print letter-sized pages on your local paper.

Ordering publications

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

<http://www.elink.ibm.link.ibm.com/public/applications/publications/cgi-bin/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, see the following Web site for a list of telephone numbers:

<http://www.ibm.com/software/tivoli/order-lit/>

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate the features of the graphical user interface.

For additional information, see Appendix G, “Accessibility,” on page 303.

Tivoli technical training

For Tivoli technical training information, refer to the following IBM Tivoli Education Web site:

<http://www.ibm.com/software/tivoli/education>

Support information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

- Searching knowledge bases: You can search across a large collection of known problems and workarounds, Technotes, and other information.
- Obtaining fixes: You can locate the latest fixes that are already available for your product.
- Contacting IBM Software Support: If you still cannot solve your problem, and you need to work with someone from IBM, you can use a variety of ways to contact IBM Software Support.

For more information about these three ways of resolving problems, see Appendix H, “Support information,” on page 305.

Conventions used in this guide

This guide uses several conventions for special terms and actions and operating system-dependent commands and paths.

Typeface conventions

This guide uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:**, and **Operating system considerations:**)
- Keywords and parameters in text

Italic

- Words defined in text
- Emphasis of words (words as words)
- New terms in text (except in a definition list)
- Variables and values you must provide

Monospace

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Revision bars (|)

Throughout the book, the changed or new sections

Operating system-dependent variables and paths

This guide 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. The names of environment variables are not always the same in Windows and UNIX. For example, %TEMP% in Windows is equivalent to \$tmp in UNIX.

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

Chapter 1. Introduction to Inventory

The complexity of client/server management increases as enterprise-wide networks span broad geographic locations and incorporate systems and software from a variety of vendors. Tivoli Management Framework is a distributed environment designed specifically to deploy secure, heterogeneous, and scalable systems management applications. These applications, such as the Inventory component of IBM Tivoli Configuration Manager, provide solutions for managing a complex client/server environment. Inventory also uses Scalable Collection Service (SCS), a Tivoli Management Framework service that enables efficient, asynchronous collection of large amounts of data across complex networks.

Using traditional methods to assess inventory, you must travel to each system, write down the software and hardware inventory information you collect, and enter the information into a spreadsheet or database program. When users upgrade software and hardware, you must update the spreadsheet or database.

This method is time-consuming and difficult to keep current. As a result, administrators and accounting personnel cannot automatically reuse inventory information to perform system and software upgrades and management tasks. However, with Inventory, gathering and maintaining up-to-date inventory information in a distributed environment is quick, accurate, and easy.

Inventory gathers data about hardware and software to help system administrators and accounting personnel manage complex distributed client/server enterprises. With Inventory, administrators and accounting personnel can perform the following tasks:

- Manage all enterprise systems from a central point
- Determine the installed software base
- Confirm a software distribution
- Supplement and replace physical inventory functions
- Assist in procurement planning
- Check software requirements
- Control assets

This chapter provides an overview of how Inventory and SCS work, a list of features, a description of how Inventory works with other Tivoli applications, and information about configuring security for Inventory tasks and information.

Components used by Inventory

Inventory relies on the following components to collect and store information:

- The Tivoli management region server (Tivoli server), which is responsible for issuing scan requests to the target systems.
- The Inventory application, which includes both the graphical user interface (GUI) components and the command line interface (CLI) utilities.
- The Inventory Gateway application, which allows a managed node to act as a gateway for distributing inventory profiles to endpoints.
- The relational database management system (RDBMS) server and the *configuration repository* in the RDBMS. The configuration repository is the RDBMS

that contains the schema (tables and columns) in which software and hardware inventory information is stored. The configuration repository schema provides a structure for storing information collected during an inventory scan.

- The *RDBMS interface module (RIM) host*, which enables Inventory to communicate with an RDBMS.
- One or more RIM objects, which connect Inventory to the RDBMS for access to the configuration repository. You can configure multiple RIM objects to write SCS data in parallel to the configuration repository.
- The *Web Gateway* component, if you plan to distribute scans to pervasive devices or allow users to perform inventory scans using the Web Interface.

SCS uses the following components for Inventory collections:

- *Repeater sites* organized into a *repeater hierarchy*. Repeater sites are systems that use the *multiplexed distribution* (MDist 2) service. MDist 2 parameters control the way that information is distributed throughout a Tivoli environment. A repeater hierarchy is the order in which information flows from one repeater to the next and then to the targets of the distributed data.

SCS uses a *collector hierarchy* that mirrors the MDist 2 repeater hierarchy. SCS sends data upstream through this hierarchy, in the opposite direction of MDist 2 distributions.

See the *Tivoli Management Framework Planning for Deployment Guide* for more information about configuring a repeater hierarchy.

- *Collectors*, which are repeater sites on which SCS has been installed. Specifically, a collector is an SCS daemon process on either a managed node or gateway that stores and then forwards data to other collectors or to the inventory data handler. This guide refers to any managed node as a collector if SCS has been installed on the node and the node is part of the repeater hierarchy.

Collectors are composed of the following components:

- The *depot*, which persistently stores data that has been collected from scan targets or other collectors.
- The *queues*, which hold the *collection table of contents* (CTOCs). CTOCs contain, among other things, the name and size of the data file and the source and destination of the collected data. The input queue controls the order in which CTOCs are processed for collection. The output queue controls the order of the CTOCs as they are sent out from the collector. The completed, deferred, and error queues hold CTOCs for completed and deferred data collection and error conditions, respectively.
- A multithreaded scheduler daemon that processes input and output queues and controls data flow through the collector depot.
- A *collection manager*, which maintains the collector hierarchy based on repeater hierarchy information obtained from the MDist 2 repeater manager.
- An *inventory data handler*, which is the Inventory object that receives data from collectors and sends the data to one or more RIM objects. The inventory data handler can be considered the final collector in a Inventory system. Like collectors, the inventory data handler has a depot and queues. However, the inventory data handler uncompresses and decodes the data and sends it to one or more RIM objects rather than requesting collection from an upstream collector.
- The *status collector*, which collects, stores, and distributes status information for each target in a scan. You can configure the status collector to keep lists of

completed scans, successful scans, failed scans, and error messages. The status collector maintains this information throughout the scan, so scan status information is available during the scan.

The status collector is installed on the same managed node as the inventory data handler. If that node fails and is restarted, status information tracked by the status collector is automatically restored.

- The *inventory callback object*, which performs the following functions:
 - When inventory data cannot be collected using the collector hierarchy, MDist 2 sends the data to the inventory callback object, which sends it to the inventory data handler. The data is then sent through one or more RIM objects to the configuration repository.
 - For endpoint scans, MDist 2 sends a status message to the inventory callback object indicating whether it successfully delivered the inventory profile to the target. This message indicates only that the endpoint processed the profile, not that the scan data successfully reached the configuration repository.
- A *pervasive callback object*. For pervasive device scans, MDist 2 sends a status message to a special pervasive callback object. This message indicates only whether the job for the devices to be scanned was created successfully on the Web Gateway component; the data is not sent through one or more RIM objects to the configuration repository until a device actually connects to the Web Gateway component and a scan occurs.

How does Inventory work?

Inventory relies on resources and management concepts provided by Tivoli Management Framework. Inventory supports the concept of *management by subscription*, which is the concept of managing network resources by creating sets of profiles and distributing the profiles (through *profile managers*) to physical entities called subscribers. A profile is a container for application-specific (in this case, Inventory-specific) information and a profile manager is a container for a set of profiles that have subscribers, or systems targeted for distribution in common.

Valid targets for an inventory profile distribution are profile managers, *endpoints*, and *resource groups*. An endpoint is a system that is managed from another system, called a gateway. It is used as a target of an operation such as receiving a profile or executing a task. A resource group is a type of subscriber that can contain pervasive devices or users as targets of operations.

To use Inventory, you must first create and customize at least one inventory profile within a profile manager. Then you must distribute the inventory profile to subscribers. The profile then performs an action on each target system. It can distribute a configuration file to the system, run a scan on the system, or both.

For endpoint systems, the scanner creates *MIF (Management Information Format)* files. MIF files are ASCII files that contain information in a standard format. These files are collected during hardware, software, or custom scans. Next, Inventory processes the information in MIF files and uses SCS or MDist 2 to send the data to the configuration repository.

For pervasive devices, Inventory processes the information forwarded from the Web Gateway component and uses SCS to send the data to the configuration repository.

After Inventory updates the configuration repository, you can view the information with the Tivoli Management Framework query feature. For example, you can

query the configuration repository for all systems that have an outdated version of a software product that will need upgrading in the next year.

Inventory also works with SCS to manage the flow of data efficiently across your network.

How scalable collection works

Scalable collection enables efficient, asynchronous collection of large amounts of data across complex networks. With scalable collection, you can significantly reduce the time needed to scan and store vast amounts of data. SCS gives you the ability to scale data collection.

SCS sends scan data to its destination as the scan on each target completes, so it sends smaller chunks of data through the network. Another advantage of SCS is its ability to stage data through a hierarchy of collector nodes, which distributes the processing load across the Tivoli management region (Tivoli region). In essence, SCS more effectively manages the stream of data while creating multiple pathways to the configuration repository. This functionality results in faster scans and increased control of network resources.

SCS is triggered by a target that has data to be collected. This target informs the gateway that the data is ready by sending the CTOC. A collector daemon running on the gateway then retrieves the data from the target and stores it on the collector in persistent storage. Depending on the configuration of your network, the data may then travel through a hierarchy of collectors before arriving at its destination.

With Inventory, all collectors send data to the inventory data handler, which then sends the data to the configuration repository.

How data is collected with SCS

Figure 1 illustrates the way SCS and Inventory work together to complete scans and store information in the configuration repository.

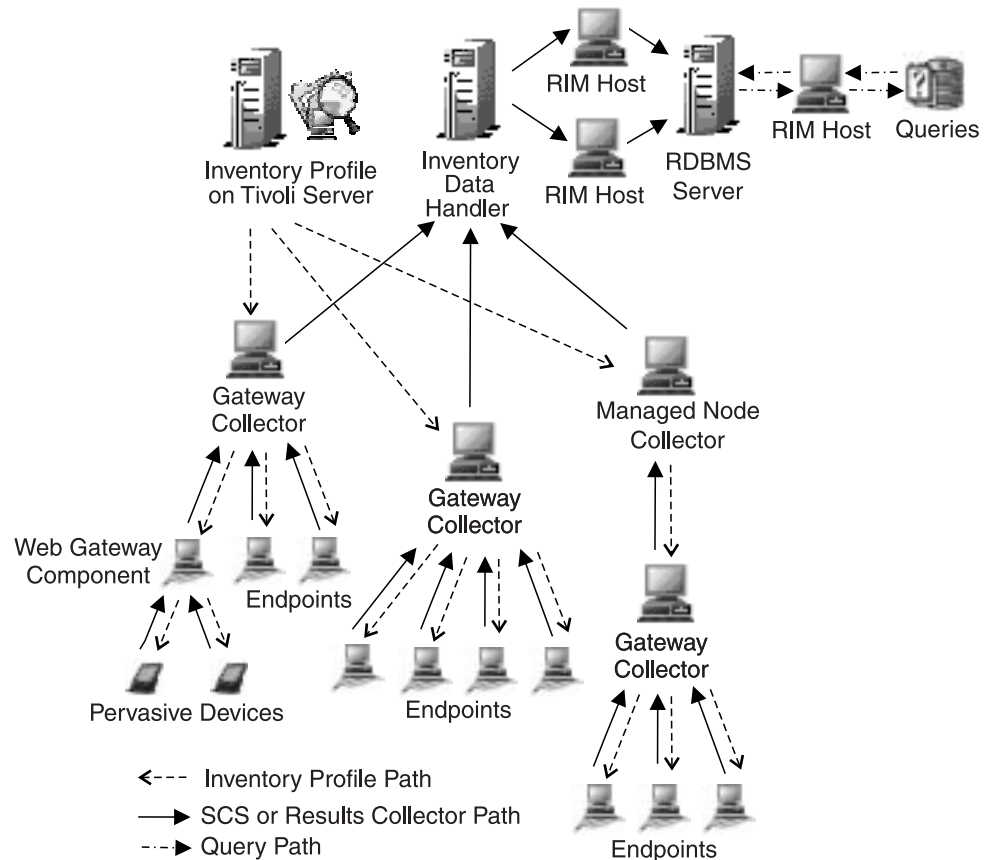


Figure 1. Data Collection Using SCS and Inventory

Figure 2 shows the role of the inventory callback object in data collection.

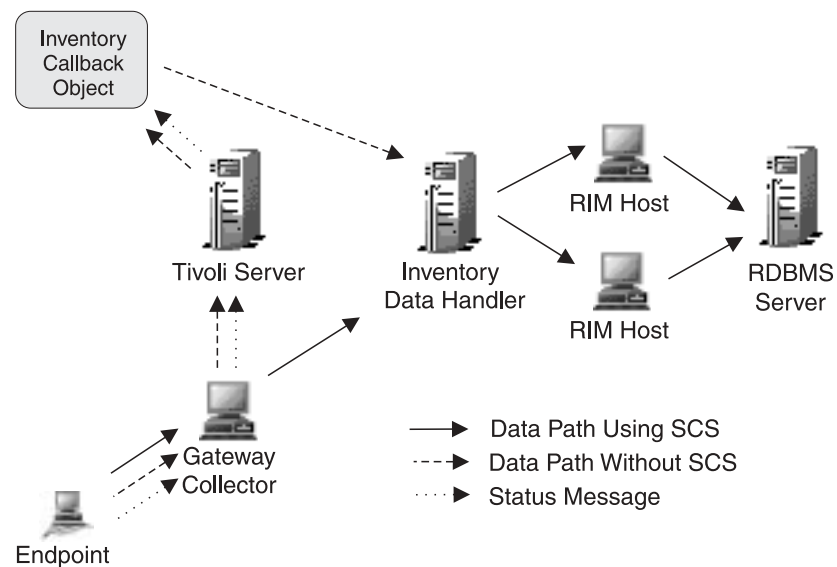


Figure 2. The Role of the Inventory Callback Object in Data Collection

Data collection using Inventory and SCS occurs in three major phases:

- First, the inventory profile scans targets, which causes the targets to send CTOCs to SCS.

- Next, SCS processes the CTOCs and moves data through the collector hierarchy to the inventory data handler.
- Finally, the inventory data handler writes the data to one or more RIM objects, which send the data to the configuration repository.

Table 1 describes the data collection process in detail and shows whether Inventory or SCS performs each action. This table applies to scans of all systems other than pervasive devices.

Table 1. Inventory Data Collection from Endpoints

Action	Component or Service
You create an inventory profile.	Inventory
You distribute the profile to endpoints.	Inventory
The profile distribution completes.	Inventory
The following events occur on an endpoint as the scan of the endpoint completes: <ol style="list-style-type: none"> 1. A MIF file is created. 2. The MIF file is parsed. 3. Scan data is generated. 4. The scan data is encoded, compressed, and saved in a file. 	Inventory
After the scan finishes on an endpoint, the endpoint makes an upcall to the gateway to request data collection by sending a CTOC. Note: If the endpoint cannot successfully forward the CTOC to the collector, the scan data for that endpoint is sent using MDist 2 and without using SCS.	Inventory
The gateway forwards the CTOC to the local collector.	SCS
When the collector on the gateway receives the CTOC, the following events occur: <ol style="list-style-type: none"> 1. The collector places the CTOC in the input queue. 2. The collector makes a downcall to the endpoint to retrieve the compressed and encoded data file. 3. The collector stores the data file in the collector depot. The CTOC moves from the input queue to the output queue. 4. The collector requests data collection from the next collector in the hierarchy (the upstream collector) by sending the CTOC to the upstream collector. 5. The upstream collector retrieves the data from the downstream collector. When the data transfer is complete, the CTOC in the output queue of the downstream collector is discarded. 6. Steps 1 through 5 repeat until the data file reaches the inventory data handler. 	SCS
The inventory data handler stores CTOCs in an input queue and scan data in a depot in the same way that collectors do. However, rather than sending a collection request to another collector, the inventory data handler uncompresses the scan data and sends it to an available RIM object.	Inventory
From the RIM object or objects, the information is sent to the configuration repository. The status collector collects the final status for each endpoint.	Inventory

Table 2 describes the data collection process for scans of pervasive devices.

Table 2. *Inventory Data Collection from Pervasive Devices*

Action	Component or Service
You create an inventory profile.	Inventory
You distribute the profile to a resource group for pervasive devices.	Inventory
Jobs are created on the Web Gateway components for the devices to be scanned. One job is created for each type of device; for example, one job for all Palm devices and one job for all Windows CE devices.	Inventory
A device connects to the Web Gateway component.	Web Gateway component
The device is scanned.	Web Gateway component
The scan data is encoded, compressed, and saved in a file on the Web Gateway component. Scan data for multiple devices can be saved in a single file.	Web Gateway component
The Web Gateway component makes an upcall to the gateway to request data collection by sending a CTOC. At this point, the scan data is sent through the collector hierarchy to the configuration repository as shown in the previous table. Note: The status collector maintains information for each device, not just for the entire Resource Group.	SCS

Inventory and SCS features

In addition to gathering information about systems in your enterprise, Inventory provides a central point of management for the systems in a distributed client/server environment. The following sections describe the features of Inventory and SCS.

Inventory features

Inventory offers the following features:

- A flexible, standardized data storage architecture—Inventory uses *Desktop Management Task Force (DMTF)* 2.0 MIF files to hold all the information gathered during an inventory scan. Inventory also stores the contents of MIF files in the configuration repository.

Note: DMTF is a consortium whose goal is to define standard syntax for data that describes how to manage the components of desktop computers.

- Extensive customization options for the inventory profile—You can configure an inventory profile to do any of the following during a distribution to endpoint systems:
 - Scan hardware and software—Inventory can scan target systems for hardware information, software information, or both. You can customize a software scan to scan particular directories and files on systems that are not pervasive devices.

On PCs, hardware scans automatically scan for BIOS information in addition to other hardware components. Also, you can choose to scan for Desktop Management Information (DMI).

- Run a script—You can include a script with an inventory profile that runs on target systems (not pervasive devices) before or after a scan during a profile distribution. Scripts can, among other things, run custom scanners that collect information and create custom MIF files from that information.
- Read hardware, software, or custom MIF files—Inventory can read the MIF files that are created by either a hardware or software scan or the MIF files that you create for systems that are not pervasive devices.

See Chapter 3, “Working with Inventory profiles,” on page 27, and Chapter 4, “Distributing Inventory profiles,” on page 51 for more information about creating, customizing, and distributing inventory profiles.

- Configurable hardware scanning—You can select the type of information to be returned from a hardware scan and reduce the amount of data generated by the scan. This reduction in data can improve the speed with which the scan runs, reduce the network bandwidth required to transport the scan data, and reduce the amount of time to write the data into the configuration repository.
- The Windows Management Instrumentation (WMI) scanner—Inventory can get data from endpoints using WMI to receive information stored in the Common Information Model (CIM) on Windows platforms supporting CIM and WMI for common inventory attributes. This feature is used automatically when it is available.
- System Management (SMBIOS) scanner—Inventory can get more detailed data from SMBIOS-complaint endpoints. Systems that use the SMBIOS specification can return details such as a system’s asset tag ID and chassis type.
- Endpoint-initiated scans—You can initiate a scan from an endpoint instead of distributing the scan from an inventory profile. This feature has many uses. For example, you can use endpoint-initiated scans to perform the following actions:
 - Scan a machine each time it is rebooted
 - Scan a laptop system each time a mobile user logs in to the network
 - Scan a machine that is disconnected from the network
- An Inventory Query window through which you can search the configuration repository for a single system that meets your criteria. See Chapter 6, “Querying inventory information,” on page 87 for more information.
- History tracking—You can create history tables that record any changes to the scan data in the configuration repository. If you do not create these tables, Inventory does not track any changes to scan data. See the *Database Schema Reference* for more information.
- Mobile endpoint support—You can create a profile for a customized inventory scan that can be pushed to a mobile endpoint. A mobile endpoint (also called a roaming endpoint) is an endpoint that is regularly disconnected from and reconnected to the network, such as a laptop. The scan can be scheduled to run when the endpoint is disconnected; the data is sent the next time the endpoint is connected to the network.
- Wake on LAN[®] technology—With MDist 2 functionality, Inventory can automatically wake up systems that have a Wake on LAN-enabled network card using the Intel Wired for Management specification.
- Scan cancel—You can cancel any or all ongoing inventory scans before the scans complete. With this command line interface (CLI) option, you can cancel scans at certain stages during the scan process. See “wcancelscan” on page 108 for more information.
- Pervasive device management—Inventory can scan pervasive devices, such as handheld devices, for hardware, software, and user-customized settings.

SCS features

The following SCS features illustrate the advantages of asynchronous data collection:

- Asynchronous data collection and asynchronous processing lead to better distribution of the processing load across more nodes in the Tivoli region, as well as better use of the network. SCS stages the data at different collectors in the network as it flows to the configuration repository, and it allows control over the data flow between successive collectors in the hierarchy.
- SCS returns scan data as the scan of each target completes. This feature reduces RIM object overload at the end of a distribution.
- After data has been collected from a target, that target can be disconnected without affecting SCS.
- The inventory data handler can write data in parallel to one or more RIM objects, each of which has one or more connections.
- SCS reduces the load on the Tivoli server. The inventory data handler stages scan data for individual targets while the configuration repository is updated. (It is recommended that you create the inventory data handler on a system other than the Tivoli server.)
- Using SCS and the Tivoli scheduler, you can schedule scans and data collection traffic for times when network traffic is at a minimum.
- Collectors in the collector hierarchy store collected data in depots. If a failure occurs, the collector can resend the data when it is back online, so you do not have to scan all the targets again.
- SCS provides functionality through which you can get information about the status of collectors, CTOCs, and scans.
- With SCS, you can determine the status of individual targets as they complete, rather than having to wait until all targets complete.
- SCS allows you to retry collecting data from targets or other collectors following a failure. You can also set the specific number of retries.

See Chapter 2, “Working with SCS,” on page 13, for more information about using the features of SCS.

Security

Inventory functions within the bounds of Tivoli Management Framework security to ensure secure and authorized use of the application. In other words, each administrator who wants to work with Inventory must be given the proper Tivoli security authorization, or role, to complete inventory tasks. For more information about assigning roles, see the *Tivoli Management Framework: User's Guide*.

Many procedures in this guide are prefaced with a table showing the role required for the procedure. See Appendix A, “Authorization roles,” on page 101, for more information about the inventory authorization roles required for each operation.

The computer system ID

IBM Tivoli Configuration Manager uses a value called the computer system ID to identify endpoints. Prior to Tivoli Configuration Manager, Version 4.2, Software Distribution and Inventory used a value that Tivoli Management Framework generated for each endpoint as the computer system ID. If you reinstalled the endpoint code, on some platforms this value was replaced with a new value. This

situation created the potential for duplicate computer system IDs to exist in the configuration repository for a single system.

To reduce the potential for duplicate computer system IDs, Tivoli Configuration Manager, Version 4.2, can store the computer system ID on each endpoint in a persistent location. On Windows and UNIX systems, the computer system ID persists even if endpoint code is deleted, reinstalled, or upgraded. Moreover, for SMBIOS-compliant systems, the computer system ID is generated using the motherboard serial number and the system serial number.

For Windows endpoints, Tivoli Configuration Manager generates the computer system ID using the following algorithm:

1. It checks the persistent storage location on the endpoint for an existing computer system ID. If a computer system ID is found, that value is used.
2. If no computer system ID is found and the scan was initiated using the Web Interface, it checks the most recently installed endpoint on the system for the latest **lcf.id** file and the unique value that Tivoli Management Framework generated for that endpoint. If that value is found, it is used as the computer system ID and is written to the persistent location.
3. If no computer system ID is found and the scan was *not* initiated using the Web Interface, it determines whether the file **lcf.id** exists in the directory **LCFROOT/dat/number**, where *number* is a value based on the number of installations. If this file exists, the value in this file is used for the computer system ID but is not stored in the persistent location.
4. If Tivoli Management Framework has not generated a value for the endpoint, Tivoli Configuration Manager checks to see if the system is SMBIOS compliant. On SMBIOS-compliant systems, a computer system ID is generated using the motherboard serial number and the system serial number and is stored in the persistent location.
5. If the system is not SMBIOS compliant, Tivoli Configuration Manager generates a unique 28-character value for the system and uses that value as the computer system ID; this value is stored in the persistent location.

For UNIX endpoints, Tivoli Configuration Manager generates the computer system ID using the following algorithm:

1. It checks the persistent storage location on the endpoint for an existing computer system ID. If a computer system ID is found, that value is used.
2. If no computer system ID is found and the scan was not initiated using the Web Interface, it determines whether the file **lcf.id** exists in the directory **LCFROOT/dat/number**, where *number* is a value based on the number of installations. If this file exists, the value in this file is used for the computer system ID but is not stored in the persistent location.
3. If the file **lcf.id** was not found or if no computer system ID is found and the scan was initiated using the Web Interface, a unique value is generated for the endpoint using the same method that Tivoli Management Framework uses. This value is used as the computer system ID.

Note: If this value cannot be created for PC systems running Linux, Tivoli Configuration Manager checks to see if the system is SMBIOS compliant. On SMBIOS-compliant systems, a computer system ID is generated using the motherboard serial number and the system serial number.

4. If Tivoli Configuration Manager cannot generate a Tivoli Management Framework or SMBIOS value as described in step 3, it generates a unique 32-character computer system ID for the system.

|
| **Note:** After upgrading the endpoint to Tivoli Management Framework, Version
| 4.3.1, Inventory uses the GUID (Globally Unique Identifier) instead of the lcf
| ID. During endpoint inventory scans, only one entry for each endpoint is
| added or updated in the configuration repository.

For OS/2, NetWare, and OS/400 endpoints, the unique value that Tivoli Management Framework generates for each endpoint is used as the computer system ID. This computer system ID is not stored persistently on the endpoint. In other words, the computer system ID for these systems is handled using the same method that Inventory and Software Distribution used before Tivoli Configuration Manager, Version 4.3.1.

Chapter 2. Working with SCS

Scalable Collection Service (SCS) uses a hierarchy of collectors to transfer data from scan targets to the inventory data handler. The data is then passed through one or more RDBMS interface module (RIM) objects to the configuration repository. You should optimize the SCS components to produce an efficient flow of data and prevent errors during data transfer.

You configure SCS using command line interface (CLI) commands. This chapter provides a brief overview of the commands you use to configure and use SCS. For detailed information about these commands, see Appendix B, “Commands,” on page 103.

For help with troubleshooting SCS, see Appendix F, “Troubleshooting,” on page 281.

This chapter includes the following:

- Information about halting, starting, resetting, and configuring collectors, and modifying the collector hierarchy. (See “Using collectors.”)
- Information about configuring the inventory data handler to write data to the RIM object and manage scan status information and moving the inventory data handler. (See “Using the Inventory data handler” on page 16.)
- Information about moving the inventory callback object. (See “Moving the Inventory callback object” on page 17.)
- Information about RIM objects. (See “Creating and configuring RIM objects for the inventory data handler” on page 17.)
- The procedure to schedule the transfer of collected scan data through the collection hierarchy. (See “Scheduling collections” on page 20.)
- An explanation of how you can control the way SCS sends data through your network. (See “Controlling the flow of network data” on page 21.)
- Procedures you can use to view information about collectors, collection tables of contents (CTOCs), queues, the inventory data handler, scans, and inventory profiles, as well as information about viewing data in the configuration repository. (See “Obtaining information about SCS” on page 22.)
- An explanation of how you can configure Inventory to gather and return status information about scans. (See “Configuring inventory status information” on page 24.)

Using collectors

You use the **wcollect** command to start, stop, and configure collectors.

Note: When using the **wcollect** command, you must specify the collector on which you want to perform the action. The collector name is always the name of the managed node or gateway on which the collector is installed.

For more information about the **wcollect** command, see “wcollect” on page 110.

Stopping and starting a collector

You must stop and restart a collector to perform certain tasks with SCS. For example, you must stop and restart a collector after you reconfigure it for the changes to take effect. Also, you must stop a collector if you create a new run-time directory, so that you can move data from the old directory to the new directory. You might also need to stop and restart a collector during troubleshooting.

SCS provides two options for stopping a collector using the **wcollect** command and the **-h** option:

graceful

Bringing the collector to a graceful halt stops the collector after it completes all active collections.

immediate

Bringing the collector to an immediate halt stops the collector without waiting for active collections to finish processing. When you restart the collector, any collections that were active when the collector was halted are automatically restarted, so no data is lost.

Once you have stopped a collector using the **wcollect** command and the **-h** option, you must use the **wcollect** command and the **-s** option to start the collector again.

Configuring a collector

After you install Inventory, you should configure collectors to ensure that the run-time directory and depot are the correct size, logging options are set to your needs, and threads are set to optimize communication between collectors. You might also need to reconfigure collectors after making changes to the collector hierarchy.

Configuring the run-time directory and depot

The run-time directory contains the depot and run-time files (*.dat and *.log). By default, the SCS run-time directory is \$DBDIR/mcollect on each collector. The depot is located within the run-time directory at \$DBDIR/mcollect/depot.

You can move the run-time directory and depot if space is limited in the \$DBDIR directory. For example, you might choose to move the depot to a larger file system if you scan very large numbers of endpoints or if the depot shares disk space with the object dispatcher, which also writes data to the \$DBDIR directory.

To move the run-time directory and depot of a collector, perform the following steps:

1. Specify a new location for the run-time directory using the **wcollect** command and the **-l** option.
2. Stop the collector.
3. Move the depot and all other data from the old run-time directory to the new directory.

The run-time directory must reside on a stable disk with a large amount of free space to ensure persistent storage of collections. This directory should not be a temporary directory. You must also provide read-write privileges for the **tmersrvd** or **nobody** account (the Tivoli unprivileged account) in this directory.

4. Restart the collector.

You can change the size of the depot using the **wcollect** command and the **-z** option. By default, the depot size is 40 megabytes (MB), but you can make it as

large as 1 gigabyte (GB). Note that you cannot make it smaller than its current size; you can only make it larger. For example, you can set the depot to 50 MB and later increase that to 55 MB, but you cannot decrease the size to 45 MB.

You can control the size of data units that are transferred between collectors. These units of data are referred to as *transmission chunks*. You set the transmission chunk size with the **wcollect** command and the **-c** option. The default size is 1 MB.

If a collector attempts to collect scan data but the depot is too small to hold the data, the collection for the scanned target fails. For more information, see “Troubleshooting procedures” on page 292.

Configuring the collector log file

The `mcollect.log` file logs the activity of a collector as data flows through it. You control the amount of information that is logged in this file by setting a value with the **wcollect** command and the **-d** option. By default, only fatal errors are logged. You specify the maximum size of this file using the **wcollect** command and the **-g** option. By default, the maximum size is 1 MB.

Configuring threads

Threads control each successful communication attempt between two collectors. Collectors run as multithreaded daemon processes. The collector uses threads primarily to maintain multiple data streams going into and out of the depots. Several configuration options of the **wcollect** command are related to threads between collectors. You can set the maximum number of input threads with the **-t** option and output threads with the **-o** option. The default number for both types of threads is 5.

An input thread becomes idle when no requests in its queues are ready to process. You can specify the amount of time that an input thread should wait before it shuts down by using the **-i** option. The default idle time is 60 seconds. To specify the amount of time that a thread waits if system or network resources are temporarily unavailable, use the **-p** option. The default value is 5 seconds.

Retries are related to threads. A *retry* is an attempt by a collector to process a collection request from another collector following a failure. Data transfer between collectors, or between collectors and targets, can fail for a variety of reasons. For example, if a thread cannot negotiate a network connection to another collector or endpoint, the thread has failed. By configuring retries and a retry delay time for collector threads, you enable SCS to recover from temporary error conditions:

- Use the **-m** option to set the maximum number of times that a collector attempts to process a collection request from a downstream collector following a failure. By default, this value is 10.
- Use the **-e** option to set the retry delay time. This is the time in seconds that a collector waits before trying again to establish communication with another collector. The default value is 1 second.

Configuring checkpoint mode

To manage the SCS internal queues and improve the overall SCS performance, you can choose how you want to monitor and handle the changes that occur in the queues.

You can specify different checkpoint management policies using the **wcollect** command and the **-b** option.

The following options are available:

none The checkpoint of the queues is totally disabled. If for any reason the SCS process dies or is killed all the information that it is processing might be lost. For this reason this configuration is not recommended.

full Each change to the queues is immediately check-pointed to the file system rewriting the entire queue. The entire queue is dumped to the file system when a new checkpoint is requested.

incremental

Each change to the queues is immediately check-pointed to the file system but in this case only the differences are written to the file system reducing in this way the time required to perform the operation. When a checkpoint is requested only the differences are written to the file system instead of the entire queue.

Modifying the collector hierarchy

Because SCS shares the multiplexed distribution (MDist 2) repeater hierarchy (although data flows in the opposite direction), you must configure the hierarchy so that it is optimized for both services.

To modify the SCS hierarchy, you use the **wrpt** command to set up your repeater topology as needed. After you make changes to the hierarchy, you must then use the **wcollect** command and the **-r** option on each collector in the modified part of the hierarchy. This command removes the locally cached collection route for a collector. After you run this command, the collector downloads new routing information from the collection manager when the collector processes the next collection request.

In interconnected regions, if the source and the target of a distribution belong to different regions, the spoke gateway collectors send their scan results back to the hub collector, known as Wan entry point collector.

See the *Tivoli Management Framework: Reference Manual* for more information about the **wrpt** command.

In interconnected regions, if the source and the target of a distribution belong to different regions, the spoke gateway collectors send their scan results back to the hub collector, known as Wan entry point collector.

Using the Inventory data handler

The inventory data handler can be considered the last collector in the SCS collector hierarchy. The inventory data handler receives data from collectors and sends the data to the configuration repository.

Configuring the Inventory data handler

You use the **wsetinvdh** command to specify how the inventory data handler writes data to a RIM object. You use the **wcollect** command to configure output threads for the inventory data handler.

You use the **wcollect** command and the **-o** option to specify the number of output threads for the inventory data handler to write to RIM objects. The default number of output threads for the inventory data handler is 1. It is recommended that this value match the total number of RDBMS connections set for all RIM objects used

by the inventory data handler. For example, if the inventory data handler uses two RIM objects to connect to the RDBMS, and each RIM object has five RDBMS connections, set the output threads for the inventory data handler to 10. You must also make sure that you have a database license available for each RDBMS connection. For information about setting RDBMS connections for RIM objects, see “Configuring RIM objects” on page 19.

Use the **wsetinvdh** command and **-r** option to specify the number of times the inventory data handler tries to write data to a RIM object. After the specified number of retries is reached, the inventory data handler sends a failure notification. The default value for this option is 5 retries.

Related to the **wsetinvdh** command and **-r** option, the **wsetinvdh** command and **-t** option enable you to specify a value from which to calculate the timeout period between retries.

See “Configuring inventory status information” on page 24 for information about configuring the status collection options of the inventory data handler.

Moving the Inventory data handler

The inventory data handler is created when you install Inventory. It is recommended that you do *not* install the inventory data handler on the same managed node as the Tivoli management region server (Tivoli server).

If you need to move the inventory data handler to a different managed node, you can do so using the **wmvinvdh** command. Inventory Server must be installed on the managed node to which you move the inventory data handler.

Use the **wmvinvdh** command and **-t** option to specify a timeout value for the command. This value specifies the time, in seconds, after which the command will time out if it cannot contact the managed node to which you want to move the inventory data handler. The default value is 120 seconds.

For instructions on using the **wmvinvdh** command, see “wmvinvdh” on page 181.

Moving the Inventory callback object

When you install Inventory, the inventory callback object is created automatically. To move the inventory callback object, use the **wmvinvcb** command. You can move the inventory callback object to any managed node in your Tivoli region on which Inventory Server is installed.

Use the **wmvinvcb** command and **-t** option to specify a timeout value for the command. This value specifies the time, in seconds, after which the command will time out if it cannot contact the managed node to which you want to move the inventory callback object. The default value is 120 seconds.

For instructions on using this command, see “wmvinvcb” on page 180.

Creating and configuring RIM objects for the inventory data handler

The inventory data handler uses a RIM object to write data to the configuration repository. When you install Inventory, the invdh_1 RIM object is created for this purpose. However, you can create one or more additional RIM objects for the inventory data handler to use.

You create additional RIM objects for the following reasons:

- To increase the number of connections to the RDBMS. The maximum number of connections per RIM object is 16.
- To distribute the work performed by RIM objects across multiple systems. For example, you can create RIM objects on separate managed nodes so that the processing required by the RIM objects is divided between the two systems.

Creating RIM objects

You create RIM objects using the **wcrtrim** command. To create RIM objects for the inventory data handler, you must specify an application label of **invdh** for the RIM object using the **-a** option.

You must also configure the maximum number of connections that the RIM object is allowed to establish with the configuration repository using the **-m** option. It is recommended that the total number of RDBMS connections set for all RIM objects used by the inventory data handler match the number of output threads set for the inventory data handler. For example, if the inventory data handler has 10 output threads and uses two RIM objects to connect to the RDBMS, you could configure each RIM object to have five RDBMS connections. You must also make sure that you have a database license available for each RDBMS connection.

For example, to create a RIM object named **invdh_2** with an application label of **invdh** and a maximum of five RDBMS connections, run the following command:

```
wcrtrim -v Informix -h bernard -d inventory \  
-H /data/rdbms/informix -u informix -s az914shm \  
-a invdh -m 5 invdh_2
```

where:

-v Informix

Specifies the name of the database vendor that the RIM object represents.

-h bernard

Specifies the name of the managed node on which the RIM object resides.

-d inventory

Specifies the name (database ID) of the database to which the RIM object connects.

-H /data/rdbms/informix

Specifies the path to the database home directory.

-u informix

Specifies the name of the database user that the RIM object uses.

-s az914shm

Specifies the server ID for the database.

-a invdh

Specifies that the application label for the RIM object is **invdh**. For RIM objects that the inventory data handler uses to connect to the RDBMS, you must set this value to **invdh**.

-m 5

Specifies the maximum number of connections that the RIM object has to the database. Set this option to a positive value from 1 to 16. The default value is 16. It is recommended that the total number of RDBMS connections set for all RIM objects used by the inventory data handler match the number of output threads set for the inventory data handler.

invdh_2

Specifies the name of the RIM object.

Configuring RIM objects

To configure RIM objects, you determine the current settings for the RIM objects and then change those settings. You can view and set most options for RIM objects using the **wgetrim** and **wsetrim** commands. For more information about these commands, see the *Tivoli Management Framework: Reference Manual*, GC32-0806.

For RIM objects that the inventory data handler uses, you must set the application label and maximum number of RDBMS connections. You cannot view these values using the **wgetrim** command; you must use the **idlcall** command. However, you can set these values using the **wsetrim** command. The following examples demonstrate how to view and set these values.

To view the application label of a RIM object, first determine the object ID (OID) of the RIM object by running the following command:

```
wlookup -r RIM_object_name
```

where *RIM_object_name* is the name of the RIM object, for example invdh_1.

The output from this command will be similar to the following example:

```
invdh_1 1228633823.2.20#RIM::RDBMS_Interface#
```

In this example, 1228633823.2.20#RIM::RDBMS_Interface# is the OID for the RIM object invdh_1. To view the application label for this RIM object, run the following command:

```
idlcall OID _get_application_type
```

where *OID* is the OID for the RIM object, in this example 1228633823.2.20#RIM::RDBMS_Interface#.

RIM objects that the inventory data handler uses to connect to the RDBMS must have an application label of invdh. Other types of RIM objects do not need an application label. To set this value, run the following command:

```
wsetrim -a invdh RIM_object_name
```

To view the maximum number of RDBMS connections set for a RIM object, run the following command:

```
idlcall OID _get_max_conn
```

where *OID* is the OID for the RIM object.

To set this value, run the following command:

```
wsetrim -m n RIM_object_name
```

where *n* is a value from 1 to 16. By default, Inventory RIM objects are created with a value of 1. It is recommended that the total number of RDBMS connections set for all RIM objects used by the inventory data handler match the number of output threads set for the inventory data handler.

Moving RIM objects

To move a RIM object, use the **wmvrim** command. For more information about this command, see the *Tivoli Management Framework: Reference Manual*.

For more information about the RIM objects installed with Inventory, see *Planning and Installation Guide*.

Scheduling collections

To schedule collections, you specify that a collector should not collect data from certain downstream collectors at a particular time. Scheduling collections can also be referred to as scheduling *offlinks*. Offlinks are links to collectors from which you do not want to collect data.

Note: You can specify offlinks only between collectors. You cannot specify offlinks between collectors and endpoints.

You can control the time when collections are transferred through the collection hierarchy by specifying the times when the links between collectors are turned on and off. For example, a bank might have a network that contains teller terminals as well as several other computers. Because the teller terminals must have quick access to bank records on the network server during business hours, the bank can choose to prohibit collections from the teller terminals during these hours. Figure 3 illustrates this example.

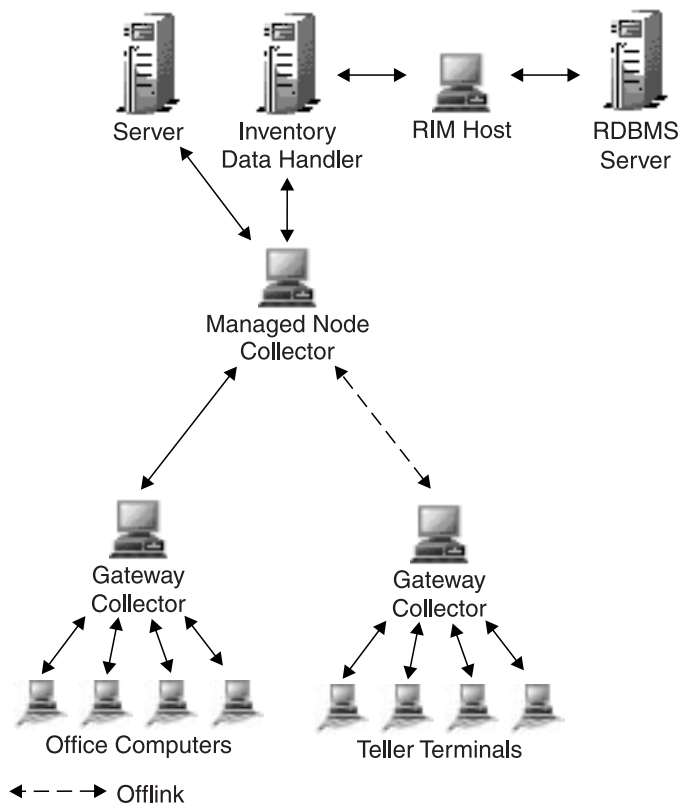


Figure 3. Using Offlinks

When you turn off a link between collectors, the CTOC for a pending collection moves to the deferred queue of the downstream collector. When the link between the collectors is restored, the CTOC in the deferred queue is processed.

You schedule collections by defining tasks and jobs in Tivoli Management Framework and then scheduling the jobs with the Tivoli scheduler. To do this, you

must be familiar with the concepts described in the chapter about task libraries in the *Tivoli Management Framework: User's Guide*.

The following is an overview of the procedure to schedule collections:

1. Create a task that turns off links to a collector, then halts and restarts the collector so the changes take effect:

```
wcollect -x offlinks_range_to_prohibit_collection collector
wcollect -h immediate
wcollect -s
```

where:

offlinks_range_to_prohibit_collection

Specifies the range of offlinks. Offlinks are the object dispatcher numbers of collectors from which the specified collector must not collect data.

collector

Specifies the collector to which links must be turned off.

See “wcollect” on page 110 for more information about the **wcollect** command and **-x** option.

2. Create a task that turns on the links to all systems for which links were previously turned off, then halts and restarts the collector so the changes take effect:

```
wcollect -x "" collector
wcollect -h immediate
wcollect -s
```

3. Repeat steps 1 and 2 on each collector for which you want to schedule collections.
4. Create jobs to run these tasks.
5. Use the Tivoli scheduler to control when and how often to run these jobs.

You can also schedule collections by creating a job that halts and restarts collectors using the **wcollect** command and the **-h** and **-s** options. Alternatively, you can schedule a job that shuts down the output queue or input queue of a collector using the **wcollect** command by setting the **-o** or **-t** options for that collector to 0.

Note: When you reconfigure a collector, the changes do not take effect until you restart the collector.

Controlling the flow of network data

SCS provides features to control the flow of SCS data across your network. These features are helpful if you have slow network links or if you want to specify when SCS data crosses your network. SCS provides the following mechanisms to control data flow:

Offlinks

Offlinks are the primary way to regulate SCS traffic across your network. You can use offlinks to enable and disable SCS traffic between collectors at specified times. To use offlinks, you install a collector on either side of the network that requires flow control, then schedule offlinks using the Tivoli scheduler. For more information about scheduling offlinks, see “Scheduling collections” on page 20 and “wcollect” on page 110.

Transmission chunk size

You use the **wcollect** command and **-c** option to configure the size of

transmission chunks. This option enables you to control the size of the SCS data packet that crosses the interobject message (IOM) connection. The default transmission chunk size is 1 MB. If you specify a smaller size, the data is sent in smaller fragments. Decreasing transmission chunk size might be beneficial for slow links because the link is not congested with large block transmissions of SCS data. You configure transmission chunk size on the downstream collector. For more information about configuring transmission chunk size, see “Configuring a collector” on page 14 and “wcollect” on page 110.

Note: Offlinks and transmission chunks affect data transmission between collectors, or between a collector and the inventory data handler. These mechanisms do not affect transmissions between an endpoint and the gateway collector.

Input Threads

Collectors use input threads to open an IOM session to a downstream collector for retrieving data. You can specify the maximum number of input threads for a collector using the **wcollect -t** command. You can increase the maximum number of input threads to allow more concurrent SCS IOM sessions, or decrease the number to reduce SCS IOM traffic coming into the collector. For more information about configuring the input threads of a collector, see “Configuring threads” on page 15 and “wcollect” on page 110.

Note: By configuring the input threads of a gateway collector, you can help to reduce the load on the gateway by limiting the number of SCS sessions that are simultaneously active to endpoints from that gateway. This also helps reduce SCS network traffic coming into the gateway.

Entries in the output queue

You can configure the maximum number of entries that can be added to the data handler input and output queue using the **wcollect -n** command. This setting keeps the size of the checkpointGL_iqfile.dat file small, avoiding the long delays updating the file when it was allowed to become too large. This setting has no effect on interconnected Tivoli regions.

Monitor for the output queue

You can monitor the amount of traffic for the output queue on the gateway and collector comparing the specified value to the number of accumulating retries on the CTOCs in the output queue of the collector. If the comparison threshold is exceeded, it automatically reduces the number of output threads on that collector to 1. Enable this monitor using the **wcollect -w** command. This setting has no effect on interconnected Tivoli regions.

Obtaining information about SCS

Using SCS commands, you can view the following information:

- The configuration and status of a collector
- A description of the CTOCs on a collector
- The contents of the queues of a collector
- The configuration of the inventory data handler
- Details about current inventory scans
- Details about SCS and Inventory status collector options of an inventory profile

Viewing collector settings

Use the **wcollect** command to get information about the current configuration of a collector. You can view the level of debugging information being written to the collector log file, maximum size of the log file, location of the run-time directory, depot size, size of data transmission chunks, thread settings, collection retry settings, offlinks, and other information.

To view configuration information about a collector, enter the following command:

```
wcollect collector
```

where *collector* is of the form **@ManagedNode:collector_name**, **@Gateway:collector_name**, **@InvDataHandler:inv_data_handler**, or **@InvDataHandler:tlm_data_handler**.

Viewing collector status

Use the **wcstat** command to get status information about a collector. You can view this information for assistance in troubleshooting collectors.

To view status information about a collector, enter the following command:

```
wcstat collector
```

where *collector* is of the form **@ManagedNode:collector_name**, **@Gateway:collector_name**, **@InvDataHandler:inv_data_handler**, or **@InvDataHandler:tlm_data_handler**.

Viewing information about queues and CTOCs

Use the **wcstat** command to get information about one or more queues or CTOCs of a collector.

To view information about the queues of a collector, enter the following command:

```
wcstat -q [ioecd] collector
```

where [**ioecd**] specifies the queue or queues for which you want information (input, output, error, completed, or deferred, or any combination), and *collector* is of the form **@ManagedNode:collector_name**, **@Gateway:collector_name**, **@InvDataHandler:inv_data_handler**, or **@InvDataHandler:tlm_data_handler**.

You can view the following information about queues: CTOC ID, priority of queued data, collector status, source and destination information, the method to be called on the destination object, scan ID, collection status, and the number of times that this collector has retried submitting requests to an upstream collector.

To view information about a CTOC, enter the following command:

```
wcstat -v ctoc_id collector
```

where *ctoc_id* specifies the CTOC for which you want status information, and *collector* is of the form **@ManagedNode:collector_name** or **@Gateway:collector_name**.

Information about a CTOC includes the CTOC ID, priority of the scanned data, collector status, the source and destination of the scan data, the method to be called on the destination object, client properties, collection status, and number of retries.

Viewing inventory data handler settings

The **wgetinvdh** command returns configuration information about the inventory data handler, including scan completion notification and the number of retries as well as the timeout period for attempts to write data to a RIM object. The **-a** option causes the **wgetinvdh** command to return all available information about the inventory data handler. You can use several other options to return specific information. See “**wgetinvdh**” on page 131 for more information.

Viewing information about inventory scans

The **wgetscanstat** command returns details about inventory scans. If you run the **wgetscanstat** command without any options, it returns a list of scan IDs of currently pending inventory scans along with the names of the inventory profiles that initiated those scans. You can also use options that provide information about the success, failure, or pending status of specific scans or all current scans. See “**wgetscanstat**” on page 154 for more information.

Viewing inventory profile settings

The **wgetinvglobal** command returns information about status notification settings, distribution settings, and data settings for an inventory profile. If you run this command with no options, it returns all available information about these settings for the inventory profile. You can use other options that provide specific information. See “**wgetinvglobal**” on page 133 for more information.

Data stored in the configuration repository

To view inventory data stored in the configuration repository, you can use any of the predefined queries provided with Inventory or create your own queries. For more information, see Chapter 6, “Querying inventory information,” on page 87.

Configuring inventory status information

You can configure Inventory to return status information about scans. *Status information* is a record of whether the scan has completed on each target in a profile distribution.

You use the **wsetinvdh** and **wsetinvglobal** commands to configure status collector options, which specify how status information is collected, stored, and distributed:

- Use the **wsetinvdh** command and the **-s** option to specify whether the inventory data handler stores status information. If a system failure occurs, status information is automatically restored when the inventory data handler is back online. By default, status information is stored.
- Use the **wsetinvdh** command and the **-d** option to specify the directory where the inventory data handler stores status information. The default location for this directory is `$DBDIR/inventory/stat_dir` on the managed node where the inventory data handler resides.
- Use the **wsetinvglobal** command and the **-l** option to specify the location to which notifications are sent. You can choose to send notification to the Inventory notice group, to the log file, the IBM Tivoli Enterprise Console® console, or any combination of those options. You can also choose not to send notification.
- Use the **wsetinvglobal** command and the **-f** option to specify the name of the log file where status information is stored. The **wsetinvglobal** command and the **-h** option specify the host upon which the log file is stored. These options apply only if the **wsetinvglobal** command and the **-l** option specify to send notification to a log file.

- Use the **wsetinvdh** command and the **-n** option to specify the interval, in minutes, at which status information is sent. For example, if you set this value to 10, every 10 minutes the inventory data handler sends all completed scan notifications in a single notice called a *bundle*. The inventory data handler sends the bundles to the Inventory notice group, a log file, the Tivoli Enterprise Console console, or a combination of those options, depending on the setting of the **wsetinvglobal** command and the **-l** option. The default value for the **wsetinvdh** command and the **-n** option is 10 minutes.
- Use the **wsetinvglobal** command and the **-t** option to set the circumstances under which a notice is sent for the specified profile. You can choose to send notification when scans complete successfully, when scans fail, or for all targets.
- Use the **wsetinvdh** command and the **-u** option to specify whether the inventory data handler sends a notice to the Inventory notice group when an unsolicited update of scan data occurs. An unsolicited update is an update that is not initiated by distributing a scan to an endpoint remotely from another system. Examples include an endpoint-initiated scan using the **wepscan** command or an update of configuration repository data by an application or component other than Inventory.
- Use the **wsetinvglobal** command and the **-n** option to specify when notification is sent when a scan completes on each target.
- Use the **wsetinvdh** command and the **-q** option to specify the maximum number of targets in a bundle. For example, if you set this value to 5, each time scans complete on five targets, the inventory data handler bundles the status information and sends it. The default value is 10 targets.

You can use the **wgetscanstat** command to return status information about current inventory scans. You can use the **wgetinvglobal** command to return information about status collector options for a specified profile.

Chapter 3. Working with Inventory profiles

This chapter describes the steps necessary to create, customize, clone, and delete inventory profiles. When setting up your inventory profiles, consider what information you need to gather, how you will use the information, and the constraints of your Tivoli environment. Once you have determined how you can best use the information gathered by Inventory, you are ready to configure inventory profiles.

See Chapter 4, “Distributing Inventory profiles,” on page 51 for more information about distributing inventory profiles and how Inventory populates the configuration repository. See Chapter 5, “Collecting custom information with Inventory,” on page 59 for information about customizing Inventory to gather information that is not provided by default.

This chapter includes the following information:

- An explanation of how to use inventory profiles. (See “Using inventory profiles.”)
- Information about creating an inventory profile. (See “Creating an inventory profile” on page 28.)
- Information about setting the scanning instructions and other options in an inventory profile. (See “Customizing an inventory profile” on page 32.)
- Steps for creating an inventory profile by cloning an existing inventory profile. (See “Cloning an inventory profile” on page 47.)
- Steps for deleting an inventory profile. (See “Deleting an inventory profile” on page 48.)
- Steps for renaming an inventory profile. (See “Renaming an inventory profile” on page 49.)

Using inventory profiles

You can use an inventory profile to distribute a configuration file, scan a target system, or run a custom script or executable.

Inventory provides several options for customizing inventory profiles. For example, you can create an inventory profile that collects different sets of data from PC and UNIX systems. This flexibility enables you to collect only the data you need from each platform. You can also specify the hardware data that you want to collect, and you can configure the files and directories to include or exclude during software scans.

Note: Inventory does not scan NFS-mounted systems.

You can customize a profile to run a script on target systems before or after the system is scanned. Scripts can be used to run optional scanning programs. For example, you could create and then run a script that gathers user information from a text file and then converts the data to a Management Information Format (MIF) file. You can also update the BIOS dictionary file provided with Inventory through a script that runs on your profiles. See Chapter 5, “Collecting custom information with Inventory,” on page 59 for more information about scripts specific to the BIOS.

You can control whether the inventory profile runs a scan, sends the data to the configuration repository, or both. Using this option, you can create one inventory profile that scans the target and stores the scan data locally and another inventory profile that sends the data to the configuration repository.

Instead of saving all data from a scan, you can choose to save only the differences between the current scan and the previous scan. When you do this, Inventory compares the current MIF files to the MIF files generated from the previous scan and sends only the differences to the configuration repository. This reduces the amount of data returned through the network.

Creating an inventory profile

You create an inventory profile in a profile manager. Use profile managers to organize your profiles into logical groups. You can either create an inventory profile in an existing profile manager or create a new profile manager. For example, if you want to use profile managers to organize Tivoli profiles according to function, create a new profile manager named Inventory Profiles for all the inventory profiles in a policy region. On the other hand, if you have profile managers that organize targets according to operating system, you could create an inventory profile in each profile manager. Each profile could include a script or executable and scan instructions that are specific to the operating system of the systems in the profile manager.

You can create a *database profile manager* or a *dataless profile manager*. A database profile manager can distribute to any profile manager (database or dataless) and to resource groups—but not to endpoints. A dataless profile manager can distribute to endpoints, but not to other profile managers. Therefore, you must create dataless inventory profiles for endpoint subscribers and database inventory profiles for profile manager or resource group subscribers. Although you cannot subscribe endpoints to a database profile manager, you can subscribe endpoints to a dataless profile manager and make that profile manager a subscriber of a database profile manager.

Make sure that the subscribers of the profile manager are appropriate targets for the profile. Valid targets for an inventory profile are endpoints, profile managers, and resource groups. If the profile manager has managed nodes as subscribers, Inventory ignores them.

For information about profile managers, see the *Tivoli Management Framework: User's Guide*.

When you name an inventory profile, there are several things to consider. First, if you are planning to have a large number of inventory profiles, choose a name for each profile that reflects its function. For example, a profile that is customized to scan only hardware information could be named `Hardware_Scan`.

The following table provides the authorization roles required to create an inventory profile:

Operation	Context	Role
Create an inventory profile	Profile manager	senior or super

You can create a profile from either the Tivoli desktop or the command line.

Desktop

Complete the following steps to create an inventory profile from the Tivoli desktop:

1. In a policy region, add the **InventoryConfig** and **ProfileManager** resources to the list of managed resources.

For instructions about adding resource types and creating profile managers in a policy region, see the *Tivoli Management Framework: User's Guide*.

2. Create or select a profile manager in which the inventory profile will reside.
3. Subscribe targets to which you want to distribute profiles from this profile manager.

For more information about profile managers and subscribers, see the *Tivoli Management Framework: User's Guide*.

4. From the policy region, double-click the profile manager in which you want to create the inventory profile. The Profile Manager window is displayed.



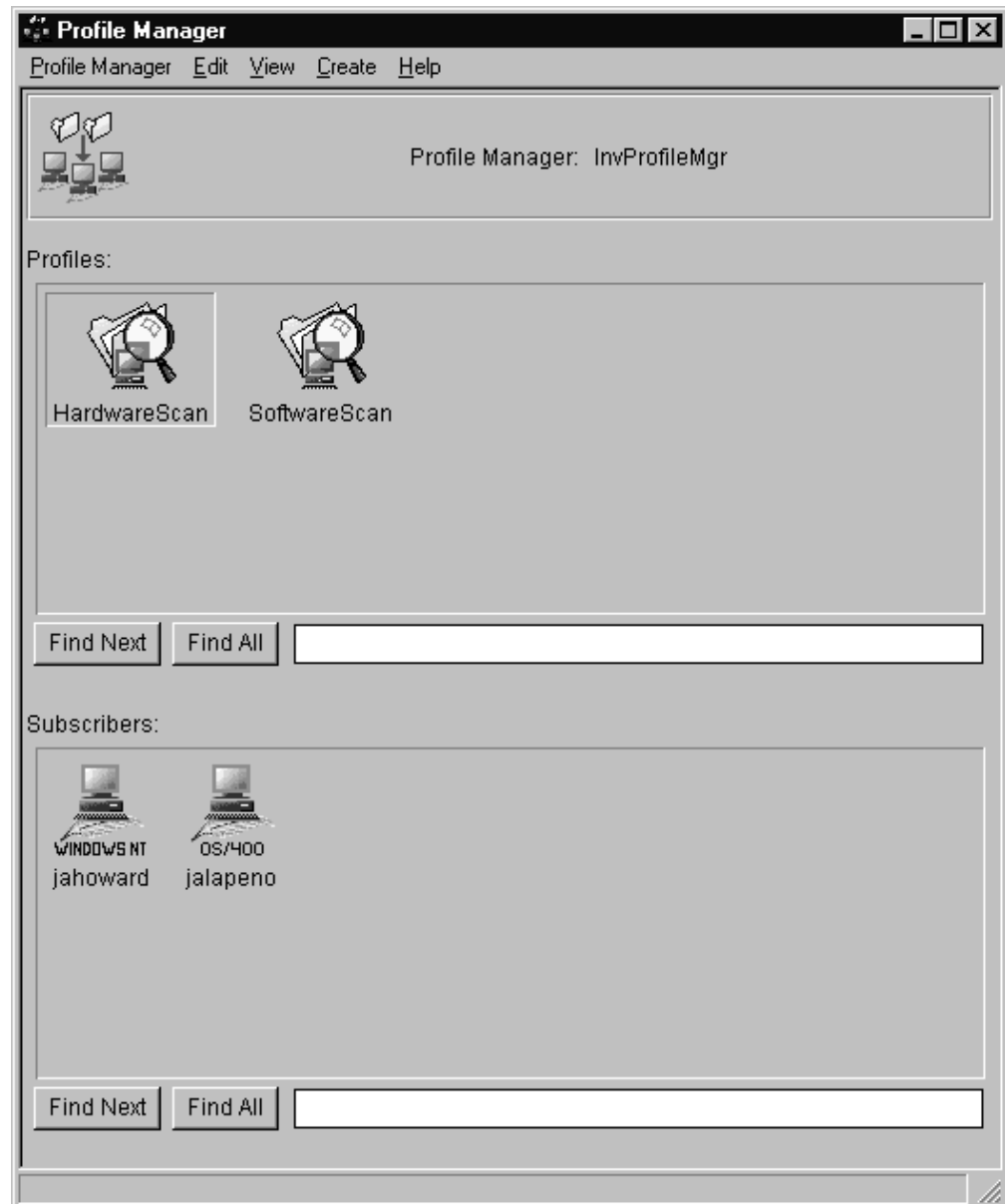
5. In the Profile Manager window, select **Create->Profile**. The **Create Profile** dialog is displayed.



6. In the **Name/Icon Label** text box, enter a unique name for the profile.
7. From the **Type** scrolling list, select **InventoryConfig**.

Note: If the **InventoryConfig** resource type is not available, you must add this resource type as a managed resource for your policy region. For more information on this procedure, see the *Tivoli Management Framework: User's Guide*.

8. Click **Create & Close** to create the new profile and return to the Profile Manager window. An icon representing the new inventory profile is displayed in the Profile Manager window. The profile has the default scanning options. For instructions on customizing a profile, see "Customizing an inventory profile" on page 32.



Command line

Because inventory profiles must reside in a profile manager, you must create a policy region and profile manager in which to create the profile. To create policy regions and profile managers, see the *Tivoli Management Framework: Reference Manual* for information on the **wcrtprf** and **wcrtprfng** commands.

To use the **wcrtprf** command to create the **Enterprise** profile in the **Jamaica** profile manager, enter the following:

```
wcrtprf @ProfileManager:Jamaica InventoryConfig Enterprise
```

where:

@ProfileManager:Jamaica

Specifies **Jamaica** as the name of the profile manager in which to create the profile.

InventoryConfig

Specifies the type of profile to create.

Enterprise

Specifies **Enterprise** as the name of the new profile.

For more information about using the command line to create a profile, see the **wcrtprf** command in the *Tivoli Management Framework: Reference Manual*.

Customizing an inventory profile

You can customize profiles or sets of profiles to serve various purposes in your Tivoli environment. When you customize a profile, you specify what happens on target systems when the profile is distributed. For example, you can create an inventory profile to collect the complete hardware and software configuration on each system in your Tivoli environment. Then, you can create another profile for subsequent scans that updates the configuration repository with only new or changed data about the hardware and software on those systems.

When you customize inventory profiles, you can specify what happens when you distribute the profile, including the following:

- Whether to run a scan or just distribute a configuration file
- How to save scan data to the configuration repository (differences or current results)
- The type of scan to run, for example, hardware, software, or both
- The type of information to collect from scans of pervasive devices
- Whether to read scan results and which scan results to read (hardware, software, or custom MIF files)
- Which files to scan
- Which directories to scan
- Which patches are missing or installed
- What hardware information to collect
- What data to collect from scans of DMI service layers on PC systems
- What software information to collect: information about installed products, matching signature information, header information (on PC systems), basic file information, or a combination of these options
- Whether to apply a custom filter to scans for basic file information
- Whether to generate checksum values for scanned files
- Whether to run scripts that you specify on target systems

From the Tivoli desktop, you can customize inventory profiles using the Inventory graphical user interface (GUI). This chapter provides an overview of the Inventory GUI. For detailed information about the Inventory GUI, see the Inventory online help.

From the command line interface (CLI), you can use a number of commands to gather information about and customize inventory profiles. For a brief overview of

these commands, see “Customizing inventory profiles from the command line” on page 46. For detailed information about these commands, see Appendix B, “Commands,” on page 103.

Customizing global properties

From the Inventory GUI, you set global properties for an inventory profile using the Global Properties window. To display this window, select **Global Properties** in the left pane of the Inventory Administration window. The following window is displayed:

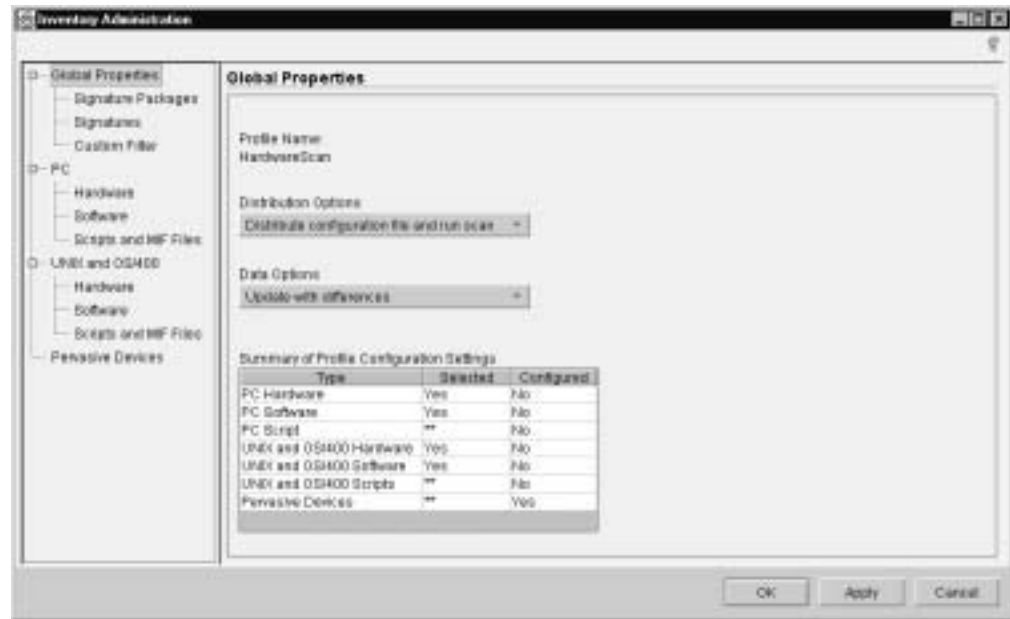


Figure 4. Global Properties Window

Global properties include the distribution and data options for a profile. These options apply globally to all scans that a profile distributes.

Note: The Global Properties options do not apply to scans of pervasive devices. For information about scanning pervasive devices, see “Customizing scans of pervasive devices” on page 45.

The global properties are as follows:

- **Distribution Options** specify what actions occur when you distribute a profile. You can either distribute the profile without running the scan or distribute the profile and run the scan.
- **Data Options** specify how the data gathered by this profile is stored in the configuration repository. You can choose one of the following options:

Update with differences

This option compares MIF files read during the current distribution with MIF files from the previous distribution and saves the differences. Only data that has been added or changed since the last scan is stored in the configuration repository. Data from previous scans that is not present in the current scan is deleted from the configuration repository. For example, if you remove a hard drive from a system and then rescan the system using the **Update with differences** scan option, the record for the

hard drive is deleted. However, if you deselect the **Storage** hardware scan option and then rescan the machine, the hard drive data is not deleted. If you do not have history tables set up, no record of changes is kept. For more information about history tracking, see the *Database Schema Reference*.

Select this option to reduce the amount of information that is sent to the configuration repository.

Note: The first time that you scan a system, all scan data is returned, even if this option is selected.

Replace with current results

This option deletes the data in the configuration repository that corresponds to the type of scan you are running (hardware scan, scan of registry or operating system for installed product information, signature scan, scan for basic file information, or scan for header file information), and then stores the data gathered by this scan in the configuration repository. For example, if you scan a system for all hardware data, and then rescan the machine for processor information only using the Replace with current results option, the scan deletes all hardware data in the configuration repository for the scanned machine and then stores only the processor information for that machine.

Select this option when you want to save a fresh image of the hardware and software on the target machines.

If you add history tables to an environment with scan data already in the configuration repository, it is recommended that you use the **Replace with current results** option for the first scan, and then use **Update with differences** for subsequent scans. For more information about history tables, see the *Database Schema Reference*.

Customizing a software scan

You customize software scans from the Inventory GUI using the Software window.

Software scan options for PC

To customize software scans of PC systems, select **PC->Software** in the left pane of the Inventory Administration window. The Software window is displayed.

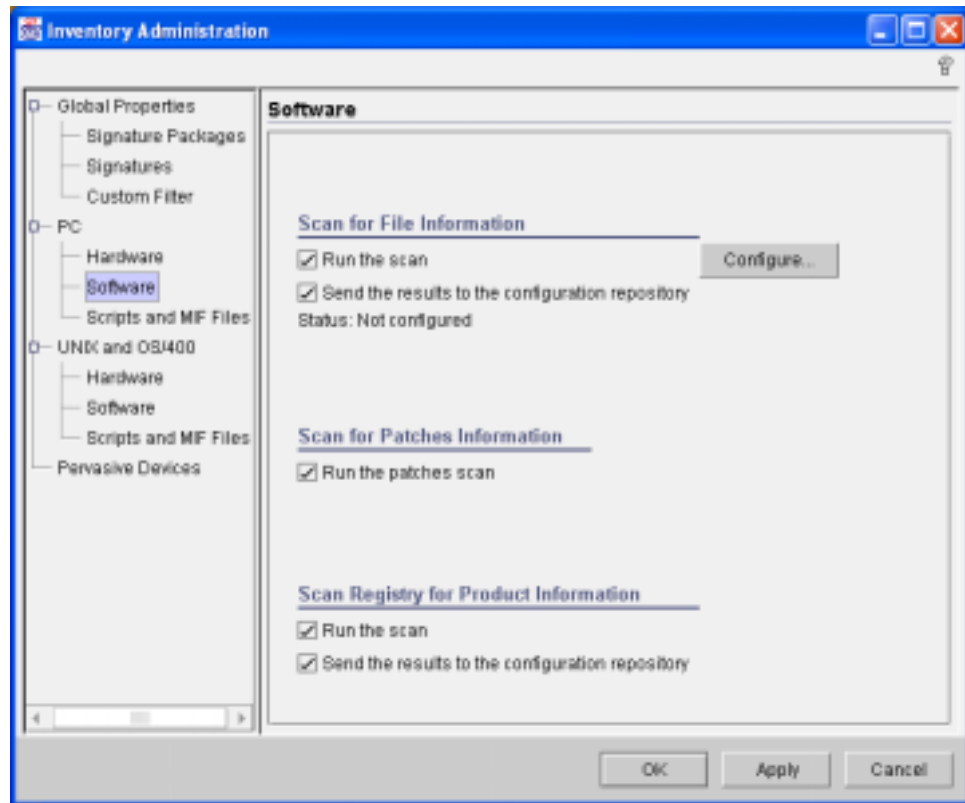


Figure 5. Software Window for PC Systems

From this window, you can enable and configure scans for file information, scans for patches, and scans of the registry for product information.

- **Scan for File Information** scans the files stored on the local hard drive or drives of an endpoint. To enable this scan, select one or both of the following options:
 - **Run the scan** scans the endpoint and then stores the data on the endpoint in a MIF file.
 - **Send the results to the configuration repository** collects the MIF file from the endpoint and sends the data to the configuration repository.
- **Scan for Patches Information** scans the patches installed on the hard drives of an endpoint for detecting the state of the patches. To enable this scan, select the following option:
 - **Run the patch scan** scans the endpoint, stores the data on the endpoint in a MIF file and then sends the data to the configuration repository.
- **Scan Registry for Product Information** scans the registry of endpoints on supported Windows platforms for information about installed products. You must use the query NATIV_SWARE_QUERY to gather data from this scan.

Note: This option has no effect on NetWare and OS/2® endpoints.

To enable this scan, select one or both of the following options:

- **Run the scan** scans the endpoint and then stores the data on the endpoint in a MIF file.
- **Send the results to the configuration repository** collects the MIF file from the endpoint and sends the data to the configuration repository.

Software scan options for UNIX and OS/400

To customize software scans of UNIX and OS/400 systems, select **UNIX and OS/400->Software**. The Software window is displayed.

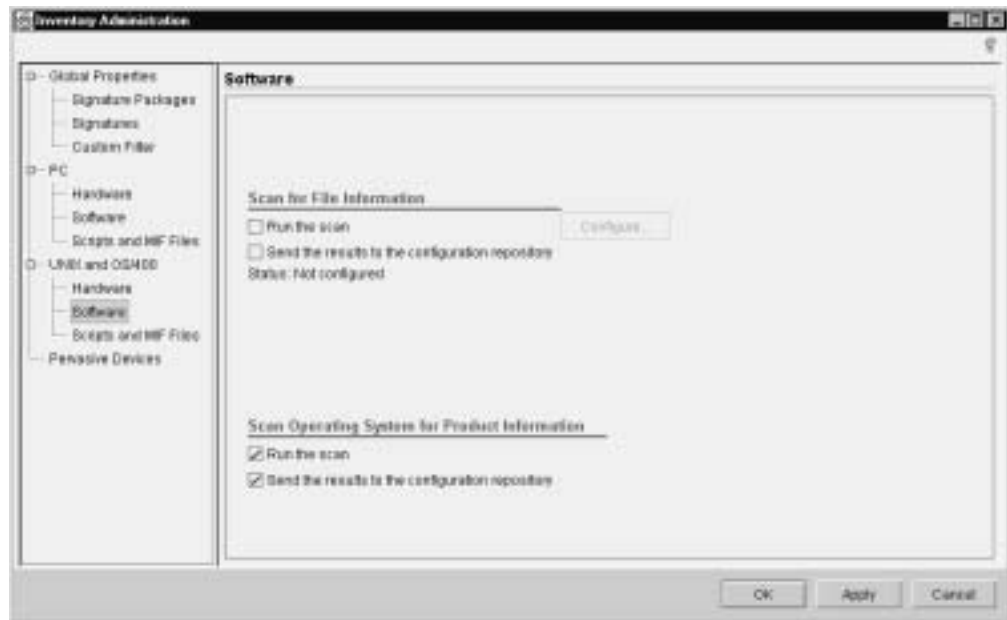


Figure 6. Software Window for UNIX and OS/400 Systems

From this window, you can enable and configure scans for file information and scans of the operating system for product information.

- **Scan for File Information** scans the files stored on the local hard drive or drives of an endpoint.

Note: This option has no effect on OS/400 systems.

- **Scan Operating System for Product Information** scans the operating system for information about installed products and patches. You must use the query NATIV_SWARE_QUERY to gather data from this scan.

To enable these scans, select one or both of the following options:

- **Run the scan** scans the endpoint and then stores the data on the endpoint in a MIF file.
- **Send the results to the configuration repository** collects the MIF file from the endpoint and sends the data to the configuration repository.

Software scan configuration options

The Software Scan Configuration window enables you to customize your scans for file information. You access the Software Scan Configuration window by clicking **Configure** from the Software window. Using this window, you can specify the type of information that the scan gathers and specify the files and directories to include or exclude during the scan.

Figure 7 shows the Software Scan Configuration window for PC systems:

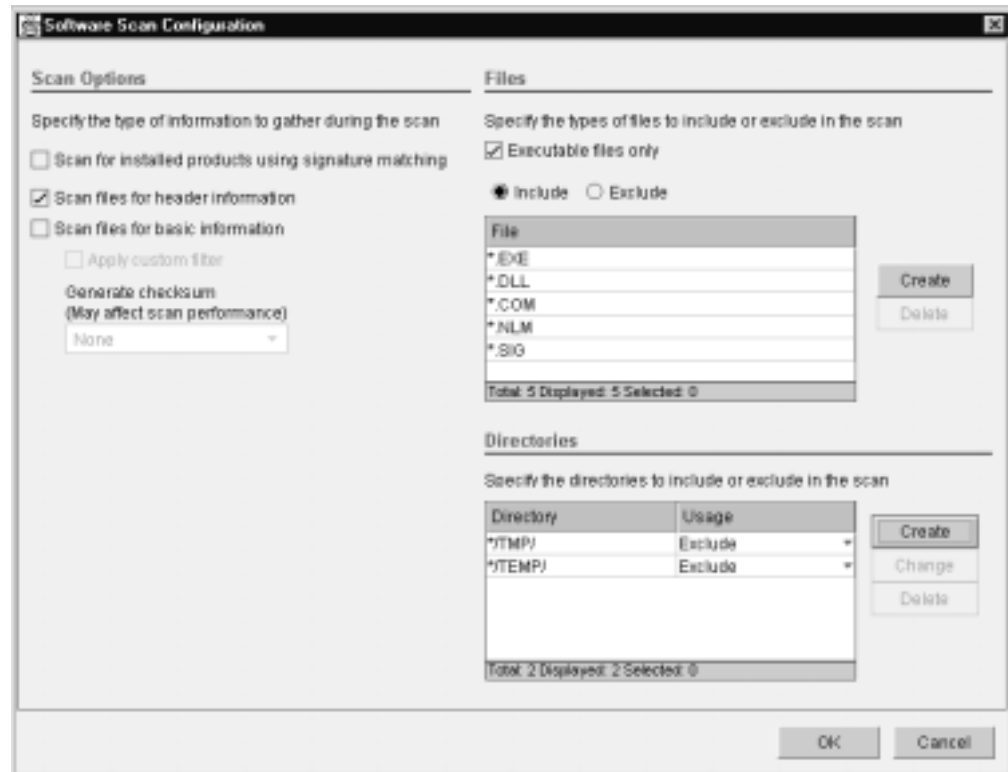


Figure 7. Software Scan Configuration Window for PC Systems

Figure 8 shows the Software Scan Configuration window for UNIX systems:

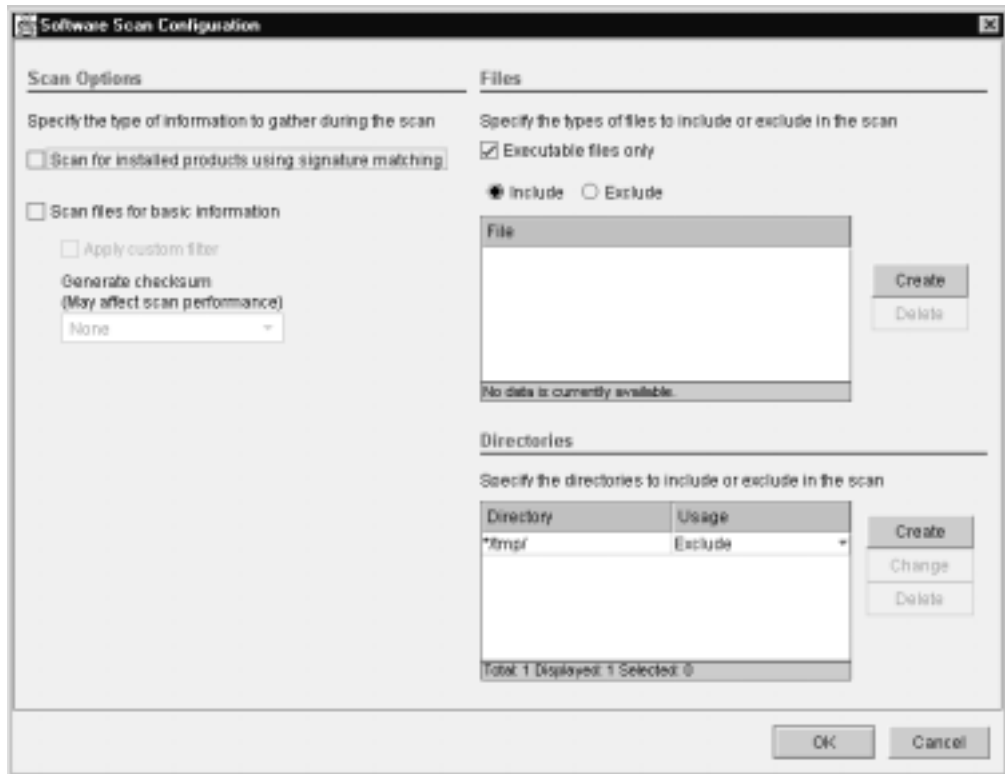


Figure 8. Software Scan Configuration Window for UNIX Systems

The following sections describe the options in the Software Scan Configuration window.

Scan Options: Under **Scan Options**, specify the type of information that you want to collect. Each option uses a different method to collect data.

- **Scan for installed products using signature matching.** In this type of scan, the list of signatures is distributed to the endpoint. This list is compared to the files scanned on the endpoint. When a file matches a signature, the signature data for that file is sent to the configuration repository. Signature data includes the name, size, and usually the quick checksum value of the file used to identify the software product.

Note: Before you scan using signatures, you must install the signatures. For more information about signatures, including the procedure to install them, see “Using signatures” on page 59.

- **Scan files for header information.** This option searches the header of scanned files for header information. Microsoft® Windows headers usually contain the following data: company name, product name, and product version. This option is not available for UNIX software scans. This option has no effect on scans of NetWare and OS/2 endpoints.
- **Scan files for basic information.** Select this option to collect the name and path of scanned files. Other information, such as date of creation, modification, and last access, is collected on some platforms.

The **Apply custom filter** option enables you to filter scans for basic information. When you select this option, scans for basic information are limited to the files listed in the custom filter. For more information about the custom filter, see “winvfilter” on page 157 or the Inventory online help.

The **Generate checksum** option allows you to collect the checksum value of scanned files. Checksum algorithms map the contents of a file to a single fixed value. By comparing these values, you can determine whether the contents of the file have changed over time, possibly due to data corruption or a virus. For example, you could gather the MD5 checksum value for all files in your enterprise, and then monitor this value for changes to detect “Trojan horse” programs.

Files: Use the **Files** scrolling list to specify the file names or file types that you want to include or exclude during the scan. To reduce scan time, limit the files that the profile will scan. You can provide specific file names to include or exclude, or you can include or exclude file types by providing a file extension such as *.exe. This list applies to scans for matching signatures, scans for header information, and scans for basic file information.

Select the **Executable files only** option to scan only executable files.

Directories: In the **Directories** scrolling list, specify the directories that you want to include in the scan, those that you want to exclude from the scan, or both. To reduce scan time, limit the directories that the profile will search. This list applies to scans for matching signatures, scans for header information, and scans for basic file information.

Note: If you specify two paths, one to be included in the scan and the other to be excluded from the scan, the include operation supersedes the exclude operation. For example, when specifying the C:/ directory to be excluded from the scan, and the C:/Windows directory to be included in the scan, the C:/Windows directory is scanned by Inventory.

Software patch scan options for PC

You should download and install from the Microsoft Web site the Windows Update Agent (WUA) on the Tivoli endpoints of your environment.

Before you run a patch scan that returns significant patch information, the .cab file should be located on the Tivoli region server under the directory \$(DBDIR)/inventory.

Table 3 lists the tools used for software patch scans:

Table 3. Patch tools

Resource	Description
Patch tools	<p>Windows Update (WU) offline scan file</p> <p>It is the security policy catalog in a Microsoft environment where missing patches are discovered running WUA. This file is downloaded and updated regularly using an automated process defined in the Automation Server workflow. For details on the .cab file changes, see http://support.microsoft.com/kb/926464.</p> <p>Depending on the installed WUA, Patch Management solution supports one of the following .cab files:</p> <p>wsusscn2.cab</p> <p>This new offline scan file is available in addition to the existing WU offline scan file, Wsusscan.cab. The Wsusscn2.cab offline scan file has a new format. To download the updated version of the .cab file see http://go.microsoft.com/fwlink/?LinkId=74689.</p> <p>To manage mixed environments, where different levels of WUA are installed, deploy the new .cab file only after having updated WUA on all endpoints.</p> <p>wsusscan.cab</p> <p>You can download this file from http://go.microsoft.com/fwlink/?LinkId=39043. By March 2007, Microsoft will no longer support the existing WU offline scan file. At that time, you will have to use the new Wsusscn2.cab file.</p> <p>QChain.exe</p> <p>Enables the installation of multiple patches without restarting the computer between each installation.</p>

The files will be distributed to the endpoints next time a patch scan is run.

For more details on these files, refer to the IBM Tivoli Configuration Manager 4.3.1 Patch Management Guide.

If you installed the WSUS Patch Automation solution, replace the mssecure.cab, mbsacli.exe, and ApprovedItems.txt files with the following files everywhere:

Table 4. WSUS patch management files

SUS Files	WSUS Files
mssecure.cab	wsusscan.cab
mbsacli.exe	WindowsUpdateAgent20-x86.exe
ApprovedItems.txt	ApprovedChanges.txt

Customizing a hardware scan

By default, Inventory scans systems for a number of hardware components. However, you can customize an inventory profile to scan only the hardware components that you select.

Note: You must collect processor, memory, and operating system data if you want to use the INVENTORY_HWARE query.

For PC hardware scans, you can configure the inventory profile to collect hardware information from the Desktop Management Interface (DMI) layer of a PC, and you can select the DMI attributes to include in the scan. You customize hardware scans from the desktop using the Hardware window.

PC hardware scans

To customize hardware scans of PC systems, select **PC->Hardware** in the left pane of the Inventory Administration window. The Hardware window is displayed.



Figure 9. Hardware Window for PC Systems

From this window, you can enable and configure the Tivoli hardware scanner, the DMI scanner, or both scanners.

Tivoli hardware scanner: The Tivoli hardware scanner uses a combination of hardware, system, and registry calls to obtain characteristics about hardware components. On some Windows operating systems, the scanner incorporates Common Information Model (CIM) and Windows Management Instrumentation (WMI) support to obtain standards-based information.

In this case, the data obtained by performing a hardware scan and reported to the database will be exactly the same data obtained by the WMI interfaces.

You can customize which hardware attributes that you want to scan for. For example, you could create an inventory profile that scans only for installed

memory. To customize the list of hardware data collected by the Tivoli hardware scanner, click **Configure**. The Tivoli Hardware Scanner window is displayed.

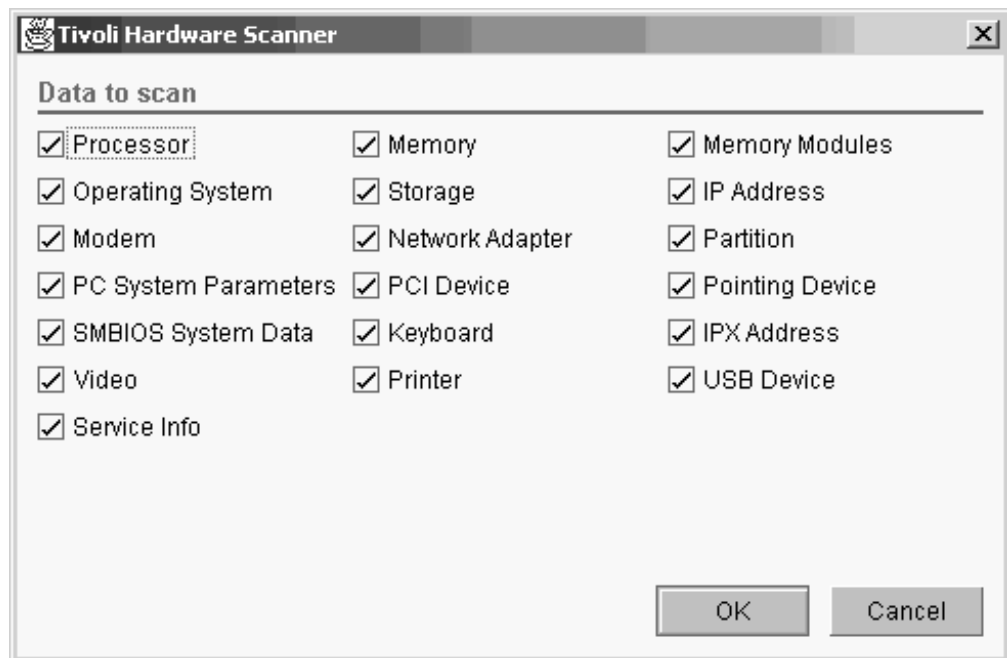


Figure 10. Tivoli Hardware Scanner window for PC Systems

DMI scanner: The DMI scanner scans the DMI layer of a PC for hardware information. DMI is an industry standard for describing and accessing information about PCs and PC components. Some PC systems contain a DMI service layer, which contains information about the PC on which it is installed. Using DMI standards, you can manage your PC hardware independent of a specific computer, operating system, or management tool.

For example, you can identify the serial number on systems from any manufacturer that has implemented the DMI service layer on their computers. Depending on the manufacturer, the DMI service layer also might contain information about the cost of ownership associated with your hardware.

To scan a DMI layer, you must create a list of the hardware attributes that you want to scan for. To create this list, you retrieve and then edit each type of DMI layer that you want to scan. See the Inventory online help for more information about this procedure.

For help with troubleshooting DMI scans, see “DMI” on page 281.

UNIX and OS/400 hardware scans

To customize hardware scans of UNIX and OS/400 systems, select **UNIX and OS/400->Hardware**. The Hardware window is displayed.

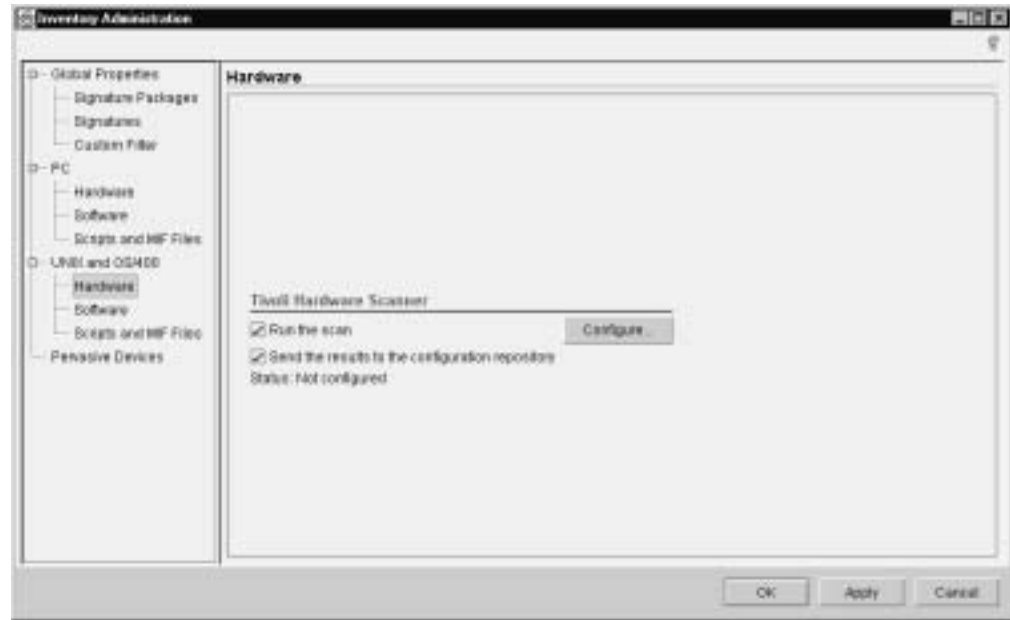


Figure 11. Hardware Window for UNIX and OS/400 Systems

From this window, you can enable and configure the Tivoli hardware scanner. To enable this scan, select one or both of the following options:

- **Run the scan** scans the endpoint and then stores the data on the endpoint in a MIF file.
- **Send the results to the configuration repository** collects the MIF file from the endpoint and sends the data to the configuration repository.

To customize the list of hardware data collected by the Tivoli hardware scanner, click **Configure**. The Tivoli Hardware Scanner window is displayed.

Note: The Memory Modules group applies only to Linux ix86 endpoints.

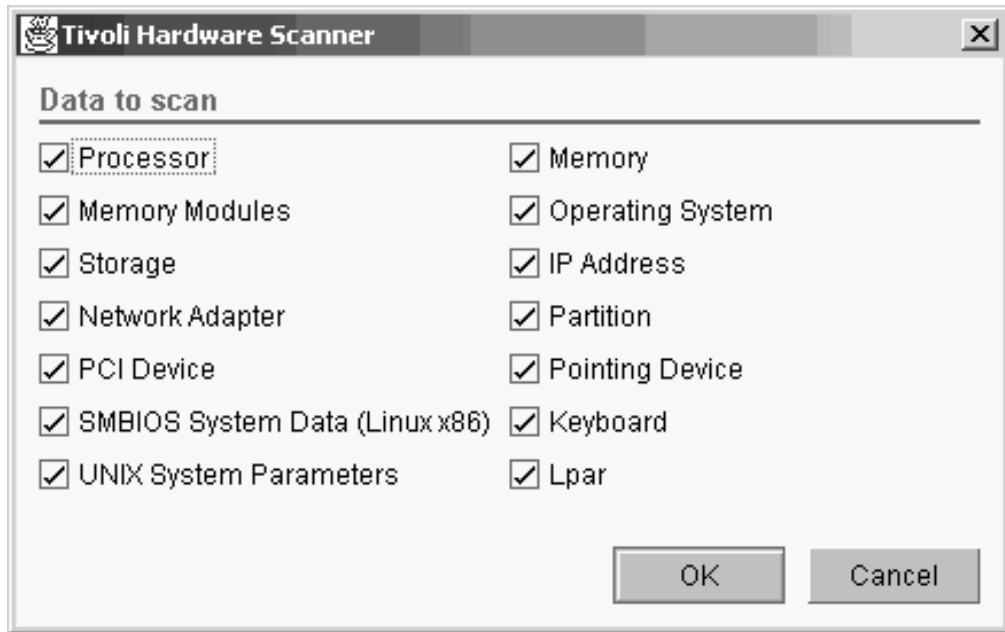


Figure 12. Tivoli Hardware Scanner Window for UNIX and OS/400 Systems

Using custom scripts and MIF files

You can customize an inventory profile to run a custom script during a profile distribution. You can run scripts before the scan, after the scan, or both. (Scripts that run after the scan complete before MIF files are read.) You can use this feature to generate a custom MIF file or to modify a MIF file created during a scan. For more information about using custom MIF files, see “Using custom MIF files” on page 66.

You can also customize an inventory profile to read a custom MIF file during a profile distribution. When a custom MIF file is read, the data from the file is sent to the configuration repository. Before you use Inventory to collect custom MIF files, you must create tables in the configuration repository to store the custom information. The MIF file must be present on the scan target when the scan occurs.

Note: You cannot run a custom script on NetWare endpoints.

To add a custom script or MIF file for PC systems, select **PC->Scripts and MIF Files**. To add a custom script or MIF file for UNIX and OS/400 systems, select **UNIX and OS/400->Scripts and MIF Files**. The Scripts and MIF Files window is displayed.



Figure 13. Scripts and MIF Files Window

In the upper text box, type or paste in a batch or command script that you want to run before an inventory scan. In the center text box, type or paste in a batch or command script that you want to run after an inventory scan. In these text boxes, you must enter the entire script, not the path to a script file.

In the lower text box, list any custom MIF files that you want to be read during a distribution. To add a file to the list, click **Add**. To delete a file from the list, click the file in this text box, and then click **Delete**.

Customizing scans of pervasive devices

You customize scans of pervasive devices from the desktop using the Pervasive Devices window.

To customize scans of pervasive devices, select **Pervasive Devices** in the left pane of the Inventory Administration window. The Pervasive Devices window is displayed.

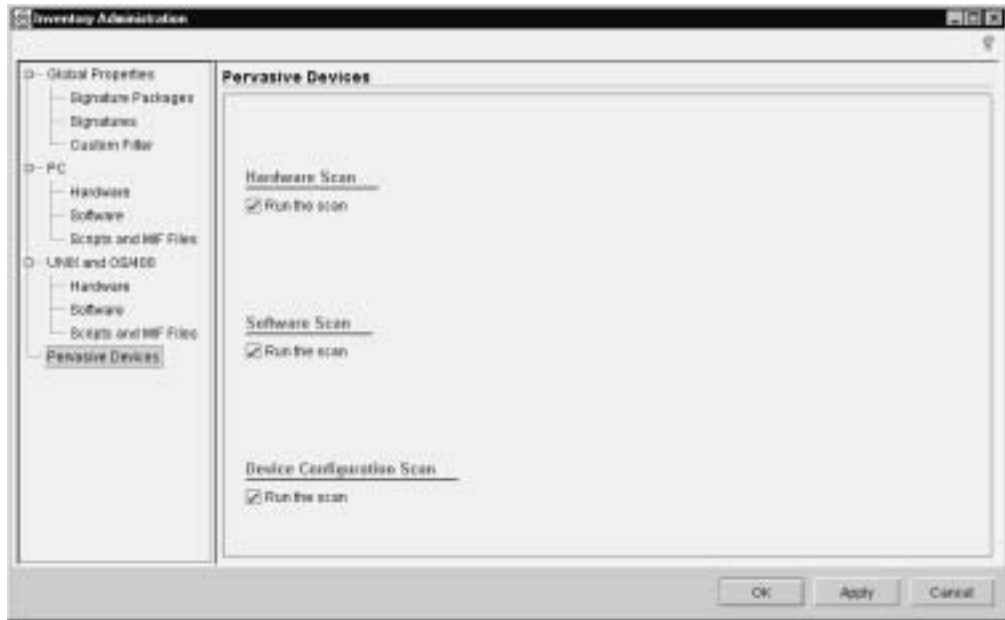


Figure 14. Pervasive Devices Window

From this window, you can enable or disable scans for hardware data, software data, and preferences for the pervasive devices in your enterprise:

- **Hardware Scan** collects hardware information about pervasive devices.
- **Software Scan** collects software information about pervasive devices.
- **Device Configuration Scan** collects the settings of configurable device settings such as time format, date format, or alarm sound.

Notes:

1. When you scan a pervasive device, the scan data is automatically sent to the configuration repository. Moreover, the **Replace with current results** option is set for all scans of pervasive devices.
2. Refer to the *User's Guide for Deployment Services* if the following tables remain empty in the cm_db after performing successfully a scan on a WinCE device:
 - WINCE_AGENT
 - WINCE_CFG
 - WINCE_FILE
 - WINCE_NATIV
 - WINCE_NET

Customizing inventory profiles from the command line

To customize an inventory profile using the command line, you can first get information about the profile's current settings and then change the settings.

To view the global settings for an inventory profile, use the **wgetinvglobal** command. To view the settings for scans of PC systems, use the **wgetinvpcfiles**, **wgetinvpchw**, and **wgetinvpcsw** commands. To view the settings for scans of UNIX and OS/400 systems, use the **wgetinvunixfiles**, **wgetinvunixhw**, and **wgetinvunixsw** commands. To view the settings for scans of pervasive devices, use the **wgetinvpvdconfig**, **wgetinvpvdhw**, and **wgetinvpvdsw** commands.

To change the global settings for an inventory profile, use the **wsetinvglobal** command. To change the settings for scans of PC systems, use the **wsetinvpcfiles**, **wsetinvpchw**, and **wsetinvpcsw** commands. To change the settings for scans of UNIX and OS/400 systems, use the **wsetinvunixfiles**, **wsetinvunixhw**, and **wsetinvunixsw** commands. To change the settings for scans of pervasive devices, use the **wsetinvpvdconfig**, **wsetinvpvdhw**, and **wsetinvpvdsw** commands.

Cloning an inventory profile

You can make a copy of an inventory profile by cloning it. (You must rename the profile.) This feature is especially useful if you have PC systems in your Tivoli environment that have files residing on different drives. You can clone the original profile and change the drive letter for the files to be scanned.

The following table shows the context and authorization roles required to perform this task:

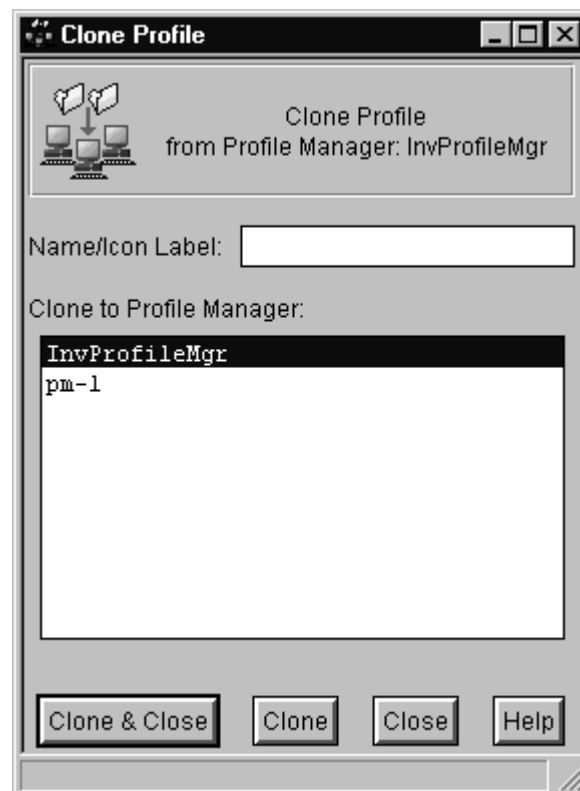
Operation	Context	Required Role
Clone a profile	Profile manager	senior or super

You can perform this task from either the Tivoli desktop or the command line.

Desktop

Complete the following steps to clone an inventory profile from the Tivoli desktop:

1. From a profile manager, select the profile you want to clone.
2. Click **Edit->Profiles->Clone**. The Clone Profile window is displayed.



3. Enter the name for the new profile in the **Name/Icon Label** field.
4. Select the profile manager in which the new profile will reside from the **Clone to Profile Manager** list.
5. Click **Clone & Close** to create the clone of the profile and return to the Profile Manager window.

Command line

To use the **wcrtprf** command to clone the **UNIX_First_Scan** inventory profile, enter the following command:

```
wcrtprf -c @UNIX_First_Scan \  
@ProfileManager:First_Scan_Profiles InventoryConfig UNIX_Update
```

where:

-c @UNIX_First_Scan

Specifies **UNIX_First_Scan** as the profile to clone. You must use the at sign (@) when specifying the profile name.

@ProfileManager:First_Scan_Profiles

Specifies **First_Scan_Profiles** as the profile manager to contain the new inventory profile.

InventoryConfig

Specifies the type of profile to clone.

UNIX_Update

Specifies **UNIX_Update** as the name of the new profile. Do not use an object path (/Regions/...) or registered name (@name). Specify the new name.

For more information about using the command line to clone inventory profiles, see the **wcrtprf** command in the *Tivoli Management Framework: Reference Manual*.

Deleting an inventory profile

When you delete a profile, it is removed from the Tivoli object database and its icon is removed from the profile manager.

Note: If you delete an inventory profile for which you have scheduled a distribution, you must delete the job from the scheduler. Deleting a profile does not automatically delete the job. See the *Tivoli Management Framework: User's Guide* for more information about using the scheduler.

The following table shows the context and authorization roles required to delete a profile:

Operation	Context	Required Role
Delete an inventory profile	Profile manager	senior or super

You can delete an inventory profile from either the Tivoli desktop or the command line.

Desktop

Complete the following steps to delete an inventory profile from the Tivoli desktop:

1. From a profile manager, select the icons of the profiles you want to delete and select **Edit->Profiles->Delete**. The Delete Profiles window is displayed.

Note: You cannot recover a deleted profile.



Disregard the message “The original profiles AND all of their copies will be deleted” because it does not apply to inventory profiles.

2. To confirm the deletion, click **Delete**. Inventory deletes the profile.

Command line

To use the **wdel** command to delete the **UNIX_First_Scan** profile and remove its icon from the **First_Scan_Profiles** profile manager in the **Inventory** policy region, enter the following command:

```
wdel /Regions/Inventory/First_Scan_Profiles/UNIX_First_Scan
```

or

```
wdel @InventoryConfig:UNIX_First_Scan
```

where:

/Regions/Inventory/First_Scan_Profiles/UNIX_First_Scan

Specifies the object path of the inventory profile to delete. You can specify object paths or registered names of Tivoli objects such as profiles and profile managers.

For more information about the **wdel** command, see the *Tivoli Management Framework: Reference Manual*. See Appendix B, “Commands,” on page 103 for more information about object paths.

Renaming an inventory profile

To rename an inventory profile, follow these steps:

1. Clone the profile. (See “Cloning an inventory profile” on page 47.) Specify the new name for the cloned profile.
2. Delete the original profile. (See “Deleting an inventory profile” on page 48.)

Chapter 4. Distributing Inventory profiles

After you create and customize your inventory profiles, you can distribute them to scan targets in your Tivoli environment. This chapter describes the steps that you take to distribute inventory profiles from the Tivoli desktop and from the command line interface. If you have not yet created inventory profiles, see Chapter 3, “Working with Inventory profiles,” on page 27 for instructions.

This chapter includes the following information about inventory scans:

- An explanation of what happens during an inventory scan and steps for distributing an inventory profile. (See “Distributing inventory profiles.”)
- Information about how to initiate a scan from an endpoint rather than distributing the scan from an inventory profile (See “Performing an endpoint-initiated scan” on page 55.)
- The procedure to scan machines that are disconnected from your Tivoli management region (Tivoli region). (See “Scanning disconnected systems” on page 56.)
- An explanation of how information is saved in the configuration repository. (See “Populating the configuration repository” on page 58.)
- Information about cancelling active inventory scans. (See “Cancelling scans” on page 58.)

Distributing inventory profiles

You can use the Tivoli desktop or command line interface (CLI) to distribute an inventory profile. You can distribute an inventory profile to an endpoint, a profile manager, or a resource group. Resource groups can contain either pervasive devices or users. You can distribute an inventory scan to only one type of target at a time: endpoints, pervasive devices, or users. However, you can distribute a scan to multiple targets of each type. For example, you can distribute a scan to 500 endpoints, but not to an endpoint and a pervasive device. For more information about resource groups, see *User's Guide for Deployment Services*.

Scanning the machines in your environment for the first time can be time-consuming. However, after you have scanned the machines once, you can use an inventory profile that is set to compare the results of the current scan with the results of the previous scan.

If you choose **Update with differences** on the Global Properties window, Inventory saves only new or changed data to the configuration repository. Because differences constitute a smaller amount of data, less data is sent across the network and saved in the configuration repository. See Chapter 3, “Working with Inventory profiles,” on page 27 for information about setting up scans.

Inventory provides information about events and errors that result from an inventory profile distribution. By default, this information is sent to the Inventory notice group. Using the `wsetinvglobal` command and `-l` option, you can configure Inventory to send this information to a log file, the Tivoli Enterprise Console console, the Inventory notice group, or any combination of those locations. See “wsetinvglobal” on page 187 for more information about the `wsetinvglobal`

command. See the *Tivoli Management Framework: User's Guide* for more information about viewing group notices. The following list includes some of the situations in which Inventory posts a notice:

- A scan is unsuccessful because a target is not available.
- An RDBMS operation fails.
- A custom MIF file is not found in a specified location.
- A custom MIF file is not formatted correctly.
- Warning messages related to license management extension.

By default, an inventory profile cannot be distributed to targets that are not subscribers to the profile manager that contains the inventory profile. However, using the **wsetinvglobal** command and **-s** option, you can configure a profile so that it can be distributed to targets that are not subscribers to the profile manager that contains the inventory profile. For more information, see “wsetinvglobal” on page 187.

Inventory uses the MDist 2 service to distribute profiles. MDist 2 enables you to assign a priority to profile distributions. You can choose high, medium, or low priority. You control priority processing of distributions by assigning a number of connections that a repeater can open for distributions with a given priority. Moreover, if no connections are available for a given priority, the repeater tries to borrow a connection from a lower priority. For more information about assigning priorities to distributions and setting maximum priority connections, see the *Tivoli Management Framework: User's Guide*.

The following table provides the authorization roles required to distribute an inventory profile:

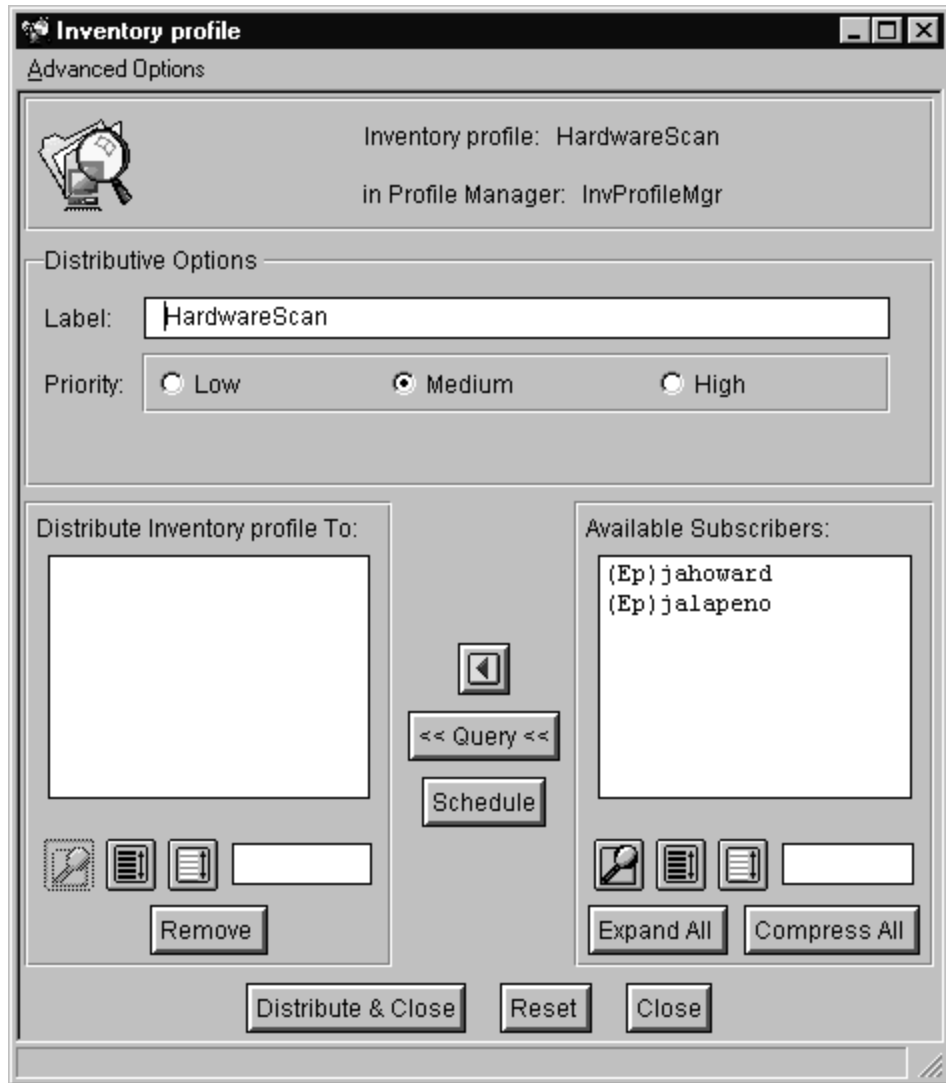
Operation	Context	Role
Distribute an inventory profile	Inventory profile	Inventory_scan , senior , or super

Note: To distribute an inventory profile from the Tivoli desktop, a user with the **Inventory_scan** role must also have the **user** role in the policy region in which the profile resides.

Desktop

Complete the following steps to distribute an inventory profile from the Tivoli desktop.

1. In a profile manager, right-click an icon for an inventory profile, and then select **Distribute**. The Inventory Profile window is displayed.



2. Set the priority for the distribution by selecting **Low**, **Medium**, or **High**.
3. In the **Available Subscribers** scrolling list, select the targets to which you want to distribute the profile. The **Available Subscribers** list shows the subscribers for the profile manager in which the profile resides.
4. Complete one of the following options:
 - Click the arrow button to move targets between the **Available Subscribers** and **Distribute Inventory Profile To** scrolling lists.
 - Click **Query** to select targets through a query. See Chapter 6, “Querying inventory information,” on page 87 for more information about using queries to select targets.
5. Optional: To modify the timeout settings for this profile, click the **Advanced Options** menu in the upper left corner of the window, and then select **Timeout Settings**. The Timeout Settings window is displayed.

Timeout Settings

Inventory profile: HardwareScan
in Profile Manager: InvProfileMgr

Endpoint Management

Wake on LAN endpoint ☐

Timeout Settings

Deadline: Month Day Year Hour Minute
1 1 2010 1 00

Notification Interval: Minutes

Send Timeout: Seconds

Execution Timeout: Seconds

Set & Close Set Close

6. From this window, you can set the following options:

Wake on LAN Endpoint

Specifies that the operation sends a Wake on LAN message to trigger the start of a system that is not available when an inventory profile is distributed. For scans of pervasive devices, this option starts the endpoint on which the Web Gateway component resides. This option works only on systems that have a network card that is enabled for Wake on LAN.

Deadline

Specifies the date and time at which a distribution expires (the date that it fails for unavailable target systems, including pervasive devices). If a target has not received a distribution by this time, MDist 2 no longer attempts the distribution. The default value is 72 hours from the time the profile is distributed.

Notification interval

Specifies the interval in which each repeater bundles and returns the completed results of the MDist 2 distribution. The default value is 1 minute.

Send timeout

Specifies the amount of time that a repeater waits for a target system to receive a block of data (for example, a repeater distributing an inventory profile to an endpoint). Each time a packet is sent over the network, the timeout for that packet is set to this value. This timeout is used to detect network or endpoint failures. If a timeout occurs, the

distribution remains in the repeater's queue and a retry occurs according to the **conn_retry_interval** value set with the Tivoli Management Framework **wmdist** command. The default is no timeout.

Execution timeout

Specifies the length of time that MDist 2 waits for an endpoint to return scan status after the profile has been distributed. For scans of endpoints, this value includes the time that it takes to run the scan and any custom scripts and make an upcall to SCS to request data collection. If the upcall to SCS fails, this value includes the time that it takes to send data to the inventory callback object. For scans of pervasive devices, the execution timeout is the length of time MDist 2 waits for a job to be created on the Web Gateway component.

If you plan to run a scan that will take a long time to complete, for example if you must scan a large number of drives, you would set this value to allow for that time. The default is no timeout.

7. Complete one of the following options:
 - Click **Distribute & Close** to distribute the profile to the selected targets. If the scan fails for any of the targets, a window is displayed to inform you which subscribers failed.
 - Click **Schedule** to schedule scans to automatically retry a scan to any target for which the scan might fail.

The Add Scheduled Job window is displayed. For more information about the Tivoli scheduler, see the *Tivoli Management Framework: User's Guide*.

Command line

You use the **wdistinv** command to distribute an inventory profile. You can list one or more targets or specify a text file that contains a list of targets for the distribution. This command also enables you to set a number of MDist 2 distribution options for an inventory profile, such as priority level and deadline for the distribution.

See “wdistinv” on page 122 for detailed information about the **wdistinv** command and usage examples.

Performing an endpoint-initiated scan

Inventory enables you to initiate a scan from an endpoint instead of distributing the scan from an inventory profile. This feature has many uses. For example, you can use endpoint-initiated scans to perform the following functions:

- Scan a machine each time it is rebooted.
- Scan a laptop system each time a mobile user logs in to the network.
- Scan a machine that is disconnected from the network.

Note: The endpoint-initiated scan is not available on Reliant-UNIX workstations.

You can run an endpoint-initiated scan automatically using a script or manually using the **wepscan** command. For more information about **wepscan**, see “wepscan” on page 127.

To initiate a scan on an endpoint, you must first distribute a configuration file to the endpoint. You can create a profile that distributes a configuration file to an endpoint without scanning it by selecting the **Distribute configuration file**

distribution option in the Global Properties window. For more information about creating an inventory profile, see Chapter 3, “Working with Inventory profiles,” on page 27.

As software scan options for PC and UNIX, ensure you select **Run the scan** to perform a scan on the endpoint, and **Send the results to the configuration repository** to create the DAT file on the endpoint. If the upload of the results is needed, you can run the **wepscan -s** command.

The following procedure is provided as a sample to demonstrate one use of the endpoint-initiated scan feature. This procedure starts a scan on a Windows NT® endpoint each time a user logs in. This type of endpoint-initiated scan requires two tasks: setting up the endpoint environment and initiating the scan. This example combines both tasks in a batch file and then runs the batch file whenever the user logs in.

To set up this type of endpoint-initiated scan, you would need to follow these general steps:

1. Create a batch file similar to the following example:

```
@ECHO OFF
if %SYSTEMROOT%/==/ set SYSTEMROOT=%WINDIR%
if %SYSTEMROOT%/==/ goto bad
CALL %SYSTEMROOT%\TIVOLI\LCF\1\lcf_env.cmd
if %LCFROOT%/==/ goto bad
if not exist %LCFROOT%\inv\SCAN\wepscan.exe goto notfound
%LCFROOT%\inv\SCAN\wepscan
goto quit
:bad
echo Could not source environment. Scan aborted.
goto quit
:notfound
echo Could not find Inventory Endpoint Scanner.
:quit
echo Done.
```

2. Place this batch file on your login server in a shared directory or on the local computer that you would like to scan on login.
3. Open the Windows NT User Manager. In the User Properties window, access the User Environment Profile. In the Logon Script Name text box, type the full path and name for the batch file in the text box.

Scanning disconnected systems

You can use the **wepscan**, **winviso**, and **wloadiso** commands to run isolated scans. An isolated scan is a scan of a system that is not in your Tivoli region. The system could even be disconnected from the network. You can run isolated scans on supported Windows and UNIX systems except Linux for S/390®.

Using the **winviso** command, you can copy to an endpoint all of the files necessary to scan a disconnected system (libraries, executables, signatures, and so on). You can then copy files from the endpoint to the disconnected system and run an isolated scan on the disconnected system using the **wepscan** command. This creates a .DAT file that contains the scan data. You must manually move the .DAT file from the disconnected system to the endpoint. You can then run the **wloadiso** command on the endpoint to send the scan data to the configuration repository.

To run an isolated scan, follow these steps:

1. Choose an endpoint that you want to use to scan the disconnected system. For example, you might choose an endpoint that is located near the disconnected system, so you can manually transfer files between the endpoint and the disconnected system using diskettes if necessary.
2. From a managed node with Inventory installed, run the **winviso** command, specifying the endpoint that you chose in step 1 and the Inventory profile scan as described in the following example:

```
winviso @InventoryConfig:profile_name @Endpoint:endpoint_name
```
3. Create a directory on the disconnected system to contain the files needed for scanning.
4. On the endpoint, locate the zip file (for example `w32-ix86.zip`) in the `$LCFROOT/inv/ISOLATED/common/` directory, unzip it, and manually copy the file from this directory to the directory on the disconnected system that you created in step 3. On the same directory, copy also the **wsusscan.cab** catalog, if you want to perform a patch management inventory scan.
5. Manually copy the `config.iso` file from the `$LCFROOT/inv/ISOLATED/profile_name` directory on the endpoint to the directory on the disconnected system that you created in step 3.
6. On the disconnected system, run the **wcodeset** command from the directory that you created in step 3. (Run the command with no options.) This command returns the names of the codeset files that you need to scan the disconnected system.
7. In the `$LCFROOT/inv/ISOLATED/common/codeset` directory on the endpoint, unzip the `codeset.zip` file.
8. Copy the codeset files that you need (the codeset files returned by the **wcodeset** command) to the disconnected system. Store the files in the directory that you created in step 3.
9. On the disconnected system, unzip the zip file from the directory that you created in Step 3, and run:

```
wepscan -i -n DAT_file_name
```

where

DAT_file_name

Is the name of the DAT file.

Note: On Linux and UNIX platforms, before running the **wepscan** command, run from the same directory the following script:

```
iso_env.sh
```

This script sets the environment variables.

10. Move the DAT file from the disconnected system to the `$LCFROOT/inv/ISOLATED/depot` directory on the endpoint. The DAT file is created in the directory from which you ran the **wepscan** command.

Note: The **wepscan** command installs Common Inventory Technology (CIT) Version 2.5 if not already present. Run this command only if logged on as **root**. This limitation derives from the Common Inventory Technology (CIT) component.

11. Run the **wloadiso** command from `$LCFROOT/inv/ISOLATED/depot` directory on the endpoint to send the scan data to the configuration repository. Before running the **wloadiso -f file.dat** command, set up the lcf

environment to access the shared libraries needed by the command. See the **wepscan** command for the procedure on how to set the environment.

For more information about isolated scans, see “wepscan” on page 127, “winvmgr” on page 161, and “wloadiso” on page 178.

Populating the configuration repository

Inventory saves the results of scans in the configuration repository. The configuration repository contains the tables that store the information collected during inventory profile distributions. The configuration repository schema describes the contents of the tables in the configuration repository and the relationships among the various tables. The *Database Schema Reference* provides a complete list of the tables, views, and queries provided with Inventory.

When you run a hardware or software scan on target machines, Inventory collects a pre-defined set of hardware and software components from the MIF files generated by the scanners. However, you can also use an inventory profile distribution to gather and save custom information that is not collected by a hardware or software scan. See Chapter 5, “Collecting custom information with Inventory,” on page 59 for information about using custom MIF files and extending the configuration repository. See “Adding a custom table to the configuration repository” on page 69 for instructions and example scripts that demonstrate how to create a new table. See “Viewing custom information” on page 85 for instructions and example scripts that demonstrate how to create a new view.

Cancelling scans

You can use the **wcancelscan** command to cancel active inventory scans. Using the **wcancelscan** command and the **-a** option, you can cancel all outstanding scans. Using the **wcancelscan** command and the **-i** option, you can specify one or more scans to cancel. You can also view information about cancelled scans using the **wcancelscan** command and the **-v** option.

For more information about the **wcancelscan** command, see “wcancelscan” on page 108.

Chapter 5. Collecting custom information with Inventory

This chapter covers the procedures for collecting custom and optional information with the inventory component of IBM Tivoli Configuration Manager. You can collect custom information by creating custom signatures, gathering user information with UserLink, or creating custom MIF files. You store this information in custom tables in the configuration repository. You can also configure the inventory component to scan PC systems for BIOS (Basic Input/Output System) information.

With some additional configuration, you can extend the functionality of an inventory profile distribution. Before you do any custom configuration, you should be familiar with the options on the inventory profile. See Chapter 3, “Working with Inventory profiles,” on page 27 for instructions about customizing an inventory profile. When you customize an inventory profile, you specify how to scan target machines and what information to save.

This chapter includes the following information about collecting custom and optional information with Inventory:

- Information about using signatures and signature packages to identify installed software applications. (See “Using signatures.”)
- Information about using custom MIF files to collect information that is not generated by hardware or software scans. (See “Using custom MIF files” on page 66.)
- Information about how to create tables in the configuration repository to hold custom data that you collect. (See “Designing custom tables” on page 67.)
- Information about creating history tables for tables that you create in the configuration repository. (See “Creating history tables for custom tables” on page 68.)
- Examples that demonstrate how to add custom tables and history tables to the configuration repository. (See “Adding a custom table to the configuration repository” on page 69.)
- Information about collecting custom information from users using UserLink, followed by an example. (See “Collecting information with UserLink” on page 70.)
- Information about views and an example that demonstrates how to add a view (See “Viewing custom information” on page 85.)

Using signatures

A signature is the set of information that identifies a certain software application, such as the name and size of the executable file for the software application. Inventory provides a default set of signatures that you can use to collect information about the software installed on the endpoints in your enterprise. With IBM Tivoli Configuration Manager, Version 4.3.1 a new signature file format is supported, XML-based, that replaces the old SWSIGS.INI format. The new signature file format enables you to specify new types of signatures such as Windows registry keys. To fully exploit the new file format, use the integration with IBM Tivoli License Manager. The old format is still supported by IBM Tivoli Configuration Manager.

These signatures are stored in the \$BINDIR/./generic/inv/SIGNATURES directory in the IBM_SoftwareCatalog.xml file.

To use these signatures, you must install them with the **winvsig** command as shown in the following example:

```
winvsig -a -f $BINDIR/./generic/inv/SIGNATURES/IBM_SoftwareCatalog.xml
```

This command installs the signatures in the configuration repository in the SIGNATURE table.

You can edit the signatures provided with Inventory, delete them, or you can add your own signatures. For example, you can add a signature that is not currently provided with Inventory or for an application that was developed in-house.

Inventory uses signatures to determine which software applications are installed on the machines you scan. When you run a signature scan on an endpoint, Inventory distributes the signatures to the endpoint, and searches for matched elements. When an element matches a signature, the signature data for that element is sent to the configuration repository. Because only data for matching elements is sent, this scan returns less data to the configuration repository than scans for basic file information or header information.

Signatures representing files

Signature data includes the name, size, and usually the Quick checksum value of the file used to identify the software product. This identifying file is usually the primary executable file for the product, for example NOTEPAD.EXE. The signature data collected during a scan, or updated with a software package including signature files, is stored in the configuration repository in the MATCHED_SWARE table. You can use the INVENTORY_SWARE query to return all the software information about a machine. See Chapter 6, “Querying inventory information,” on page 87 for more information about Inventory queries.

Signature scans return data for matching files only. Therefore, if you scan your enterprise using only signature scans, the configuration repository contains data only for matched files, not for all files on each scanned system. After you create a new signature or edit an existing one, you must then rescan your enterprise to gather the new or revised signature data.

However, if you have performed a scan for basic information, the configuration repository contains information about all files on scanned systems, not just those that match signature data. In this instance, you can write a query to view the desired file information, rather than modifying the signatures and then rescanning your enterprise.

Signatures representing registry keys

Signature data are used to find the specified registry key and its value in the Windows registry.

Note: A new signature catalog containing XML signature files can be downloaded from the following IBM web site.

<http://www-1.ibm.com/support>

In the integration with IBM Tivoli License Manager, when importing a signature catalog, it is recommended to avoid running an inventory scan or downloading the signature catalog to the IBM Tivoli License Manager agents by using the **wtlminfoget** command.

Inventory signatures are updated quarterly. Registered users can download them by following these instructions:

- Navigate to <http://www.ibm.com>
- Select "Support and downloads"
- Select "Software"
- Select "Download"
- In the search bar enter "Software Signature Catalog"
- Search for the most recent
 - Software Catalog YYYY-MM-DD for ITLM, version 2.2/2.3 - All Software **OR**
 - Software Catalog YYYY-MM-DD for ITCM 4.2.3 Fixpack 2 and Higher **OR**
 - Software Catalog 2007-05-31 for Inventory

Note: This last catalog is in the old INI format.

From the following FTP web site

ftp://ftp.software.ibm.com/software/tivoli_support/misc/Cand0/TivoliCatalog/

you can download the following catalog files:

- ITLM22_SoftwareCatalog_YYYY-MM-DD.xml
- YYMMDDSWSIG.S.INI
- ITLM22ForIBM_SoftwareCatalog_YYYY-MM-DD.xml
- IBMUseOnlySoftwareCatalog_YYYY-MM-DD.xml
- IBMSoftwareCatalog_YYYY-MM-DD.xml

Note: The files that can be used by Tivoli Configuration Manager are:

- ITLM22_SoftwareCatalog_YYYY-MM-DD.xml
- YYMMDDSWSIG.S.INI

The other files are for Tivoli License Manager only.

Since June 2007 the new naming convention of these files is the following:

- itlcm22-ibmProducts-fullSwCat-YYYYMMDD.xml
- itlcm22-allProducts-fullSwCat-YYYYMMDD.xml
- itlcm21-ibmProducts-fullSwCat-YYYYMMDD.xml
- itlcm21-allProducts-fullSwCat-YYYYMMDD.xml
- inventory-allProducts-fullSwCat-YYYYMMDD.ini

Note: The files that can be used by Tivoli Configuration Manager are:

- itlcm22-allProducts-fullSwCat-YYYYMMDD.xml
- inventory-allProducts-fullSwCat-YYYYMMDD.ini

The other files are for Tivoli License Manager only.

Modifying signatures

You can use either the Tivoli desktop or the command line interface to add, edit, or delete signatures in the configuration repository.

Note: From the Tivoli desktop GUI and from the command line interface can be modified, added, disabled, or removed only the Java Virtual Machine (JVM) signatures.

Signatures that point to files bigger than 2 GB are not supported.

Desktop

In the left pane of the Inventory Administration window, click the expansion box for **Global Properties** if necessary and then click **Signatures**. The Signatures window is displayed in the right pane.

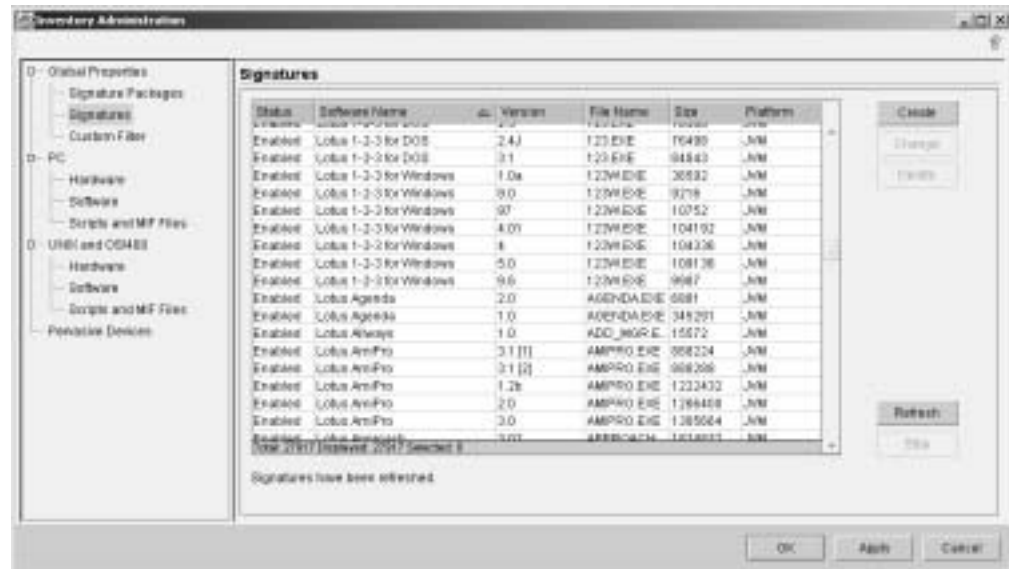


Figure 15. Signatures Window

See the Inventory online help for the procedures to add, edit, or delete a signature in the configuration repository using the Signatures window.

Note: In the integration with IBM Tivoli License Manager, this panel is read-only.

Command Line

You use the **winvsig** command to work with signatures from the command line. Using this command, you can add, edit, or delete signatures either individually or in batches reading from a file. See “winvsig” on page 172 for detailed information about the **winvsig** command and usage examples.

Defining signatures

This section explains how to create an XML file that contains the signature definitions. Software signatures are used in system management applications to represent and detect installed software products.

Note: This guide describes only signatures representing files and Windows registry keys. The XML file format enables you to specify also more complex signature types. To fully use these additional features, use the IBM Tivoli License Manager catalog manager available if the integration with IBM Tivoli License Manager is enabled. For a complete description of the XML file format, refer to the IBMSoftwareUpdate.xsd file described in Appendix D, “XML schema definition for signature catalogs,” on page 267.

The signature identifies a detectable software instance.

The signature catalog is arranged in a hierarchical structure. The root element in the catalog is the **<IBMSoftwareUpdate>** element, which contains all other elements. The following XML file is a sample signature catalog:

```
<?xml version="1.0" encoding="UTF-8"?>
<IBMSoftwareUpdate xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
Copyright="(C) Copyright IBM Corporation 2006. All rights reserved."
Date="2006-04-26 12:00:00.000000" Description="Sample Software Catalog"
Version="1.0">

  <Platforms>
    <Platform ID="p7" Name="Windows"/>
    <Platform ID="p5" Name="Solaris"/>
    <Platform ID="p4" Name="AIX"/>
    <Platform ID="p2" Name="Unix"/>
    <Platform ID="p8" Name="i5/OS"/>
    <Platform ID="p6" Name="HPUX"/>
    <Platform ID="p3" Name="Linux"/>
    <Platform ID="p1" Name="JVM"/>
  </Platforms>

  <Signatures>
    <File ID="s1" TargetPlatform="p4" Version="2.1" CustomerDefined="true"
Description="IBM Patch" Name="patch" Size="272974"/>

    <File ID="s2" TargetPlatform="p7" Version="1.7" CustomerDefined="true"
Description="Microsoft Windows Platform SDK" Name="WINDBG.EXE" Size="511760"/>

    <File ID="s3" TargetPlatform="p7" Version="2.0" CustomerDefined="true"
Description="Borland Multimate Advantage II" Name="WP.EXE" Size="258176"/>

    <File ID="s4" TargetPlatform="p7" Version="5.1.0.0" CustomerDefined="true"
Description="BMC Remedy Action Request System" Name="ASERVERD.BAT" Size="15112538"/>

    <File ID="s5" TargetPlatform="p2" Version="2.0" CustomerDefined="true"
Description="Sun Adrian&apos;s Rules &amp; Tools" Name="monlog" Size="2426"/>

    <File ID="s517" TargetPlatform="p5" Version="6.1" CustomerDefined="true"
IsDeleted="true" Description="BEA Weblogic" Name="weblogic.jar" Size="25665013"/>

    <WinReg ID="s5331" TargetPlatform="p7" Version="11.0" CustomerDefined="true"
Description="Microsoft Office Professional Edition 2003"
Key="HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\UNINSTALL\
{9011042D-6000-11D3-8CFE-0150048383C9}\" Type="string"
Data="Microsoft Corporation"/>

    <WinReg ID="s4286" TargetPlatform="p7" Version="5.1"
Description="WebSphere Studio Application"
Key="HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WEBSPPHERE STUDIO\
2.1\PRODUCT\COM.IBM.WSAPPDEV.GA.WIN32\EDITION"
Type="string" Data="Full"/>
  </Signatures>
</IBMSoftwareUpdate>
```

The **<Platforms>** section of the file is mandatory, while the **<Signatures>** section can be customized by specifying a value for the following attributes:

ID Is the signature identifier. Its value can be any string. It must be unique among all the signatures defined in the XML file.

TargetPlatform

Is the operating system to which the signature applies. Allowed values are the ID's defined in the <**Platforms**> section of the XML file. In the sample signature catalog shown above, the ID's are having values from "p1" to "p8".

Version

Is the version of the signature.

IsDeleted

Specifies if the signature should be deleted from the catalog. The default value is "false".

CustomerDefined

Specifies that the signature is not contained in the IBM signature catalog, but is defined by the customer. Its value must always be set to "true".

Description

Is the name of the software signature.

Name Is the name of the file that the signature refers to. For signatures that apply to Windows platforms, the name must be specified using uppercase characters.

Size Is the signature file size specified in bytes.

Key Is the fully-qualified path of the Windows registry key that should be discovered. If the value for this key ends with \, it means that this sub-key is the default sub-key.

Type Is the type of data contained in the Windows registry key. Allowed types of data are:

- binary
- dword
- expand_string
- string
- multi_string

Data Is the value of the Windows registry key. This attribute is optional. If not specified, the scan checks only the existence of the key and not its value.

Note: The attributes **Key**, **Type**, and **Data** can be specified only in the signatures representing registry keys. The attributes **Name**, and **Size** can be specified only in the signatures representing files. The other attributes can be specified in signatures representing both files and registry keys.

Using signature packages

Inventory enables you to logically group two or more signatures. A group of signatures is called a *signature package*. Signature packages use multiple signatures to identify a single software application or a collection of applications. For example, signature packages enable you to detect the following items:

- A suite of software. For example, a signature package can return a list of systems that have Lotus® Word Pro®, Lotus 1-2-3®, and Lotus Freelance Graphics® installed.
- A standard software configuration. For example, you can view a list of systems that have the mandatory software applications for an accountant or a bank teller.

- A set of files that define the specific version of an application, for example the primary executable of a software package and related files such as .DLL files or language packs.

You can use SIG_PACKAGE_QUERY to view the software data in the configuration repository that matches a signature package.

Notes:

1. Signature packages work only with data that is collected from signature scans. Before you can view data that matches a signature package, you must run a signature scan on the appropriate endpoints and make sure all scan data is in the configuration repository.
2. A signature package is not valid if one or more of that signatures it contains has been deleted. Before you remove a signature, make sure that the signature you want to remove is not being used in any signature packages.

You can use either the Tivoli desktop or the command line interface to add, edit, and delete signature packages and view the signature packages that you have created. Signature package data is stored in the configuration repository in the SIG_PACKAGE table.

Desktop

In the left pane of the Inventory Administration window, click the expansion box for **Global Properties** if necessary and then click **Signature Packages**. The Signature Packages window is displayed in the right pane.

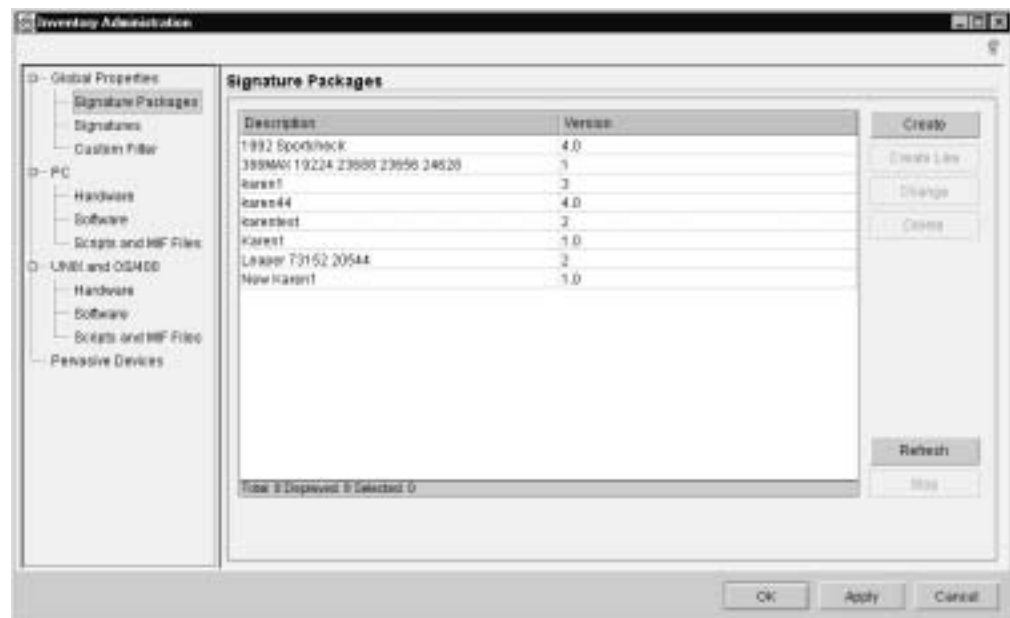


Figure 16. Signature Packages Window

See the Inventory online help for the procedure to add, edit, or delete a signature package using the Signature Packages window.

Command Line

You use the **winvpackage** command to work with signature packages from the command line. Using this command, you can add, edit, or delete signature packages either individually or in batches reading from a file. You can also view a

list of the signature packages that you have created and the signature packages that are not valid. See “winvpackage” on page 167 for detailed information about the **winvpackage** command and usage examples.

Using custom MIF files

You can use Inventory to collect information that is not obtained during a software or hardware scan by creating custom MIF files. You can create a custom MIF file manually or generate one using another application. For example, you could write a scanning program that collects custom data and then writes this data to a custom MIF file. You can then use Inventory to collect the data from the MIF file and send it to the configuration repository.

To collect custom MIF files, you set an inventory profile to read the MIF files during a profile distribution. An inventory profile distribution can read any MIF file that adheres to the Desktop Management Task Force (DMTF) 2.0 MIF format.

Before you use Inventory to collect custom MIF files, you must create tables in the configuration repository to store the custom information. See “Designing custom tables” on page 67 for more information.

To ensure that Inventory can read the MIF file, use the following guidelines:

- Ensure that the MIF file is compliant with DMTF 2.0 MIF standards, including the following:
 - Each group and attribute must have a name and description.
 - The type and size must be specified for each attribute.
- Read and follow the guidelines about custom tables in the “Designing custom tables” on page 67 and “Adding a custom table to the configuration repository” on page 69.
- Do not modify existing tables in the configuration repository.
- Do not name MIF groups the same as existing table names in the configuration repository.
- If you use tables in the MIF file to store multiple entries within a group, use the unique attribute or attributes as the key. Then, include those attributes as part of the primary key in the table that you create for the MIF group.
- Ensure that the name of the custom MIF file is unique within the Tivoli management region (Tivoli region). For example, do not use the names of the MIF files that Inventory generates during a scan. Inventory generates the following MIF files:
 - dmiscan.mif
 - mrmmbios.mif
 - tivfscan.mif
 - tivhscan.mif
 - tivrsan.mif
 - tivsscan.mif
 - tivwscan.mif
 - useradd.mif

The following is an example of how to create a MIF file.

```
START COMPONENT
NAME = "CUSTOMDATA.MIF"
DESCRIPTION = "Custom data that is not scanned"
```



```

START GROUP
  NAME = "SYS_INFO"
  ID = 1
  START ATTRIBUTE
    NAME = "SERIAL_NUMBER"
    ID = 1
    ACCESS = READ-ONLY
    TYPE = STRING(32)
    VALUE = "123456789012"
  END ATTRIBUTE
  START ATTRIBUTE
    NAME = "MAKER"
    ID = 2
    ACCESS = READ-ONLY
    TYPE = STRING(32)
    VALUE = "Compaq"
  END ATTRIBUTE
END GROUP

END COMPONENT

```

Using the custom scanning program

You can write your own scanning program that collects custom data. In order to use this scanning program on the endpoints, you have to download it to the endpoints. If you want to have Framework automatically download the scanning program to the endpoints, you can use the dependency mechanism.

You have to update the dependency set named Inventory -Config -depset by running the command `wdepset`.

Use the `wtransfer` command to move your scanning program to the gateway in the path you specified in the dependency set.

Designing custom tables

After you determine the information you want to collect, decide how that information will be organized in the configuration repository schema. Inventory uses pre-defined tables to store the results of Inventory hardware and software scans. Before you add custom information, you must create tables and columns in the configuration repository to store the information.

Consider the following guidelines when creating custom tables:

- Consider how much information you want to collect and how often to collect it. Consider the number of new tables and the number of columns in each table.
- Consider these naming guidelines for the new tables and columns. When selecting names for the new tables and columns consider the following:
 - The table and column names must be the names of the MIF file groups and attributes, respectively. For example, if you name the new table `USER_INFO`, you must also use `USER_INFO` for the name of the MIF group. The information stored in the `USER_INFO` group in the MIF file is stored in the `USER_INFO` table. Similarly, the column names you select for the `USER_INFO` table are the names that you must use for the MIF attributes that are included in the `USER_INFO` MIF group.
 - DB2® and Informix table and column names are limited to a maximum of 18 characters. Moreover, if you plan to create a history table for your custom

table, you must limit the name of the custom table to a maximum of 16 characters. See “Creating history tables for custom tables” for more information.

- Table and column names can include the following characters:
 - Lowercase a through z
 - Uppercase A through Z
 - Numeric characters 0 through 9
 - Underscores

You cannot use spaces. Table and column names must start with a letter.

- Custom tables and columns should not have the same name as existing tables and columns in the configuration repository. See the *Database Schema Reference* for existing table and column names.
- Do not use SQL-92 reserved words for custom table and column names. For more information about SQL-92 reserved words, consult a database administrator or SQL documentation.
- To determine the information type of each column, consider what type of information is stored in each column. The column type you use must match the attribute type in the MIF file.
- To determine the location of the tables in the configuration repository schema, consider how to organize the information in relation to other Inventory information.
- Consider whether the new tables should be related to existing tables in the schema. You can associate the tables you add to existing tables in the schema by using primary and foreign keys. For example, to relate a new table to an existing table in the configuration repository, make the primary key of the existing table a foreign key in the new table.
- The column COMPUTER_SYS_ID must be included in the primary key of each custom table.

Creating history tables for custom tables

To use history tracking with custom tables, you must create a history table when you create your custom table. You must name the history table the same name as the custom table, prefixed with an uppercase H and an underscore character (H_). For example, if your custom table is named MY_TABLE, the history table for this custom table must be named H_MY_TABLE.

If you are using a DB2 or Informix RDBMS for your configuration repository, you must restrict the names of your history tables to a maximum of 18 characters, including the H_ prefix, to comply with the naming convention for history tables.

The history table must contain the same column names as the custom table. Each column must be the same data type as the corresponding column in the custom table. In addition, the history table must contain the following columns, which the history tracking feature populates when it writes to history tables:

Column Name	Data Type
RECORD_ACTION	CHAR(6)
PRFL_ACTION	VARCHAR(20)
RECORD_TIME	TIMESTAMP, or appropriate date structure for your RDBMS.

For more information about these columns, see the *Database Schema Reference*.

Inventory does not use foreign key constraints in history tables. If you create a custom history table correctly, Inventory automatically detects and populates the history table.

Adding a custom table to the configuration repository

You must add the new tables and columns to the configuration repository before you use Inventory to collect custom MIF files. When you add a table, you specify the table name, number of columns in the table, data type and width of each column, and whether any of the columns are a primary key, foreign key, or both. Before you create a new table, you should be familiar with SQL and relational database concepts, such as primary keys, foreign keys, data types, NULL fields, and so on.

When adding a custom table, watch for any error messages that indicate the table was not created correctly.

The following SQL example adds the USER_INFO table to the configuration repository. The USER_INFO table can be updated when information changes and includes the following columns:

- LAST_NAME
- FIRST_NAME
- EMP_NUMBER
- OFFICE_NUMBER
- COMPUTER_SYS_ID

A custom MIF file created to match this table must include a group named USER_INFO that consists of attributes named LAST_NAME, FIRST_NAME, EMP_NUMBER, and OFFICE_NUMBER.

Note: The following example is for an Oracle configuration repository.

To add the USER_INFO table, first log into the configuration repository as the configuration repository user or owner. Next, type the following from the command line:

```
DROP TABLE USER_INFO;

CREATE TABLE USER_INFO(
  COMPUTER_SYS_ID  varchar2(64) NOT NULL,
  LAST_NAME        varchar2(64) NOT NULL,
  FIRST_NAME       varchar2(64) NOT NULL,
  EMP_NUMBER       varchar2(32) NOT NULL,
  OFFICE_NUMBER    varchar2(12) NOT NULL,
  constraint USERINFO_PK PRIMARY KEY (COMPUTER_SYS_ID),
  constraint USERINFO_FK FOREIGN KEY (COMPUTER_SYS_ID)
  references COMPUTER (COMPUTER_SYS_ID)
);
commit;
```

To delete information from custom tables using the **winvrmnode** command, insert the name of the custom table in the INVENTORY_TABLES table as shown in the following example:

```
insert into INVENTORY_TABLES(TABLE_NAME)
values('USER_INFO');
```

The **winvrmnode** command reads the table names from this table and then deletes the tables one at a time.

If you want to create a history table for the USER_INFO table, type the following from the command line:

```
DROP TABLE H_USER_INFO;

CREATE TABLE H_USER_INFO(
  COMPUTER_SYS_ID      varchar2(64),
  LAST_NAME            varchar2(64),
  FIRST_NAME           varchar2(64),
  EMP_NUMBER           varchar2(32),
  OFFICE_NUMBER         varchar2(12),
  RECORD_ACTION         char(6),
  PRFL_ACTION           varchar2(20),
  RECORD_TIME          DATE
);
commit;
```

Note that no foreign key constraints are used for the history table.

To delete information from history tables using the **winvrmnode** command, you must insert the name of the history table in the INVENTORY_TABLES table.

If you want to query the custom information, you might also want to create a new view. See “Viewing custom information” on page 85 for more information. You can also use the **wcrtquery** command to add custom queries to your query library. See the *Tivoli Management Framework: Reference Manual* for more information.

Running a query using a custom MIF file

In order to create and run a query to retrieve the data collected using a custom MIF file, follow these steps:

1. Connect to the Inventory database where you have run the schema script to add a custom table.
2. Insert a record in the table INVENTORY_TABLES specifying the new table that has been generated.

The SQL statement should be for example: `INSERT INTO INVENTORY_TABLES VALUES 'NEW_TABLE_NAME'`

Now it is possible to create a new INVENTORY_QUERY from the Tivoli desktop.

Collecting information with UserLink

Inventory provides a Web form called UserLink that you can use to collect custom information from users. UserLink is particularly useful for collecting information that you cannot gather with inventory scans, such as the office numbers and telephone extensions of employees.

Note: You must enable the Tivoli Web interfaces before using UserLink. For more information about the Tivoli Web interfaces, see the *IBM Tivoli Enterprise: Installation Guide*.

Collecting information with UserLink is a two-part process. First, administrators use a collection of forms named the User Data Template to design a form called

the User Data Form. Next, users complete the User Data Form, which is a Web form that is accessible through UserLink for Inventory.

The User Data Template enables you to specify the text boxes on the User Data Form. When you add or delete MIF groups or attributes with the User Data Template, you are indirectly specifying which MIF groups and attributes will comprise the MIF file used by the User Data Form, `useradd.mif`, the next time it is generated on target machines. The groups and attributes that you create are stored in the Tivoli name registry.

Each time users access UserLink for Inventory and open the User Data Form, the current set of groups and attributes in the name registry is used to dynamically create the User Data Form Web page. A text box for each attribute is displayed on the User Data Form.

When users click the Submit button on the User Data Form, the `useradd.mif` file is created and stored on the local machine.

Before you can store the user data in the configuration repository, you must create custom tables in the configuration repository to hold the data from the `useradd.mif` file. The tables and columns that you create must correspond with the MIF groups and attributes that comprise the `useradd.mif` file.

To save the information in the `useradd.mif` file into the configuration repository, you must customize an inventory profile to read `useradd.mif` during the profile distribution.

Overview

The example in the following sections illustrates how to collect the first name, last name, employee number, and office number from users and how to associate that information with the machine where it was entered. The following is an overview of the procedures required to use UserLink for Inventory to gather custom user information.

1. Determine the information that you want to gather from users and how that information should be organized in the configuration repository schema. See “Planning to collect user information” on page 72.
2. Determine which end users should have access to UserLink for Inventory. Create one or more Tivoli administrators that have authorization for UserLink. See “Giving users access to UserLink” on page 72.
3. Add new tables to the configuration repository to hold the custom information. (You should not store custom information in the pre-existing tables in the configuration repository.) See “Designing custom tables” on page 67, “Creating history tables for custom tables” on page 68, and “Adding a custom table to the configuration repository” on page 69.
4. Create the corresponding MIF groups and attributes using the User Data Template Web forms. This determines the text boxes that are included in the User Data Form. See “Creating MIF groups and attributes with the user data template” on page 75.
5. Instruct users to connect to UserLink for Inventory using the `userlink.htm` file. See “Using the `userlink.htm` file” on page 83. Users then complete the User Data Form. When they click the Submit button, the `useradd.mif` file is created. See “Completing the user data form” on page 83.
6. Customize an inventory profile to read `useradd.mif` during the profile distribution. See “Retrieving and saving user information” on page 84.

7. Distribute the inventory profile to retrieve the useradd.mif file. See “Retrieving and saving user information” on page 84.
8. To query the custom information, create a new view in the configuration repository. See “Viewing custom information” on page 85.

Planning to collect user information

You can use UserLink for Inventory to collect any information that users can enter. For example, they could enter equipment information that is not collected by a hardware or software scan, such as telephone or fax machine information. However, this type of static information can easily be collected in a custom MIF file that you create manually one time. (See “Using custom MIF files” on page 66 for more information.) UserLink for Inventory is especially useful for collecting dynamic information (for example: the current location of a machine, including the office number, building number, and the department number of the user) because you can ask users to open and complete the User Data Form on a regular basis.

Giving users access to UserLink

To provide access to UserLink, you must designate all UserLink users as Tivoli administrators on Tivoli Management Framework and give them the **Inventory_end_user** authorization role.

You can create a Tivoli administrator for each user. However, it is recommended that you create a single administrator for many users by adding the Tivoli management region server (Tivoli server) login name of each user in the Create Administrator window. You can also create an administrator that includes a global Tivoli server user account, and then publish that name and password to all users who need UserLink access. Users that have **senior**, **super**, **user**, or **admin** authorization in the Tivoli management region (Tivoli region) do not need the **Inventory_end_user** role to access UserLink.

Users who edit the User Data Template must have a user login name of **tmersrvd** or **nobody**. Users who complete the User Data Form can have any user login name. Therefore, you might want to create two administrators: an administrator named **tmersrvd** or **nobody** for those users who need access to the User Data Template, and another administrator for users who complete the User Data Form.

Note: To prevent end users from modifying critical files on the Tivoli server, ensure that all system directories and directories containing the Tivoli installation are write-protected.

When users click on any link in the UserLink for Inventory main page, they must provide their Tivoli server login name and password. Doing this allows users to securely connect to the Tivoli server and access the UserLink for Inventory functions.

You can create an administrator for UserLink purposes from either the desktop or the command line. For more information about creating administrators, see the *Tivoli Management Framework: User's Guide*.

Desktop

Complete the following steps to give users access to UserLink:

1. Run the Tivoli desktop on the Tivoli server.
2. Double-click the **Administrators** icon to open the Administrators window.
3. Select **Create->Administrator**. The Create Administrator window is displayed.



4. Complete the **Administrator Name/Icon Label** text box.
5. Complete the **User Login Name** text box:
 - If you are creating an administrator for users who edit the User Data Template, specify one of the following user login names:
 - **tmersrvd** (if your Tivoli server is a Windows NT or HP-UX system)
 - **nobody** (if your Tivoli server is any UNIX platform other than HP-UX)
 - If you are creating an administrator for users who will complete the User Data Form but will not need access to the User Data Template, enter the Tivoli server login name of one of the users that will have access to UserLink through this administrator:
6. (Optional) Enter a group name in the **Group Name** text box. See the section about Tivoli administrators in the *Tivoli Management Framework: User's Guide* for information about groups.
7. Click **Set TMR Roles**. The Set TMR Roles window is displayed. Complete the following steps:



- a. Select the **Inventory_end_user** role from the **Available Roles** list. To maintain a secure Tivoli environment, do not add any other Tivoli roles.
 - b. Click the left arrow button to move your selection to the **Current Roles** list.
 - c. Click **Set & Close** to save your changes and return to the Create Administrator dialog.
8. Click **Set Logins**. The Set Login Names window is displayed. Complete the following steps:



- a. In the **Add Login Names** text box, type the Tivoli server login name for a user who needs access to UserLink.
- b. Press **Enter**. The name appears in the **Current Login Names** list.
- c. Repeat steps 8a and 8b above until you have added all the users that need access to UserLink.

- d. Click **Set & Close** to save your changes and return to the Create Administrator dialog
9. Click **Create & Close**.

The icon for the administrator you create appears in the Administrators window for your Tivoli region.

To change settings for an administrator from the desktop, complete the following steps.

1. Right-click on the administrator icon you want to change to open the pop-up menu.



2. Select one of the following menu items to make changes to the administrator for UserLink access:
 - **Edit TMR Roles**—With this menu item you can give the administrator access to UserLink for Inventory.
 - **Edit Logins**—With this menu item you can add, change, or delete the logins of users who are assigned to this administrator and can access UserLink.

Command Line

Use the **wcrtadmin** command to give users access to UserLink.

The following example creates the administrator **end_user_admin**. It gives **end_user_admin** access to UserLink and assigns the user login names **jbrown**, **mgonzales**, and **cwu** to the administrator:

```
wcrtadmin -l jbrown,mgonzales,cwu \
-r @distinguished:InventoryUserLink,\
Inventory_end_user end_user_admin
```

where:

-l jbrown,mgonzales,cwu

Specifies the Tivoli server user IDs of the users you want to designate as administrators.

-r @distinguished:InventoryUserLink

Specifies the distinguished object that is created during the installation of Inventory to hold the methods needed for UserLink operations.

Inventory_end_user

Specifies the Tivoli role assigned to the **end_user_admin** administrator.

end_user_admin

Specifies the label for the administrator icon on the Tivoli desktop.

You can use the **wsetadmin** command to add more logins at a later time. The **wsetadmin** command also provides you with a way to give an existing administrator authorization for UserLink for Inventory. The following example gives the administrator **end_user_admin** access to UserLink for Inventory.

```
wsetadmin -r @distinguished:InventoryUserLink,Inventory_end_user \
end_user_admin
```


where:

-r @distinguished:InventoryUserLink

Specifies the distinguished object that is created during the installation of Inventory to hold the methods needed for UserLink operations.

Inventory_end_user

Specifies the new authorization role assigned to the administrator.

end_user_admin

Specifies the administrator whose settings are being changed.

See the *Tivoli Management Framework: Reference Manual* for information about **wcrtadmin** and **wsetadmin** command syntax and usage. Rather than entering each new login name individually through the Tivoli desktop or with the **wsetadmin** command, you might want to create a script to automate this process.

Creating MIF groups and attributes with the user data template

After you add the custom table or tables needed to store user information in the configuration repository, the next step is to create the corresponding MIF groups and attributes with the User Data Template. Use the names of the custom tables as the names of the MIF groups and the names of the columns as the names of the attributes in the User Data Template. Create a group for each new table you added to the configuration repository, and use the table name as the name of the group. The group name must match the table name exactly, including case. For example, the MIF group that corresponds with the USER_INFO table also must be named USER_INFO. Create attributes within the group for each of the columns in the table. For example, the USER_INFO group consists of attributes named LAST_NAME, FIRST_NAME, EMP_NUMBER, and OFFICE_NUMBER, which match the names of the columns in the table USER_INFO.

You can also use the User Data Template to delete MIF groups or attributes. To delete a MIF group, you must first delete all the attributes in the group. When you delete a MIF attribute, the text box that corresponds to that attribute is not displayed on the User Data Form the next time users access the form, and the column in the configuration repository by that name is not updated. You can also add attributes to collect more information, but, before you retrieve the information, you must add the corresponding columns to the tables in the configuration repository.

To create the USER_INFO group and its attributes with the Inventory Web pages for the administrator, complete the following steps:

1. To access the User Data Template Web page, open the following URL with a Web browser:

http://ServerName:OservPortId

where *ServerName* is the host name of the Tivoli server, and *OservPortId* is the number of the port where the object dispatcher is listening. By default, the port number for the object dispatcher process is 94. An example of this URL might be **http://lin:94**.

The Tivoli Web Access page is displayed.

2. Click the **Inventory** link. The Web browser displays a window with user name and password text boxes. (If there is not an Inventory link, Inventory has not been installed.)

3. Enter your Tivoli administrator login name and the password you used to log in to the machine. If authentication fails, ensure that the following items are set correctly:
 - When you run the **odadmin** command, “Remote client login allowed” should be set to TRUE. If it is not, enter the following command:
`odadmin set_allow_rconnect TRUE`
 - Your administrator roles must include **RIM_update** and **RIM_view**.
 - On a Tivoli server running a UNIX operating system, the user and password must be included in the `/etc/passwd` file as valid login IDs.
 - If the Tivoli server is an NT primary domain controller (PDC) or a backup domain controller (BDC), enter *Domain\Login* in the **User Name** field, where *Domain* is the name of the domain controlled by the PDC or BDC and *Login* is the administrator’s login name.

Note: Once you are authenticated, the Inventory Web pages can be accessed until the browser is closed.
4. To create MIF groups and attributes, click the **User Data Template** link. The Inventory User Data Template is displayed.



5. To add a MIF group named USER_INFO, complete the following steps:
 - a. Click the **Add MIF Group** link. The Add MIF Group page is displayed.

Add MIF Group

Current MIF Groups

MIF Group to be Added

Enter a name and description for the MIF group to be added.
The group name will also be the name of a table in the configuration repository. Therefore, use uppercase letters and do not use spaces in the name.

Group Name

Group Description

Add

Reset

[Return to User Data Template](#)

- b. In the Group Name text box, enter a name for the MIF group, for example, **USER_INFO**. Enter the name in uppercase. Do not include spaces, because the name must also be the name of a table in the configuration repository. Spaces are not allowed in table names.
- c. In the Group Description text box, enter a description of the MIF group, for example, **Employee Information**. A description is required by the Desktop Management Interface (DMI) 2.0 standard; it is displayed on the User Data Form.
- d. Click **Add** to add the MIF group. A Results page is displayed.



6. To add an attribute to the USER_INFO group, click the **Return to User Data Template** link. The User Data Template is displayed.
 - a. Click the **Add or Delete MIF Attribute** link.
The Add or Delete MIF Attribute page is displayed.

Add or Delete MIF Attribute

Current MIF Groups

Select the MIF group to which the attribute will be added or deleted.

☐ USER_INFO - Employee information

Operation

☐ Add Attribute
☐ Delete Attribute

[Return to User Data Template](#)

- b. Select the MIF group to which the attribute is to be added.
- c. Select **Add Attribute** and click **Go**.

The Add MIF Attribute page is displayed.

Add MIF Attribute

Current Attributes in the USER_INFO Group

The name, description, and type is listed below for each attribute in the group.

Attribute to be Added

Enter a name, description, and type for the MIF attribute to be added.
The attribute name will also be the name of a column in the configuration repository. Therefore, use uppercase letters and do not use spaces in the name.

At Attribute Type, select the type of information that the attribute will store.
If the information will be used in mathematical computation, select INT.
Otherwise, select STRING.

Attribute Name

Attribute Description

Attribute Type

Add Reset

- d. In the Attribute Name text box, enter a name for the attribute, for example **LAST_NAME**. Do not use spaces in the attribute name, because this is also be the name of a column in the configuration repository. Spaces are not allowed in the configuration repository table or column names.
- e. In the Attribute Description text box, enter a description for the attribute, for example **Last Name**. This description is displayed on the User Data Form next to a text box.
- f. From the Attribute Type list, select a type. Select **INT** if the information will be used in a mathematical computation. Otherwise, select **STRING**. The data type you select must correspond with the data type you specified for the corresponding column. For this example, select **STRING-64**. This corresponds to the data type and length of the column named LAST_NAME in the USER_INFO table that is added in “Adding a custom table to the configuration repository” on page 69.
- g. Click **Add** to add the attribute to the MIF group. A Results page is displayed.



- h. Repeat steps 6a through 6g for every MIF group and attribute you want to include in the useradd.mif file.
7. To see a current list of MIF groups and attributes, click the **View MIF Groups and Attributes** link on the User Data Template page. The MIF Groups and Attributes page is displayed.

MIF Groups and Attributes

Group Name
USER_INFO

Group Description
Employee information

Attribute Name
EMP_NUMBER
Attribute Description
Employee Number
Attribute Type
INT32

Attribute Name
FIRST_NAME
Attribute Description
First Name
Attribute Type
STRING(64)

Attribute Name
LAST_NAME
Attribute Description
Last Name
Attribute Type
STRING(64)

8. To see how the information you have entered is displayed on the User Data Form, click the **Preview User Data Form** link on the User Data Template page. A preview of the User Data Form that users will see is displayed.

User Data Form

Group: Employee information

Employee Number:

First Name:

Last Name:

Office Number:

[Return to User Data Template](#)

Using the userlink.htm file

To connect to the UserLink for Inventory features, users open the userlink.htm file with a Web browser. Endpoints receive the userlink.htm file when they log into the gateway. The file is stored in the \$DRIVE:/etc/Tivoli/\$LANG directory, where \$DRIVE is the drive where the Tivoli agent is installed and \$LANG is the language used on the endpoint. For more information about \$LANG, see the *Tivoli Management Framework: Release Notes*.

Note: It is recommended that users add the userlink.htm page to the list of bookmarks in their Web browsers so they can access it easily.

Completing the user data form

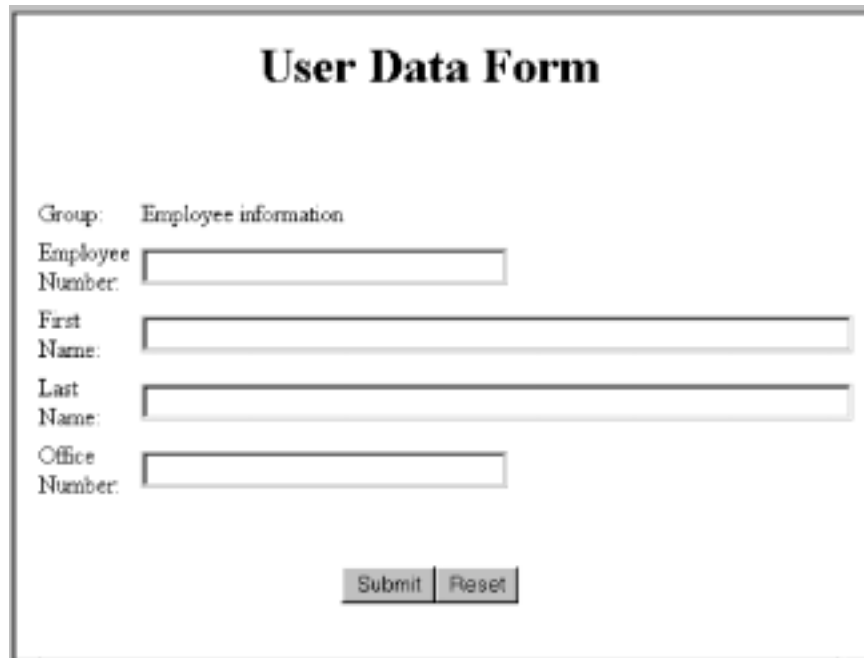
You must instruct users to access UserLink for Inventory and complete the User Data Form with the relevant information. The User Data Form is generated each time users click the User Data Form link from UserLink. To ensure that this form includes the most current text boxes, instruct users to reload the User Data Form before entering data into it.

When the users complete the User Data Form, the useradd.mif file is created on the endpoint in the \$LCF_BASE_DIR/inv/SCAN directory. The \$LCF_BASE_DIR is the directory in which you installed the Tivoli agent.

It is recommended that you either immediately retrieve and save the useradd.mif file in the configuration repository or move it to a different directory after it is generated. The useradd.mif file is overwritten every time a user clicks the **Submit** button on the User Data Form.

To connect to UserLink and complete the User Data Form, follow these steps:

1. Open the `userlink.htm` file with a Web browser. See “Using the `userlink.htm` file” on page 83 for location information.
The preliminary UserLink for Tivoli page is displayed.
2. Click the **Web Interface for Tivoli** link. The interior UserLink for Tivoli page is displayed.
3. Click the **UserLink for Inventory** link. The UserLink for Inventory page is displayed.
4. Click the **User Data Form** link. A prompt for your user name and password is displayed.
5. Enter the name and password configured for the end user. The User Data Form page is displayed.



The screenshot shows a web form titled "User Data Form". It contains several input fields: "Group" with a dropdown menu showing "Employee information", "Employee Number" with a text box, "First Name" with a text box, "Last Name" with a text box, and "Office Number" with a text box. At the bottom right, there are two buttons labeled "Submit" and "Reset".

6. Complete the text box or text boxes on the User Data Form with the appropriate information.
7. Click **Submit**. (When the users complete the User Data Form, the `useradd.mif` file is created on the local machine.)

Retrieving and saving user information

To retrieve the `useradd.mif` files from each user, you must customize an inventory profile to read the `useradd.mif` files. To customize the inventory profile from the Inventory GUI, enter `useradd.mif`, including its path, in the text box for custom MIF files in the Scripts and MIF Files window. To customize the inventory profile from the command line, specify `useradd.mif`, including its path, using the **`wsetinvpcfiles`** or **`wsetinvunixfiles`** command and the **`-m`** option. For more information, see “Customizing an inventory profile” on page 32.

After all the users have completed the User Data Form on their machines, you can use the customized inventory profile to collect the `useradd.mif` file from their machines and save the user information into the configuration repository. During the distribution, the information in the `useradd.mif` file is saved in the configuration repository in the custom tables you created.

Viewing custom information

RDBMS views are created so groups of information in a database can be accessed easily by a query. A view is a way to group information from related tables. For example, the `PROCESSOR_MODEL`, `PROCESSOR_SPEED`, and `COMPUTER_SYS_ID` columns reside in different tables throughout the configuration repository. A view named `PROCESSOR_VIEW` includes all of the information in these columns. So, instead of running a query for each table, you can access all the information with one query by using `PROCESSOR_VIEW`.

To query custom information, create a new view by creating a script that adds the new view to the configuration repository.

Remember the following rules when creating a new view:

- Do not edit the views provided with Inventory.
- Make sure that the new view contains the `TME_OBJECT_ID` and `TME_OBJECT_LABEL` columns. If you want to create a subscription list using the **wruninvquery** command, run one of the subscription queries provided with Inventory or run a new query that returns the names of machines that meet the query criteria.
- Add the name of the view to the `QUERY_VIEWS` table in the configuration repository. If you do not add the name of the view to this table, the view will not be available from the ellipses (...) button on the Create Query or Edit Query dialog. See “Creating a query” on page 88 for more information about selecting views from the query dialogs.

For example, the following script creates a view named `LOCATION_USER_VIEW`. This view enables you to query the information in the `LAST_NAME`, `FIRST_NAME`, `EMP_NUMBER`, and `OFFICE_NUMBER` columns in the custom table `USER_INFO`, which was created by the example in the “Adding a custom table to the configuration repository” on page 69. Note that some of the lines in the **select** statement consist of a table name followed by a column name, separated by a period. If you specify a column that resides in more than one of the listed tables, you must enter the table name and a period before the column name in the **select** statement.

Note: The following example applies to an Oracle RDBMS.

```
drop view LOCATION_USER_VIEW
create view LOCATION_USER_VIEW
as
select
  TME_OBJECT_ID,
  TME_OBJECT_LABEL,
  LAST_NAME,
  FIRST_NAME,
  EMP_NUMBER,
  OFFICE_NUMBER,
  USER_INFO.COMPUTER_SYS_ID
from USER_INFO, COMPUTER
where
  USER_INFO.COMPUTER_SYS_ID =
    COMPUTER.COMPUTER_SYS_ID;

commit;
insert into QUERY_VIEWS values ('LOCATION_USER_VIEW');
commit;
```

After you write the file, save it as *filename.sql*, and then close it. You can add the new view to the configuration repository by running the file as the user `invtiv` with an interactive SQL tool.

You can use the new view with any new or existing query by either entering the name of the view in the **Table/View Name** text box on the Create Query, Edit Query, and Inventory Query windows or by clicking the ellipses (...) button next to the **Table/View Name** text box (if you added the name of the view as a row to the `QUERY_VIEWS` table in the configuration repository).

Chapter 6. Querying inventory information

The Tivoli Management Framework query facility enables you to create SQL statements to retrieve information gathered from inventory profile distributions.

The query feature consists of query libraries and queries. Query libraries reside in policy regions and are created to contain queries. Each query is designed to retrieve a specific set of information from a specific view or table in the configuration repository. There are several pre-defined queries provided with Configuration Manager. You can also create your own queries. This chapter describes the different ways you can use queries to get inventory information from the configuration repository, including the following:

- An explanation of how to use the pre-defined queries provided with Inventory. (See “Using the inventory, pervasive, and subscription queries.”)
- An example of how to create your own query from the Create Query window. (See “Creating a query” on page 88.)
- An explanation of how to run a query you have created and how to view or save the results. (See “Running a query” on page 92.)
- Steps for using a query to select target machines for an inventory profile distribution. (See “Defining subscribers with queries” on page 94.)
- Steps for using a query to retrieve inventory information about one machine in a Tivoli management region (Tivoli region). (See “Querying for information about one client” on page 96.)
- An explanation of how to query interconnected Tivoli regions and display the data for each Tivoli region separately. (See “Using queries in interconnected Tivoli regions” on page 98.)

For a complete description of the Tivoli Management Framework query facility and query commands, see the *Tivoli Management Framework: User's Guide*.

Using the inventory, pervasive, and subscription queries

Inventory provides three sets of queries that access scan results in the configuration repository:

- The inventory queries retrieve different sets of hardware and software information from the configuration repository. For example, the INVENTORY_HWARE query returns basic hardware information about all machines.
- The pervasive queries retrieve different sets of hardware and software information about pervasive devices from the configuration repository. For example, PALM_CFG_QUERY returns configuration information for all Palm OS devices.
- The subscription queries that are provided with Inventory return lists of machines that meet the specifications of the query. For example, the AIX_SUBSCRIPTION query returns a list of machines that have an AIX[®] operating system.

The query libraries and queries that are provided with Inventory are created during the installation process. See the *Planning and Installation Guide* for more information about creating the inventory, pervasive, and subscription queries.

Creating a query

You can create and customize a query to retrieve the information you want from the configuration repository.

When you create a query, you specify the RIM object in which to search for information and the set of information you want to retrieve. Before you create a query, decide what information you want to access. Then, determine which tables and columns in the configuration repository contain that information and include them in the query. The *Database Schema* poster shows the configuration repository data model. The *Database Schema Reference* contains a brief description of each table.

Once you know which tables and columns you need to query, you can select the view that contains those columns. See the *Database Schema Reference* for a list of the views. If there is not a view that contains all the columns that you would like to include in your query, you can create a new view. See “Viewing custom information” on page 85 for instructions about creating a new view.

When you create a query, you must specify the following in the Create Query window:

- In the **Query Name** text box, a query that is unique in the Tivoli region.
- In the **Repository** text box, a RIM object that determines which tables and views you can use for the query.
- In the **Table/View Name** text box, a table or view within the repository to run the query against. The table or view that you select determines which columns you can use for the query.
- In the **Chosen Columns** text box, the columns within the table or view that are part of the retrieved information.

You can also specify a description and one or more where clauses for the query. Use the **Where Clause** section of the window to construct an SQL search clause that specifies which information the query will return. The query example in this section demonstrates how to use several features of the query facility.

For a complete description of how to use the Tivoli Management Framework query facility, see the *Tivoli Management Framework: User's Guide*.

The following table provides the authorization roles required to perform this task:

Operation	Context	Role
Create a query	Query library	Query_edit, admin, senior or super

You can create this query from either the Tivoli desktop or the command line.

Desktop

The following procedure provides an example of how to create a query from the Tivoli desktop. The query that is created in this example retrieves the following information from the configuration repository:

- The machines that have Microsoft Word installed and at least 64 MB of RAM available.
- The version of Microsoft Word that is installed on each machine.

Use the following steps to create a query from the Tivoli desktop:

1. From the policy region, double-click the icon of the query library in which the query will reside. The Query Library window is displayed.

Note: Queries must be created in query libraries. You can either create a query library or select an existing query library in which to store it.

2. Click **Create->Query**. The Create Query window is displayed.

3. In the **Query Name** text box, enter a unique name for the query, such as MSWord_Finder. The name can be any set of alphanumeric characters, uppercase letters, or lowercase letters. Spaces are allowed.
4. In the **Description** text box, enter a brief description of the query. This text box is optional. The description of this query will be Finds version of MSWord.
5. From the **Repository** drop-down list, select an RDBMS database against which the query will run. The **Repository** drop-down list shows the names of RIM objects in the local Tivoli region. This query will access scan results, so select **inv_query** to access the configuration repository.

6. In the **Table/View Name** text box, click the ellipses (...) button to select from a list of views in the repository. This list shows the views listed in the QUERY_VIEWS table in the configuration repository. The views for this query are INVENTORYDATA and INST_SWARE_VIEW, because they include the columns that contain information about software and physical memory.

Note: Changing the entry in the **Table/View Name** text box populates the **Available Columns** list and clears the rest of the text boxes on the window, including the **Chosen Columns** list, **Where Clause**, and **Additional Clause** text boxes.

7. Select **No Duplicates** to remove duplicate results from the query results.
8. In the **Available Columns** list, select the columns from which you want to retrieve information and move them to the **Chosen Columns** list with the left arrow button.

Note: You might not be able to see the full name of the column in this list. You can expand the **Available Columns** list by clicking and dragging the edge of the window.

Because you want this query to search for information about software and memory, choose the following columns:

- INST_SWARE_VIEW.SWARE_DESC
- INST_SWARE_VIEW.SWARE_VERS
- INVENTORYDATA.PHYSICAL_TOTAL_KB

Because the query returns a list of machines, include the TME_OBJECT_LABEL column as well so that you can identify all the machines in the query.

9. Create an SQL search clause in the **Where Clause** text area of the window to specify what information the query will return. Use the **Column Name** and **Column Value** text boxes with the operator buttons to create the SQL clause. To create the first SQL statement to be added to the **Where Clause** text box, complete the following steps:

- a. To define the software that the query will search for, enter SWARE_DESC in the **Column Name** text box

-OR-

Click the ellipses (...) button to display the list of **Available Columns**, and select SWARE_DESC from the list of column names.

- b. Choose a logical operator to establish a relationship between the entry in **Column Name** and the entry in **Column Value**. For this query, leave the equals sign (=) as the logical operator, because the **Column Value** text box will be used to specify the software for which the query should search. (If you want to search for machines that do not have a particular software, you would select **Not**.)
- c. Click the ellipses (...) button in the **Column Value** section. Select **Microsoft Word** and click **Close**, because the query should search for machines that have Microsoft Word associated with the machine ID. This completes the search criteria for this where clause.

Note: If the selected table or view has a large number of rows, the **Column Value** window might take a long time to appear.

- d. Click **Add** to add the criteria to the **Where Clause** text box.

10. To narrow the results further and return only machines that have at least 64 MB of memory, add the next SQL statement to the **Where Clause** text box. Complete the following steps:
 - a. In the **Column Name** text box, enter the name of the column that stores information about physical memory, which is **PHYSICAL_TOTAL_KB**.
 - b. From the menu next to the **Column Value** text box, select **>=** as the logical operator so the query will return both machines with the specified amount of physical memory and machines with more than the specified amount.
 - c. In the **Column Value** text box, enter the minimum amount of memory that the machines should have, which is **64000 KB**. (Memory must be entered in KB.)
 - d. Click **Add** to add the second where clause to the **Where Clause** text box.
11. The final where clause for this query specifies that the query should return information only about machines in a particular Tivoli region. To narrow the query to all the machines in a particular Tivoli region, you specify that information should be returned only for machines with a **TME_OBJECT_ID** that includes the number of the Tivoli region. To do this, complete the following steps:
 - a. In the **Column Name** text box, enter **TME_OBJECT_ID**.
 - b. From the menu next to the **Column Value** text box, select **LIKE** as the logical operator. This enables you to use a wildcard character (%) when defining the Tivoli object ID values for which the query should search.
 - c. Click the ellipses button (...). The **Column: TME_OBJECT_ID** window is displayed, which lists the **TME_OBJECT_ID** for each machine in the configuration repository. The first number of the **TME_OBJECT_ID** is the region number of the Tivoli region in which the object resides.
 - d. In the **Column: TME_OBJECT_ID** window, select the **TME_OBJECT_ID** of a machine that includes the region number for the correct Tivoli region and click **Close**. This populates the **Column Value** text box with the Tivoli object ID.

See “Using queries in interconnected Tivoli regions” on page 98 for information about determining the numeric ID of the Tivoli region. See “Querying for information about one client” on page 96 for instructions on viewing information about a particular client.
 - e. In the **Column Value** text box, change the **TME_OBJECT_ID** to be the Tivoli region number with a wildcard character, such as **105620229%**. To do this, delete the part of **TME_OBJECT_ID** after the Tivoli region number and add a percent sign (%). The query will return results only for machines in the Tivoli region that corresponds to that number. This specifies that any client in the specified Tivoli region will be included in the query results.
 - f. Click **Add**. The final clause of the where clause is displayed in the **Where Clause** text box.
12. In the **Additional Clauses** text box, add the following clause:


```
INVENTORYDATA.COMPUTER_SYS_ID =
INST_SWARE_VIEW.COMPUTER_SYS_ID
```

Note: If you choose to query more than one table or view, you must create a join of the tables or views in the **Additional Clauses** panel.
13. To create the query, click **Create**, or click **Create & Close** to create the query and return to the query library window.

14. To run the query, click **Run Query**. For more information about running queries and saving results, see “Running a query.”

Command line

The following example uses the **wcrtquery** command to create the MSWord_Finder query from the Desktop section.

```
wcrtquery -d "Finds version of MSWord" \  
-r inv_query \  
-v INVENTORYDATA.INST_SWARE_VIEW \  
-c INST_SWARE_VIEW.SWARE_DESC -c INST_SWARE_VIEW.SWARE_VERS \  
-c INST_SWARE_VIEW.TME_OBJECT_LABEL \  
-c INVENTORYDATA.PHYSICAL_TOTAL_KB \  
-w "(INST_SWARE_VIEW.SWARE_DESC='Microsoft Word') and \  
(INVENTORYDATA.PHYSICAL_TOTAL_KB>=64000) and \  
(TME_OBJECT_ID like '1947203709%')\" INVENTORY_QUERY MSWord_Finder
```

See the *Tivoli Management Framework: Reference Manual* for more information about the **wcrtquery** command.

Running a query

After you create a query to retrieve information about your Tivoli environment, you can run it and either view the results or save the results in a file.

The following table provides the authorization roles required to perform this task:

Operation	Context	Role
Run a query	The policy region that contains the query library of the applicable query	senior, super, Query_execute, or Query_edit

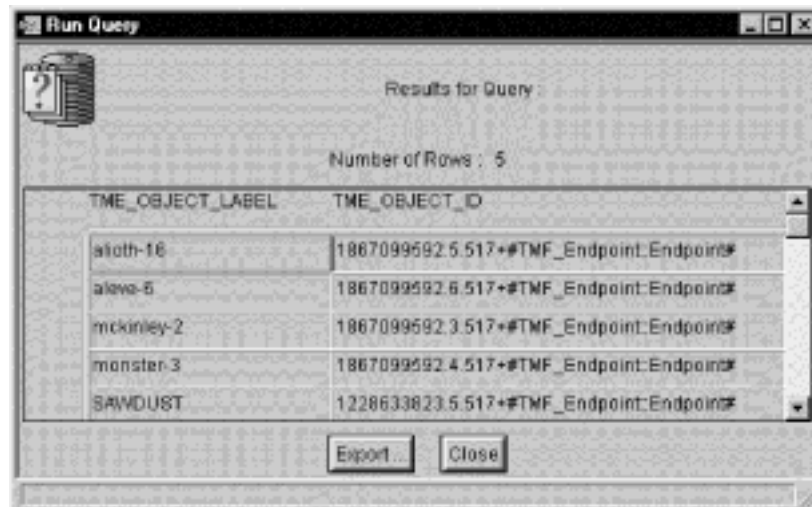
You can run a query from either the Tivoli desktop or the command line.

Desktop

To run a query from the Tivoli desktop, do *one* of the following:

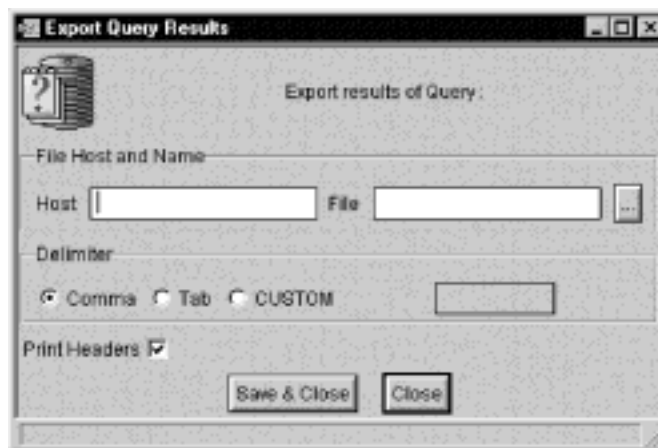
- In a query library, right-click the icon of a query and select **Run Query**
- Select **Run Query** from either the Create Query or Edit Query window.

The Run Query window is displayed, showing the results of the query in tabular form.



If you want to save the results of the query, you can do so from the Run Query window. Complete the following steps:

1. On the Run Query window, click **Export**. The Export Query Results window is displayed.



2. In the **Host** text box, enter the name of the managed node on which you would like to save the results file. If you do not specify a managed node, you will receive an error message.
3. In the **File** text box, enter a directory and name for the query results file.
-OR-
Click the ellipses (...) button to browse the file system and select a directory and file name.
4. Select one of the **Delimiter** options to specify how to separate entries in the query results file. If you want to use a delimiter other than a comma or a tab, select **Custom** and enter a delimiter in the text box.
5. If you want the output file to include the name of the query and the names of the columns, select the **Print Headers** check box.
6. Click the **Save & Close** button to create the file.

Command line

You can use the **wrunquery** command to run any query and either display the results to standard output or save the results in a file. To use the **wrunquery** command to run the Win-machines query and save the results in a file, enter the following:

```
wrunquery -n -h amon -f /tmp/query.txt -d ";" Win-machines
```

where:

-n Omits headers from the results file.

-h amon
Specifies **amon** as the name of the managed node on which to store the file.

-f /tmp/query.txt
Specifies **/tmp/query.txt** as the location and name of the query results file.

-d “;” Specifies a semi-colon as the delimiter.

Win-machines
Specifies the name of the query to be run.

See the *Tivoli Management Framework: Reference Manual* for more information about the **wrunquery** command.

Defining subscribers with queries

In addition to using queries to find out about your Tivoli environment, you can also use queries to return a list of clients in your environment that meet certain criteria. For example, you can run a query to select targets for an inventory profile distribution. When you run the query, it selects the list of the clients in the policy region that meet the query criteria.

Inventory provides you with several pre-defined queries that you can run to identify subscribers for an inventory profile. These queries belong to the SUBSCRIPTION_QUERY query library. For information about installing subscription queries, see the *Planning and Installation Guide*. For a list and description of available subscription queries, see the chapter on inventory and subscription queries in the *Database Schema Reference*.

The following table provides the authorization roles required to perform these tasks:

Operation	Context	Role
Select targets for an inventory profile distribution	inventory profile	senior or super

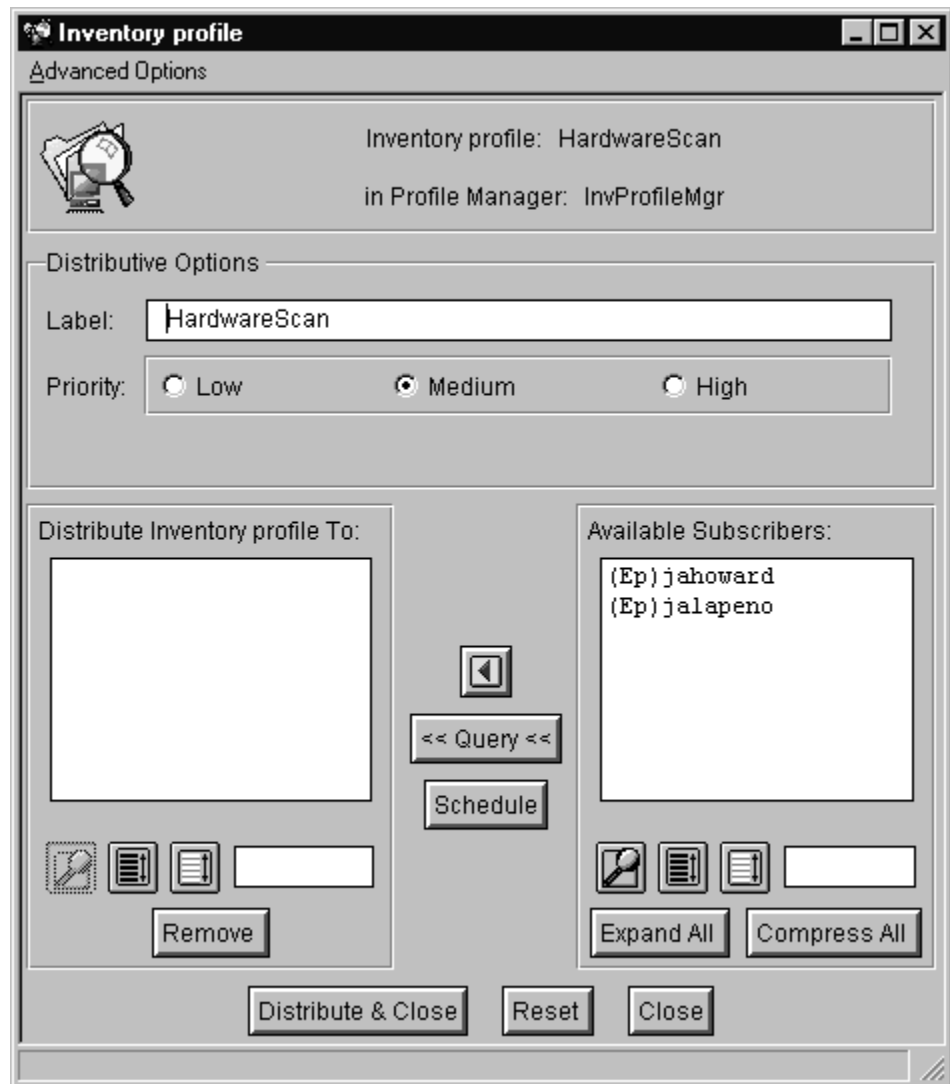
You can select targets from either the Tivoli desktop or the command line.

Desktop

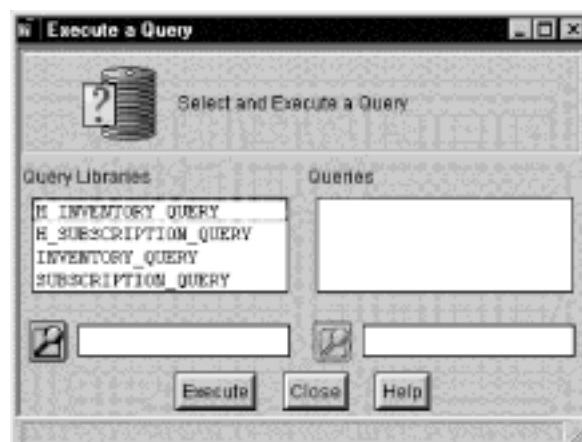
If you run a query from the Inventory Profile window, the query must include the TME_OBJECT_ID and TME_OBJECT_LABEL columns in the **Chosen Columns** list.

Complete the following steps to define subscribers for a profile by running a query from the Inventory Profile window:

1. From the pop-up menu of an inventory profile, select **Distribute**. The Inventory Profile window is displayed.



2. Click **Query**. The Execute a Query window is displayed.



3. In the **Query Libraries** scrolling list, double-click the query library that contains the query you want to run. The **Query Libraries** scrolling list shows a list of all the query libraries in the policy region. The query library you select populates the **Queries** scrolling list.
4. In the **Queries** scrolling list, select a query.
5. Click **Execute**. The Execute a Query window closes and the Inventory Profile window is displayed. The query moves the subscribers in the policy region that meet the query criteria to the **Distribute Inventory Profile To** scrolling list.

Command line

The **wruninvquery** command runs a query and returns the results in an IDL format that you can use with scripts to populate subscription lists. A query that you run with this command must include both the **TME_OBJECT_ID** and **TME_OBJECT_LABEL** columns. In contrast, the **wrunquery** command can be used with any query, and the results are sent to standard output or to a file.

To run the Win-machines query, enter the following:

```
wruninvquery -l Win-machines
```

where:

- l Lists the names of the machines that match the query criteria instead of their object IDs.

Win-machines

Specifies Win-machines as the query to run.

See the *Tivoli Management Framework: Reference Manual* for more information about the **wruninvquery** and **wrunquery** commands.

Querying for information about one client

In addition to using queries to gather information about the Tivoli environment or to generate lists of targets, you can use queries to access inventory information about one client in the Tivoli environment. A query will return the inventory scan results for the specified machine. Any query you use to return information about one client must include the columns **TME_OBJECT_ID** and **TME_OBJECT_LABEL**.

Note: This procedure does not work with pervasive devices or other resource group targets.

The following table provides the authorization roles required to run a query from the Profile Manager window:

Operation	Context	Role
Run a query about one client	The policy region that contains the query library of the applicable query	RIM_view , Query_view , Query_execute , Query_edit , senior , or super

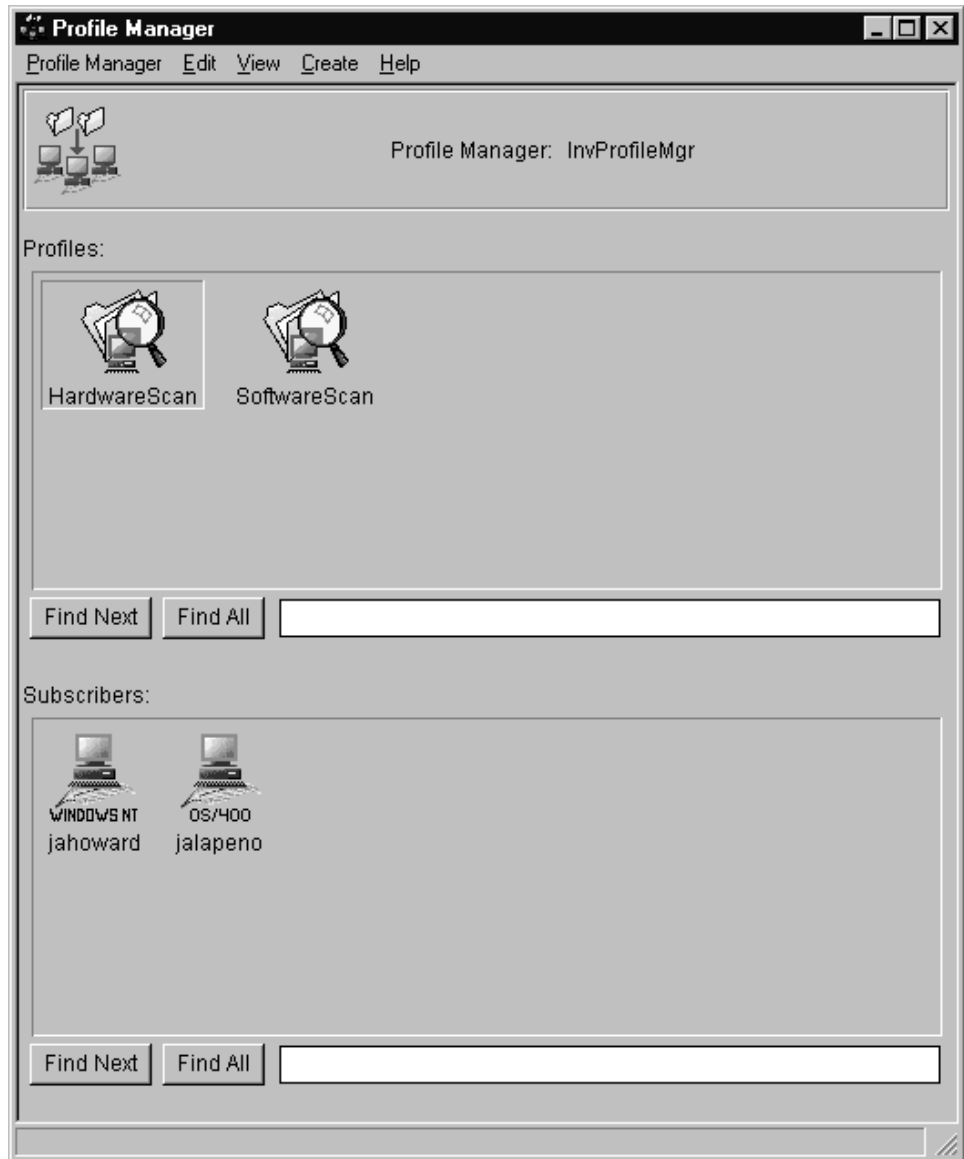
You can retrieve inventory information for a client from either the Tivoli desktop or the command line.

Desktop

Complete the following steps to display information from the icon menu of an endpoint (this procedure does not work with pervasive devices or other resource group targets):

1. From the Profile Manager window, right-click the icon of the endpoint that you want to query and select **Execute Query**.

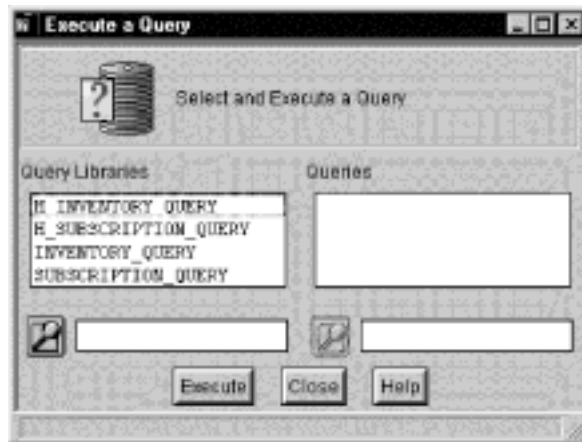
Note: Endpoint icons are visible only from within the window of the profile manager to which they are subscribed.



2. In the Execute Query window, do the following:
 - a. In the **Query Libraries** scrolling list, double-click the query library that contains the query you want to run. The **Query Libraries** scrolling list shows a list of all the query libraries in the policy region. The query library you select populates the **Queries** scrolling list.

Note: Queries that do not contain a TME_OBJECT_LABEL column, such as pervasive queries or the CHECK_PACKAGES query, will not work.

See the *Database Schema Reference* for more information.



- b. In the **Queries** scrolling list, select a query.
- c. Click **Execute**. Inventory runs the query and displays results for only the selected machine.

Command line

Use the **wqueryinv** command to return inventory information about a particular client. The following example runs the query **PARTITION_QUERY** on the managed node **tacoma** and saves the results file in the **x1.txt** file in the **/tmp** directory on the host **amon**:

```
wqueryinv -n -d ";" -h amon -f /tmp/x1.txt \  
-q PARTITION_QUERY -m tacoma
```

where:

- n** Specifies that headers should not be included in the query results file.
- d “;”** Specifies that a semi-colon will be used to separate entries in the query results file.
- h amon**
Specifies the name of the host where the query results file will be saved.
- f /tmp/x1.txt**
Specifies the directory and name of the query results file.
- q PARTITION_QUERY**
Specifies the name of the query to be executed.
- m tacoma**
Specifies the Tivoli object label of the machine that you want to retrieve inventory information about and that it is a managed node.

See “wqueryinv” on page 182 for more information about the **wqueryinv** command.

Using queries in interconnected Tivoli regions

This section explains how to query interconnected Tivoli regions and display the data for each Tivoli region separately. For example, some Inventory users collect data for multiple internal or external customers. If the systems for each customer are in separate Tivoli regions, you can use the following procedure to view each customer’s data separately.

Note: A RIM connection that points to the same database must be available in all involved Tivoli regions.

To view the data from each Tivoli region separately, you create a query that separates the data using the object ID (OID). An OID is a three-part identifier of the form *n.n.n*, for example 1264987995.1.326. The OID is made up of the following values:

- The region number. Objects in Tivoli Management Framework, Version 3.0, and above have ten-digit region numbers (1264987995 in the example above). Region numbers are generated when the Tivoli management region server (Tivoli server) is installed. An algorithm is used to generate a region number for each Tivoli region. In most cases, the number generated is unique. However, due to the random element in the region number generation, there is a small possibility that two Tivoli regions could generate the same region number. If this happens, one of the regions must be reinstalled to connect to the region with the same number.
- The object dispatcher (oserv) number. Sometimes referred to as the host number, dispatcher numbers are assigned in the order managed nodes are installed. The Tivoli server will be dispatcher 1, the next managed node installed will be 2, and so on.
- The object number. This is a number for the object itself.

When a managed node or endpoint is created in a given Tivoli region, the region number of the managed node or endpoint matches that of the Tivoli server.

To find which systems are in a specific Tivoli region, edit the INVENTORY_HWARE query to select only the columns TME_OBJECT_LABEL, TME_OBJECT_ID, and COMPUTER_SYS_ID. In the **Additional Clauses** text box, enter the following clause:

```
order by TME_OBJECT_ID
```

When you run this query, it returns each grouping of endpoints by Tivoli region.

To query specific data for a Tivoli region, edit the appropriate query by adding the following where clause:

```
TME_OBJECT_ID like 'nnnnnnnnnn%'
```

where *nnnnnnnnnn* is the first 10 digits of the OID. To query for data in multiple Tivoli regions, add a where clause for each OID.

All installed hardware and software information for a given system is tied to a computer system ID. If you want to view information about a specific endpoint, add the following where clause:

```
COMPUTER_SYS_ID = 'computer_sys_id'
```

where *computer_sys_id* is the COMPUTER_SYS_ID value of the appropriate endpoint.

Appendix A. Authorization roles

This appendix contains tables that describe the activities and roles you need to perform activities using Inventory.

Inventory roles

To perform activities using Inventory, you must be a Tivoli administrator. Depending on the activities you perform, you must have one or more of the following roles in either the Tivoli management region (Tivoli region) or in the policy region in which the operation is performed:

- **Inventory_end_user**
- **Inventory_view**
- **Inventory_query**
- **Inventory_scan**
- **Inventory_edit**
- **Query_view**
- **Query_edit**
- **Query_execute**
- **RIM_view**
- **RIM_update**
- **user**
- **admin**
- **senior**
- **super**
- **Install_product**
- **Install_client**
- **restore**
- **backup**

Inventory operations

The following table lists the authorization roles required to perform Inventory-specific tasks:

Table 5. Authorization Roles for Inventory Tasks

Operation	Required Role
Create an inventory profile	senior or super
Customize or edit an inventory profile	Inventory_edit , admin , senior , or super
View an inventory profile	Inventory_view , user , admin , senior , or super
Clone an inventory profile	senior or super
Delete an inventory profile	senior or super
Set inventory profile subscribers	admin , senior , or super
Distribute an inventory profile	Inventory_scan or admin

Table 5. Authorization Roles for Inventory Tasks (continued)

Operation	Required Role
View inventory data in the configuration repository	Query_view, RIM_view, user, admin, senior, or super
Use Inventory administrator Web pages	RIM_view
Add, delete, or edit signatures	RIM_view, RIM_update, senior, or super

Query operations

The following table lists the authorization roles required to perform query operations:

Table 6. Authorization Roles for Query Operations

Operation	Required Role
Create a query library	senior or super
Create a query	Query_edit, admin, senior, or super
Edit a query	Query_edit, admin, senior, or super
View a query	Query_execute, Query_edit, Query_view, user, admin, senior, or super
Execute a query	Query_edit, Query_execute, admin, senior, or super

Appendix B. Commands

This appendix lists and documents, in alphabetical order, the commands related to Inventory. The commands for the query facility and the relational database management system (RDBMS) interface module (RIM) are documented in the *Tivoli Management Framework: Reference Manual*.

Most Tivoli commands begin with a **w**, and often vowels are omitted to shorten the name of a command. Commands are usually developed using the **w+verb+object** syntax, which matches the way you might think of the action. For example, to set the global properties of an inventory profile, use the **wsetinvglobal** command. To view the global properties, use the **wgetinvglobal** command.

Using Tivoli commands

It is often necessary or convenient to perform a Tivoli operation from the command line interface rather (CLI) than from the Tivoli desktop. For example, you might prefer to use the command line interface for any of the following reasons:

- You do not have access to a Tivoli desktop.
- You want to group several Tivoli commands in a shell script or batch file.
- You prefer to run a command from within a script that performs multiple operations.

Command line syntax

The reference pages in this appendix use the following special characters to define the command syntax:

- [] Identifies optional options. Options not enclosed in brackets are required.
- ... Indicates that you can specify multiple values for the previous option.
- | Indicates mutually exclusive information. You can use either the option to the left of the separator or the option to the right of the separator. You cannot use both options in a single use of the command.
- { } Delimits a set of mutually exclusive options when one of the options is required. If the options are optional, they are enclosed in brackets ([]).

For example, consider the following command syntax:

wruninvquery [-i] [-T *idl-type*] [-l | -t] *query_name* [*input*]...

The ellipses (...) following the *input* option indicate that you can specify multiple query inputs. Also, if you choose to specify **-l** or **-t**, you can specify only one. These options are delimited by the **|** (logical or), indicating that they are mutually exclusive. You must specify the *query_name* option; all other options are optional, as denoted by the brackets ([]).

Examples of command syntax

If an example of a command is too long to fit within the text margins of this document, a backslash (\) is used to break the line. This indicates that you must not break the line when you actually enter the command.

Object references

When an object is referenced from a command, the reference is not the absolute object reference that is used in programming. Instead, a user-friendly name is used. This user-friendly name derives from a name given to the object by the user of the application, such as when a policy region is created.

There are two different forms of names that can be used with CLI commands:

- Registered names
- Object paths

Tivoli commands support both naming schemes. Sometimes it might be more convenient to use one form over the other. If you receive an error message indicating that a resource cannot be found, try a different naming convention.

Registered names

The Tivoli name registry contains registered names. A *registered name* is the name by which a resource instance is registered with the name registry when it is created. Every resource has a name and is of some particular type. For example, a printer called lp01 has a name lp01 and is of type printer.

The syntax for specifying a resource using the registered name facility is *@type:name*, where *type* is the resource type and *name* is the particular instance of that resource on which you want to perform some operation. Some examples of registered names used as options for the **wls** and **wmv** commands are as follows:

```
wls @PolicyRegion:Servers
wmv @ManagedNode:ayers-rock @PolicyRegion:Servers
```

The name registry does not allow two resources of the same type to have the same name within a single Tivoli management region (Tivoli region). However, it is possible for resource names to be duplicated within two or more interconnected Tivoli regions. If you attempt to perform an action on a resource with a duplicated name, an error message is returned, and the action is not performed. To avoid this situation, you should either rename one of the resources or differentiate between the resources by appending a region name to the resource name, as follows:

```
wls @ManagedNode:moria#moria-Region
```

For more information about the **wls** and **wmv** commands, see the *Tivoli Management Framework: Reference Manual*.

Object paths

Object paths are similar to path names in file systems and can be relative or absolute. An absolute path is one that starts with a slash (/) character. A relative path can start with any character, including the special path components '.' and '..'.

The syntax for specifying a resource using an object path as a name is */distinguished/parent/[type:]name*, where *distinguished* is a resource type, *parent* is the start of the object path name, *type* is used to further identify a resource, and *name* is the particular instance on which you want to perform some operation. You

often use the optional type qualifier when you need to name a particular resource that has the same name as some other resource of a different type.

Some examples of object path names used as options for the **wls** and **wmv** commands are as follows:

```
wls /Regions/Servers
wmv ../Servers/ayers-rock /Regions/Servers
```

Inventory commands

Inventory provides a command line interface for operations that you can perform from the desktop. Inventory includes the following commands:

Table 7. Inventory Commands

Command	Purpose	See Page
wcancelscan	Cancels an inventory scan.	108
wcollect	Returns information about the current configuration of a collector, starts and stops a collector, resets collection routes that are cached at the collector, and configures options for a collector.	110
wcrtinvcb	Creates an instance of the inventory callback object. (You can create only one instance per Tivoli region.)	116
wcrtinvdh	Creates an instance of the inventory data handler object. (You can create only one instance per Tivoli region.)	117
wcstat	Returns status information about a collector, returns information about a collection table of contents (CTOC), and returns the contents of the queues of a collector.	120
wdistinv	Distributes an inventory profile.	122
wepscan	Initiates scans from endpoints and isolated machines.	127
wgetinvdh	Returns configuration information about the inventory data handler.	131
wgetinvglobal	Lists the global properties that are set for an inventory profile.	133
wgetinvpcfiles	Lists information about files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of PC endpoints.	136
wgetinvpchw	Lists the options that are set for a hardware scan of a PC endpoint.	138
wgetinvpcsw	Lists the options for PC software scans that are configured for an inventory profile.	140
wgetinvpvdconfig	Returns a value that indicates whether an inventory profile is set to collect configuration data for pervasive devices.	143
wgetinvpvdhw	Returns a value that indicates whether hardware scanning of pervasive devices is enabled.	144
wgetinvpvdsw	Returns a value that indicates whether software scanning of pervasive devices is enabled.	145
wgetinvswd	Returns a value that indicates whether Inventory has been enabled to integrate with Software Distribution.	146
wgetinvunixfiles	Lists information about files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of UNIX and OS/400 endpoints.	147

Table 7. Inventory Commands (continued)

Command	Purpose	See Page
wgetinvunixhw	Lists the options that are set for hardware scans of UNIX and OS/400 endpoints.	149
wgetinvunixsw	Lists the options for UNIX and OS/400 software scans that are configured for an inventory profile.	151
wgetscanstat	Returns information about current inventory scans.	154
winvdeps	Enables and disables the Common Inventory Technology and Tivoli License Manager dependencies.	156
winvfilter	Modifies files in the custom filter list stored in the configuration repository.	157
winviso	Installs files on an endpoint that you can use to scan disconnected systems	159
winvmgr	Creates, changes, deletes, and lists the default values of MDist 2 distribution options.	161
winvmigrate	Imports the new IBM signatures and migrates the signature packages.	166
winvpackage	Logically groups two or more signatures so that you can use them together to identify software.	167
winvrnode	Removes from the configuration repository all scanned information about an endpoint or pervasive device.	170
winvsig	Modifies the list of signatures in the configuration repository.	172
winvupdatecsid	Resolves the data for duplicate endpoints in the configuration repository.	175
wloadiso	Uploads data from isolated scans.	178
wmvinvcb	Moves the inventory callback object to the managed node that you specify.	180
wmvinvdh	Moves the inventory data handler to the managed node that you specify.	181
wqueryinv	Queries the configuration repository for information about an endpoint.	182
wsetinvdh	Changes the settings of the inventory data handler.	184
wsetinvglobal	Sets the global properties for an inventory profile.	187
wsetinvpcfiles	Specifies options for files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of PC endpoints.	192
wsetinvpchw	Sets the options for hardware scans of PC endpoints.	196
wsetinvpcsw	Sets the options for software scans of PC endpoints.	199
wsetinvpvdconfig	Enables or disables scans of pervasive devices for configuration information.	203
wsetinvpvdhw	Enables or disables hardware scans of pervasive devices.	204
wsetinvpvdsw	Enables or disables software scans of pervasive devices.	205
wsetinvswd	Controls whether Inventory is enabled to integrate with Software Distribution	206
wsetinvunixfiles	Specifies options for files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of UNIX and OS/400 endpoints.	207

Table 7. Inventory Commands (continued)

Command	Purpose	See Page
wsetinvunixhw	Sets the options for hardware scans of UNIX and OS/400 endpoints.	211
wsetinvunixsw	Sets the options for software scans of UNIX and OS/400 endpoints.	214
wwaitscan	Identifies the completion of a scan.	219

Command syntax

This section lists Inventory commands, with syntax and descriptions of their functions. You can also access these listings by using the **man** command on UNIX systems.

wcancelscan

Cancels an inventory scan.

Syntax

```
wcancelscan -a [-v]
```

```
wcancelscan -i scan_ID [-i scan_ID]... [-v]
```

Description

The **wcancelscan** command cancels an active inventory scan. This command can cancel a scan at the following points:

- During the profile distribution, before the profile reaches the endpoint
- For scans of pervasive devices, at the Web Gateway component before the job is processed
- As the data travels through the collector hierarchy to the inventory data handler through SCS
- At the inventory data handler

If scan data has been passed from the inventory data handler to a RIM object, that scan data cannot be cancelled.

During a scan of multiple targets, the scan data for each target reaches the inventory data handler at different times. Therefore, it is possible that the **wcancelscan** command could cancel the scan of one target but not another.

See “Cancelling scans” on page 301 for information about troubleshooting scan cancellations.

Options

- | | |
|-------------------|--|
| -a | Cancels all outstanding scans. |
| -i scan_ID | Specifies the ID of the scan that you want to cancel. You can view IDs for current scans by running the wgetscanstat command. |
| -v | Displays messages that SCS generates for the scan cancellation. |

Authorization

super, and senior

Examples

The following example cancels scans with IDs 100 and 101.

```
wcancelscan -i 100 -i 101
```

The following example cancels all outstanding scans and displays information about the cancelled scans.

```
wcancelscan -a -v
```

The output from this command is as follows:

```
Starting to cancel Inventory profile default with scan ID 14.  
The cancel operation sent to Inventory status collector is complete.  
The cancel operation sent to MDistII manager is complete.  
The cancel operation sent to Scalable Collection Service is complete.  
The cancel operation sent to Inventory data handler is complete.
```

See Also

wgetscanstat

wcollect

Returns information about the configuration of a collector, starts and stops a collector, resets collection routes that are cached in a collector, and configures options for a collector.

Syntax

wcollect [*options*] *collector*

where *collector* is of the form **@ManagedNode:collector_name**, **@Gateway:collector_name**, **@InvDataHandler:inv_data_handler**, or **@InvDataHandler:tlm_data_handler** (*collector_name* is the name of the managed node or gateway).

The *options* are from one of the following categories:

- Active-collector options:
 - [-s | -h {immediate | graceful}]**
- Collector-configuration options:
 - [-a full_dump_queue_time]**
 - [-b checkpoint_mode]**
 - [-c depot_chunk_size]**
 - [-d {0 | 1 | 2 | 3}]**
 - [-e retry_delay_time]**
 - [-f {true | false}]**
 - [-g debug_log_size]**
 - [-i thread_idle_down_time]**
 - [-l runtime_location]**
 - [-m max_input_retries]**
 - [-o max_output_threads]**
 - [-p thread_sleep_time]**
 - [-r]**
 - [-t max_input_threads]**
 - [-x offlinks_range_to_prohibit_collection]**
 - [-z depot_size]**
 - [-wthreshold]**
- Data handler configuration option
 - [-nthreshold]**

Description

The **wcollect** command has four main functions.

1. You can obtain information about the configuration of a collector by running the command and specifying the collector for which you want information.
2. You can use the **wcollect** command to stop or start an existing collector or to delete cached routing information stored in the collector. Options that allow you do this are referred to as active-collector options.
3. You can use the **wcollect** command to configure collector attributes. Options that allow you to do this are referred to as collector-configuration options.

4. You can configure the maximum number of entries that can be added to the data handler input and output queue.

Options

The options can be divided into options for active collectors and options for collector configuration.

Active collectors

-s Starts the collector after it has been halted.

-h option

Halts the collector. Use one of the following options:

- **graceful**—The collector stops after completing any remaining active collections.
- **immediate**—The collector stops without waiting for active collections to finish processing.

Collector configuration

-a full_dump_queue_time

Specifies the time interval in minutes between a full dump of the queues when the checkpoint mode is incremental. The collector needs to do a full dump of the queues in order to compact the incremental changes stored to the file system and to reduce the disk space used.

The default value is 15 minutes. This parameter is ignored if the checkpoint mode is full or none.

-b checkpoint_mode

Specifies the checkpoint mode.

The following values are allowed:

None Disables the checkpoint.

Full Uses the legacy checkpoint mode. Dumps the entire queue to the file system, when a new checkpoint is requested.

Incremental

Writes only the differences to the file system instead of the entire queue, when a checkpoint is requested.

The default value is full.

-c depot_chunk_size

Specifies the size of the transmission chunk in kilobytes (KB). When a downstream collector sends data to the next collector, it sends the data in separate units called transmission chunks. The default chunk size is 1024 KB.

This value should be large enough to keep the transfer of data efficient, but small enough to make most efficient use of the available bandwidth.

-d option

Specifies the level of debugging information to log in the SCS log file (mcollect.log). Use one of the following options:

0 Turns off logging.

1 Logs only fatal errors.

2 Logs fatal errors and warning messages.

3 Logs all debugging messages.

The default value is 1. The SCS log file resides in the \$DBDIR/mcollect directory of each collector. On the inventory data handler, mcollect.log resides in the \$DBDIR/inventory/data_handler directory.

-e *retry_delay_time*

Sets the time in seconds to wait before trying again to process an input or output request to the collector. The default value is 1 second. You can set the maximum number of retry attempts by using **-m** *max_input_retries*.

-f *option*

Specifies whether information about completed CTOCs should be written to the log file. In any stage of the collection, CTOCs can be tracked using the **wcstat** command. Use one of the following options:

true Information about completed CTOCs is written to the log file. This is the default option.

false Information about completed CTOCs is not written to the log file.

-g *debug_log_size*

Specifies the maximum size in megabytes (MB) of the collector log file mcollect.log. When the file reaches the maximum size, it discards 90 percent of its contents and keeps the most recent 10 percent. By default, the maximum size is 1 MB.

-i *thread_idle_down_time*

Specifies the number of seconds that a thread can be idle before it is shut down. The default idle time is 60 seconds.

-l *runtime_location*

Specifies the location of the run-time directory for the collector. The run-time directory contains the depot and run-time files (*.dat and *.log). This directory must reside on a stable disk with an amount of free space large enough to ensure persistent storage of collections. The run-time directory should not be a temporary directory. You must also provide read-write privileges for the **tmersrvd** or **nobody** account (the Tivoli unprivileged account) in this directory. By default, the run-time directory is at \$DBDIR/mcollect on each collector, and the depot is at \$DBDIR/mcollect/depot.

Note: After you specify a new run-time directory, you must stop the collector, and then move the depot and any data from the old run-time directory to the new directory.

-m *max_input_retries*

Specifies the maximum number of attempts to process a collection request from a downstream collector. The default value is 10. You can set the time to wait before trying again to process a request by using **-e** *retry_delay_time*.

-o *max_output_threads*

Specifies the maximum number of output threads that the collector can process concurrently. For collectors, the default value is 5. For the inventory data handler, the default value is 1. This number should be large enough to maximize collector efficiency, but small enough to avoid excessive consumption of system resources. It is strongly recommended that this value match the total number of RDBMS connections set for all

RIM objects used by the inventory data handler. For more information about setting RDBMS connections for RIM objects, see “Configuring RIM objects” on page 19.

-p *thread_sleep_time*

Specifies the number of seconds that a thread *sleeps* (waits) if system or network limitations have been reached. The thread attempts to resume the collection process after this period is completed. The default value is 5 seconds.

-r

Removes locally cached collection routes on each collector. After you run the **wcollect** command with this option, when the collector processes the next collection request, the collector loads new routing information from the collection manager.

Run the command with this option on all affected collectors after you make any modifications to the collection hierarchy using the **wrpt** command. Doing this synchronizes the routing information stored on those collectors.

-t *max_input_threads*

Specifies the maximum number of input threads that the collector can process concurrently. This number should be large enough to maximize collector efficiency, but small enough to avoid excessive consumption of system resources. The default value is 5.

-x *offlinks_range_to_prohibit_collection*

Specifies the range of offlinks. Offlinks are the object dispatcher numbers of collectors from which the specified collector must not collect data. In other words, you use this option to turn off the links to the collectors whose object dispatcher numbers you enter.

To get the object dispatcher number of a collector, use the **odadmin** command and **odlist** option. For more information, see the *Tivoli Management Framework: Reference Manual*.

You must enclose the range of offlinks in double quotation marks (" ").

You can list the object dispatcher numbers in ascending numeric order, separated by commas, as shown in the following example:

```
wcollect -x "4,5,6,7" collector
```

Or, you can use a dash to indicate a range of object dispatcher numbers:

```
wcollect -x "4-7" collector
```

By setting this option, you can control network traffic between downstream collectors. For example, you can turn off links to some collectors during busy hours, and then restore the links when network traffic is light. This option does not affect data collection between endpoints or the Web Gateway component and the first gateway collector. This option does not affect data collection between endpoints and the first gateway collector. For scans of pervasive devices, this option does not affect data collection between the Web Gateway component and the first gateway collector.

To turn on all the links that you have previously turned off with this option, specify a null string (" ") as the value of this option.

By default, no object dispatcher numbers are specified.

-z *depot_size*

Specifies the size of the depot directory in MB. By default, this value is set to 40 MB. You can enlarge the size of the depot, but you cannot make it

smaller. For example, you can set the depot size to 50 MB. Later, you can set the size to 55 MB, but you cannot set it to 45 MB. The maximum size for the depot is 1 gigabyte (GB).

@Gateway:collector_name

Specifies the name of the collector on which to run this command. Use **@Gateway:collector_name** for collectors that are on gateways.

@ManagedNode:collector_name

Specifies the name of the collector on which to run this command. Use **@ManagedNode:collector_name** for collectors that are on managed nodes that are not configured to be gateways.

@InvDataHandler:inv_data_handler

Specifies the name of the Inventory data handler.

@InvDataHandler:tlm_data_handler

Specifies the name of the Tivoli License Manager data handler.

-wthreshold

Monitors the amount of traffic for the output queue on the gateway and collector comparing the specified value to the number of accumulating retries on the CTOCs in the output queue of the collector. If the comparison threshold is exceeded, it automatically reduces the number of output threads on that collector to 1.

Data handler configuration

-n Specifies the maximum number of entries that can be added to the data handler and Wan entry Point Collector input and output queue. Supported values range from 100 to 10 000 entries.

Authorization

senior

Notes

You can run collector-configuration options at any time, but you must restart the affected collectors for the changes to take effect. You halt and restart collectors using the **wcollect** command and the **-h** and **-s** options.

Examples

The following examples illustrate various operations that you can complete with the **wcollect** command.

Request information about current configuration

The following example returns information about the current configuration for a collector named **aztlan**:

```
wcollect @ManagedNode:aztlan
```

This command returns the following output:

```
Collector: @ManagedNode:aztlan
debug_level = DEBUG (all messages)
debug_log_size = 2 MB
runtime_location = /data/aztlan/aztlan.db/mcollect
depot_size = 40 MB
depot_chunk = 1024 KB
thread_idle_down_time = 60 seconds
thread_sleep_time = 5 seconds
max_input_threads = 5
```



```
max_input_retries = 10
max_output_threads = 5
retry_delay_time = 1 seconds
offlinks =
log_completed_ctoc = true
```

Stop a collector after collections complete

The following command stops a collector on a gateway after processing is complete on all active collections:

```
wcollect -h graceful @Gateway:drodriguez-gateway
```

Configure a collector

The following command specifies the amount of debugging information to log and the location and size of the depot for a collector.

```
wcollect -d 3 -l /tmp/dionicio/depot -z 80 @ManagedNode:aztlan
```

Note: After running this command, you must stop the collector, and then move any data from the old depot directory to the new directory.

Turn off links to a collector

The following example turns off the links from the collector on aztlan to object dispatchers 2, 5, 6, 7, 8, and 11. Therefore, the collector on aztlan cannot collect data from those systems. (This command does not affect data collection between endpoints or the Web Gateway component and the first gateway collector.)

```
wcollect -x "2,5-8,11" @ManagedNode:aztlan
```

Turn on all links to a collector

The following example turns on the links to all systems for which links were previously turned off.

```
wcollect -x "" @ManagedNode:aztlan
```

See Also

wcstat, wrpt (in the *Tivoli Management Framework: Reference Manual*)

wcrtinvcb

Creates the inventory callback object.

Syntax

wcrtinvcb *managed_node*

Description

The inventory callback object receives status messages from MDist 2. When inventory data cannot be collected using the collector hierarchy, MDist 2 sends the data to the inventory callback object, which sends it to the inventory data handler.

When you install Inventory, the inventory callback object is created automatically. This command creates an instance of the inventory callback object. For example, you can use this command to recreate the inventory callback object if it is accidentally deleted. You can create the inventory callback object on any managed node in your Tivoli region on which Inventory is installed. You can create only one inventory callback object per Tivoli region.

Options

managed_node The name of the managed node on which you want to create the inventory callback object.

Authorization

admin

Examples

The following example creates the inventory callback object on managed node fuji.

```
wcrtinvcb fuji
```

wcrtinvdh

Creates an instance of the inventory data handler and sets some options for the inventory data handler.

Syntax

```
wcrtinvdh [-d status_directory] [-n bundle_every_n_minutes]  
[-q bundle_every_n_targets] [-r max_RDBMS_retries] [-s {YES | NO}]  
[-t RDBMS_retry_delay_time] [-u {YES | NO}] managed_node
```

Description

The inventory data handler is the inventory object that receives data from collectors and sends the data to the configuration repository. The inventory data handler is created automatically during installation of Inventory. This command creates an instance of the inventory data handler. For example, you can use this command to recreate the inventory data handler if it is accidentally deleted. This command also enables you to specify the interval at which scan completion notifications are sent and set the number of retries and the timeout period for writing inventory information to the configuration repository.

Inventory must be installed on the managed node on which you create the inventory data handler. You can have only one instance of the inventory data handler object in a Tivoli region.

The **wcrtinvdh** command creates the inventory data handler instance @InvDataHandler:inv_data_handler in the Tivoli object database on the managed node that you specify.

To specify the number of output threads for the inventory data handler to write to RIM objects, use the **wcollect** command and the **-o** option. It is strongly recommended that this value match the total number of RDBMS connections set for all RIM objects used by the inventory data handler. For example, if the inventory data handler uses two RIM objects to connect to the RDBMS, and each RIM object has 5 RDBMS connections, set the output threads for the inventory data handler to 10.

Options

-d *status_directory*

Specifies where the inventory data handler stores status information that can be restored in case of a system failure. By default, the location is **\$DBDIR/inventory/stat_dir** on the managed node where the inventory data handler resides.

-n *bundle_every_n_minutes*

Specifies the interval at which scan completion notifications are sent (when the **-n** option for the **wsetinvglobal** command is set to **BUNDLE**). If you set this number, Inventory sends a notice with a list of the targets on which scans have completed during the specified time period. The default value is 10 minutes.

If no scans complete in the specified time period, no notification is sent.

If you set this option to 0, notification occurs according to the value that you set for the **-q** option of the **wcrtinvdh** or **wsetinvdh** command.

-q *bundle_every_n_targets*

Specifies the maximum number of targets in a bundle (when the **-n** option for the **wsetinvglobal** command is set to **BUNDLE**). A bundle is a group of targets about which status information is sent at one time. Status refers to the success or failure of a scan for each target of a particular inventory profile.

The default value is 10 targets.

If you set both **-q *bundle_every_n_targets*** and **-n *bundle_every_n_minutes*** to 0, no bundling occurs, and you are not notified until the scans on all targets are completed.

If you set both **-q *bundle_every_n_targets*** and **-n *bundle_every_n_minutes*** to a positive value, bundling occurs when either value (the specified number of targets or minutes) is reached.

-r *max_RDBMS_retries*

Specifies the number of times the inventory data handler tries to write data to a RIM object. After the maximum number of retries is reached, a failure notification is sent.

The default value is 5.

-s {YES | NO}

Specifies whether the inventory data handler stores status information that can be restored in case of a system failure. Use one of the following options:

YES Status information is stored. This is the default option.

NO Status information is not stored.

-t *RDBMS_retry_delay_time*

Specifies a value from which to calculate the timeout period in seconds between retries of writes to a RIM object. This timeout period works according to the algorithm *timeout*retry_count*. For example, on the first retry, with a timeout value of 30 seconds, the algorithm sets the timer to 30 * 1 or 30 seconds. On the second retry, the timer sets to 30 * 2 or 1 minute.

The default value is 30 seconds.

-u {YES | NO}

Specifies whether the inventory data handler sends a notice to the Inventory notice group when an unsolicited update of scan data occurs. An unsolicited update is an update that is not initiated by distributing a scan to an endpoint remotely from another system. Examples include an endpoint-initiated scan using **wepscan**, an inventory scan run from the Web Interface, or an update of configuration repository data by an application or component other than Inventory. Use one of the following options:

YES A notice is sent.

NO A notice is not sent. This is the default option.

managed_node

Specifies the managed node that will act as the inventory data handler.

Authorization

admin, senior, or super

Examples

The following example creates the inventory data handler on the managed node `mckinley`, sets the time interval for notification to 20 minutes (for inventory profiles with notification set to **BUNDLE**—see the **wsetinvglobal** command), and sets the number of retries for RIM failures to 3.

```
wcrtinvdh -n 20 -r 3 mckinley
```

See Also

wcollect, **wgetinvdh**, **wgetinvglobal**, **wmvinvdh**, **wsetinvdh**, **wsetinvglobal**

wcstat

Returns status information about a collector, returns information about a CTOC, and returns the contents of the queues of a collector.

Syntax

wcstat *collector*

wcstat -v *ctoc_id collector*

wcstat -q [*ioecd*] *collector*

where *collector* is of the form **@ManagedNode:collector_name**, **@Gateway:collector_name**, or **@InvDataHandler:inv_data_handler**

Description

With the **wcstat** command, you can complete the following tasks:

- Retrieve status information about a specified collector.
- Retrieve information about a specified CTOC on a collector.
- Retrieve the contents of any or all of a collector's queues.

Options

collector

Specifies the name of the collector for which you want to run the **wcstat** command. You must use one of the following formats:

- **@Gateway:collector_name** where *collector_name* is the name of a gateway that is a collector
- **@ManagedNode:collector_name** where *collector_name* is the name of a managed node that is a collector
- **@InvDataHandler:inv_data_handler** where *inv_data_handler* is the name of the Inventory Data handler

-v *ctoc_id*

Specifies the CTOC for which you want status information returned.

-q *option*

Specifies the type of queue to be returned. The following options return the contents of the specified queue. You can enter one or more of these options in any combination.

- **i**—input queue
- **o**—output queue
- **e**—error queue
- **c**—completed queue
- **d**—deferred queue

Authorization

senior

Examples

The following examples illustrate operations that you can complete with the **wcstat** command.

Return the contents of a completed queue

The following example returns the contents of the completed queue for the collector aztlan:

```
wcstat -q c @Gateway:aztlan-gateway
```

This command returns the following output:

```
CTOC ID: c1737279837112254
CTOC Properties:
  PRIORITY: 1
  COLL_STATUS: TRUE
  SOURCE_NAME: drodriguez2
  SOURCE_OID: 2112331601.2.19
  SOURCE_METHOD: mc_get_data
  DEST_OID: 2112331601.1.675
  INV_DDC::InvDataHandler
  DEST_METHOD: mc_request_collection
  DATAPACK: 2129
Client Properties:
  scan_id: 2147483647
Collection Status: CTOC_DONE
#Retries: 0
```

Return status information of a CTOC

The following example returns information about the specified CTOC:

```
wcstat -v c1737279837112254 @ManagedNode:calypso
```

This command returns the following output:

```
CTOC ID: c1737279837112254
CTOC Properties:
  PRIORITY: 1
  COLL_STATUS: OK
  SOURCE_OID: 1637823410.2.19
  DEST_OID: 1637823410.1.552#MCFTP::Server#
  DEST_METHOD: callback_method
  DATAPACK: 33637364
Client Properties:
  MC_DEST_DIR: /tmp
Collection Status: QUEUED_OUTPUT
#Retries: 0
```

See Also

wcollect

wdistinv

Distributes an inventory profile.

Syntax

```
wdistinv -I mdist2_keyword=value... @InventoryConfig:profile_name
```

```
wdistinv -T subscribers_file [-T subscribers_file]... @InventoryConfig:profile_name
```

```
wdistinv @InventoryConfig:profile_name @Endpoint:endpoint_name ...
```

```
wdistinv @InventoryConfig:profile_name @ResourceGroup:resource_group_name ...
```

Description

The **wdistinv** command enables you to distribute an inventory profile from the command line. You can distribute profiles to an endpoint, profile manager, or resource group. If you do not specify targets for the distribution, the inventory profile is distributed to all subscribers of the profile manager in which the inventory profile resides.

Options

These options apply to scans of endpoints and users. Unless otherwise specified, they do *not* apply to scans of pervasive devices.

-I *mdist2_keyword=value*

Sets a value for the MDist 2 distribution options. These options apply only to the current distribution. These options do not modify the settings of the specified profile. You can specify multiple options by repeating **-I** *mdist2_keyword=value* multiple times. You can use the following keywords and values:

deadline=*mm/dd/yyyy hh:mm*

Specifies the date and time at which a distribution expires (the date that it fails for unavailable target systems, including pervasive devices). If a target has not received a distribution by this time, MDist 2 no longer attempts the distribution. The default value is 72 hours from the time the profile is distributed. You cannot set both this value and the **dist_timeout** value.

dist_timeout=*hours*

Specifies the number of hours after the profile is distributed that the distribution expires (it fails for unavailable target systems, including pervasive devices). For the value *hours* you must specify a positive integer value. The default value is 72 hours. You cannot set both this value and the **deadline** value.

escalate_date_n=*mm/dd/yyyy hh:mm*

Specifies the date on which a reminder message must be sent to mobile or roaming targets (such as laptops) that have not yet received the profile. For the value *n*, you can specify a number, 0 through 9, which enables you to create up to ten messages. For *n*, you must specify the values in ascending numerical order, and you cannot skip a number. For each escalation date, you must also specify an escalation message using **escalate_msg**.

escalate_msg_n="message text"

Specifies a message that must be sent to roaming targets that have not received the profile by the associated escalation date. For the value *n* you can specify a number, 0 through 9, which enables you to create up to ten messages. For *n*, you must specify the values in ascending numerical order and you cannot skip a number. For each escalation message, you must also specify an escalation date using **escalate_date**. Enter a message using the following format:

```
escalate_msg_n="message text"
```

execute_timeout=seconds

Specifies the number of seconds that MDist 2 waits for an endpoint to return scan status after the profile has been distributed. For scans of endpoints, this value includes the time that it takes to run the scan and any custom scripts and make an upcall to SCS to request data collection. If the upcall to SCS fails, this value includes the time that it takes to send data to the inventory callback object. For scans of pervasive devices, the execution timeout is the length of time MDist 2 waits for a job to be created on the Web Gateway component. If you plan to run a scan that will take a long time to complete, for example if you must scan a large number of drives, you would set this value to allow for that time.

force_mandatory=y | n

Controls the way in which mandatory distributions on roaming endpoints are treated once the mandatory date is passed. Specify one of the following options:

- **y**—The mandatory distribution is automatically started as soon as the roaming user connects. This is the default option.
- **n**—The roaming user has the choice of not starting the mandatory distribution. However, the user will not be able to perform any other operations until the mandatory distribution has been performed.

hidden={y | n}

Indicates whether a distribution to a roaming endpoint is hidden. Hidden distributions are not revealed to the roaming user and are automatically distributed to the roaming endpoint. Distributions that are not hidden are visible to the roaming user, and the roaming user can defer the distribution. Use one of the following options:

- **y**—Hides the distribution.
- **n**—Does not hide the distribution.

If you set this option to **y**, you must not set values for **mandatory_date**, **escalate_date**, or **escalate_msg**.

This option performs the same function as the **wsetinvglobal** command and the **-m** option. If this option and the **wsetinvglobal** command and the **-m** option are set to conflicting values, the **wsetinvglobal** command and the **-m** option override this option.

If you set this option in an InventoryConfig profile, its value overrides the hidden keyword value.

inv_ep_debug={1 | 2 | 3}

Enables debugging of scanned endpoints and pervasive devices,

and specifies the level of debugging information to be logged. When this option is enabled, information is collected from each scanned target and saved to a log file. Use one of the following options:

- 1 Logs error messages.
- 2 Logs error and warning messages.
- 3 Logs error and warning messages and debugging information. (Scanner debugging information is not available from NetWare or OS/2 endpoints.)

If you do not specify a value, debugging information for endpoints is not logged.

Log files for each scanned endpoint are collected by the scan and saved on the inventory data handler in the \$DBDIR/inventory/data_handler/logfiles/*scan_ID*/*endpoint_name* directory, where:

- *scan_ID* is the ID of the scan.
- *endpoint_name* is the name of the endpoint for which debugging information was collected.

For scans of pervasive devices, the log file is stored in \$LCFROOT/inv/TWG/INV*scan_ID* on the endpoint that hosts the Web Gateway component. A single log file is created for all of the devices processed on the endpoint.

The name of the log file for both endpoints and pervasive devices is **INV000xx.log**, where *xx* is the scan ID.

is_fail_unavailable = y | n

Specifies if the status of the distribution must be set to failed, if the endpoint is unavailable. When this option is enabled, for example MDist 2 reports the status of the distribution as failed, when the endpoint is powered off. If this option is not enabled, MDist 2 keeps contacting the endpoint until the distribution deadline is reached.

label=description_string

Specifies a description string for the distribution. The default value is the name of the inventory profile.

mandatory_date=mm/dd/yyyy hh:mm

Specifies the date by which the distribution must be made. Distributions to roaming targets can be deferred up to this date. When the date is reached, the profile is automatically distributed to all roaming targets that have not yet accepted it. The default value is 60 hours from the time the profile is distributed.

notify_interval=minutes

Specifies the interval, in minutes, in which each repeater bundles and returns the completed results of the MDist 2 distribution. Use a positive integer. The default value is 1 minute.

priority={h | m | l}

Specifies the priority for distributions to endpoints and pervasive devices. Priority is the order in which distributions are handled by repeaters. You can set priority to one of the following values:

- h** Highest priority.

m Medium priority. This is the default value.

l Low priority.

roam_endpoints={y | n}

Indicates whether the operation defined in the **wdistinv** command supports roaming endpoints. Use one of the following options:

- **y**—The distribution can be transferred to any gateway where the roaming endpoint connects.
- **n**—After the profile is queued at a gateway it cannot be transferred to another gateway.

send_timeout=seconds

Specifies the number of seconds that a repeater waits for a target system to receive a block of data (for example, a repeater distributing an inventory profile to an endpoint). Each time a packet is sent over the network, the timeout for that packet is set to this value. This timeout is used to detect network or endpoint failures. If a timeout occurs, the distribution remains in the repeater's queue and a retry occurs according to the **conn_retry_interval** value set with the Tivoli Management Framework **wmdist** command.

wake_on_lan={y | n}

Indicates whether the operation sends a Wake on LAN message to trigger the start of a system that is not available when an inventory profile is distributed. For scans of pervasive devices, this option starts the endpoint on which the Web Gateway component resides. This option works only on systems that have a network card that is enabled for Wake on LAN. Use one of the following options:

- **y**—Sends a Wake on LAN message.
- **n**—Does not send a Wake on LAN message.

If you set this option in an InventoryConfig profile, its value overrides the **wake_on_lan** keyword value.

-T subscribers_file

Specifies a text file that contains a list of targets for the distribution. This file can list endpoints, profile managers, resource groups for users or pervasive devices, or any combination of these targets. Within this file, you must list the targets in one of the following formats:

- **@Endpoint:endpoint_name**
- **@ProfileManager:profile_manager_name**
- **@ResourceGroup:resource_group_name**

For the *subscribers_file* value, you can list files using either relative or absolute path names. You can specify multiple files by repeating **-T subscribers_file** multiple times. You can use this option alone, or you can specify one or more targets for the distribution using the **@Endpoint:endpoint_name** option.

@InventoryConfig:profile_name

Specifies the profile on which you want to run the **wdistinv** command.

@Endpoint:target

Specifies the endpoint to which you want to distribute the inventory profile.

@ResourceGroup:*resource_group_name*

Specifies the resource group that contains the pervasive devices or users that you want to distribute the inventory profile to.

Authorization

Inventory_scan, senior, or super

Examples

The following example provides a label and sets a mandatory date for the distribution of profile InvProfile:

```
wdistinv -l label="Inventory Distribution" \  
-l mandatory_date=04/06/2001 12:00 @InventoryConfig:InvProfile
```

The output for this command is as follows:

```
Distribution ID: 1929790663.8  
Scan ID: 8
```

The following example specifies a file containing a list of targets and sets the priority for the distribution of profile InvProfile:

```
wdistinv -T /tmp/endpoint.list -l priority=high \  
@InventoryConfig:InvProfile
```

The output for this command is as follows:

```
Distribution ID: 1929790663.8  
Scan ID: 8
```

See Also

wdistrib, wmdist, wrpt (in the *Tivoli Management Framework Reference Manual*)

wepscan

Initiates a scan from an endpoint.

Syntax

wepscan

wepscan -c

wepscan [-d {1 | 2 | 3}] [-s [-l]] [-t *mc_upcall_timeout*]

wepscan -i -n *DAT_file_name* [-d {1 | 2 | 3}] [-g *computer_system_ID*]

wepscan -x {sw | hw | full} [-d {1 | 2 | 3}]

Description

The **wepscan** command enables you to initiate a scan from an endpoint, rather than having to distribute an inventory profile to the endpoint from a remote system. You can specify whether to scan the endpoint, send the scan data to the configuration repository, or both. With the **wepscan** command, you can also generate log files that contain scan data, the contents of the configuration file, and debugging information.

You can also use the **wepscan** command, with the **winviso** and **wloadiso** commands, to run isolated scans. An isolated scan is a scan of a system that is not in your Tivoli region. See “Scanning disconnected systems” on page 56, “winvmgr” on page 161, and “wloadiso” on page 178 for more information.

Each time you distribute an inventory profile to an endpoint, it installs a configuration file, *config.dmp*, on the endpoint. This file contains all the scan options that are set in the inventory profile. When you run the **wepscan** command with no options, it reads the configuration file that was last distributed to the endpoint and then runs the scan according to the options in the configuration file. It then creates a Management Information Format (MIF) file, parses the MIF file, and creates an encoded and compressed data file, *INV_SA.DAT*. The data file is stored on the endpoint.

Before you run **wepscan**, you must first set up your environment to access shared libraries needed by the command. To do this, you must run the *lcf_env* script. Depending on the platform type of the endpoint, this script is one of the following: *lcf_env.bat*, *lcf_env.cmd*, *lcf_env.csh*, or *lcf_env.sh*. This script is located either in the directory in which the endpoint is installed, or in the *dat/1* subdirectory within that directory. On Windows systems, the script is located in the **%SystemRoot%\Tivoli\lcf\number** directory, where *number* is a value based on the number of installations.

Solaris Operating Environment (referred to in the rest of this document as Solaris) users and Linux users must set the *LD_LIBRARY_PATH* environment variable before using **wepscan**. For example, a Bourne or Korn shell user would set the environment value as follows:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.  
export LD_LIBRARY_PATH
```

HP-UX users must set the SHLIB_PATH environment variable before using **wepscan**. For example, a Bourne or Korn shell user would set the environment value as follows:

```
SHLIB_PATH=$SHLIB_PATH:.  
export SHLIB_PATH
```

You must run the **wepscan** command from the \$LCF_BASE_DIR/inv/SCAN directory, where \$LCF_BASE_DIR is the directory in which you installed the endpoint.

Options

- c** Reads the contents of the configuration file (config.dmp) and then prints the contents to a log file named sa_config.log. This log file is saved in the directory from which you ran **wepscan**. The **-c** option does *not* initiate a scan of the endpoint.
- d** Creates the following log files:
 - sa_results.log—This log file contains the scan data. The log file is saved in the directory from which you ran **wepscan**.
 - sa_config.log—This is the same log file that is created using the **-c** option. The log file is saved in the directory from which you ran **wepscan**.
 - INV_SA.LOG—This log file contains debugging information. It is identical to the log file that is created using the **wdistinv** command and **inv_ep_debug** keyword. This log file is saved in the directory from which you ran **wepscan**. When you run **wepscan** using the **-s** option, this log file is sent to the inventory data handler. To avoid sending the file, use the **-l** option described below.

You must specify one of the following options with the **-d** option. These options specify the level of debugging information that is written to INV_SA.LOG:

- 1 Logs error messages.
- 2 Logs error and warning messages.
- 3 Logs error and warning messages and debugging information. (Debugging information is not available from NetWare or OS/2 endpoints.)

You can also create these log files by creating an environment variable on your system named WEPSCAN_DEBUG. Set this environment variable to a value of 1, 2, or 3. These values correspond to the options you specify with the **-d** option. This environment variable is especially helpful when the **wepscan** command is not available but you want to create log files, for example when you initiate a scan using the Web Interface. After you set the WEPSCAN_DEBUG environment variable, the log files are created each time you scan the system using the Web Interface or the **wepscan** command.

- t mc_upcall_timeout** Specifies the length of time, in seconds, that an endpoint collector waits for a down call from a gateway collector before it sends a

.DAT file through the callback object. For regular distributions from the server side, use the **winvmgr** command to set the `mc_upcall_timeout` value.

Note: If you do not use the **-t** option, the default timeout value is 180.

-g *computer_system_ID*

Specifies the value that you want to use as the computer system ID for a disconnected system. You can specify an alphanumeric value of 64 or fewer characters. You must specify a unique value.

Note: The **wepscan** command and **-i** option automatically locate the computer system ID of a disconnected system. If no value is found, a computer system ID is created automatically. During subsequent scans, this same value is used. Use the **-g** option only if you want to override the computer system ID automatically generated by the **wepscan** command. For example, if you reinstall the operating system of a machine, the computer system ID of that machine might be removed and a new computer system ID generated. If scan data already exists in the configuration repository for this machine, this creates the potential for duplicate endpoint records in the configuration repository. You can use this option to ensure that the original computer system ID value is used for this machine.

-x

Runs scans invoked by the upload manager during its processing of Tivoli License Manager requests.

You must specify one of the following options with the **-x** option.

sw Performs a matched software scan using the same catalog of the Tivoli License Manager agent.

hw Performs a hardware scan similar to the one performed by the Tivoli License Manager.

full Performs both a matched software scan and an hardware scan.

-i

Runs an isolated scan. The scan data is saved on the local system in the DAT file specified in the **-n** option.

-n *DAT_file_name*

Specifies the name of the DAT file created using the **-i** option. Use the format *file_name.DAT*. If you are uploading DAT files from multiple disconnected systems into a single depot directory, consider using unique names for the DAT files of each disconnected system to avoid overwriting them when you upload them to the depot directory.

-s

Sends the scan data (the INV_SA.DAT file) to the configuration repository. This option also sends the INV_SA.LOG file to the inventory data handler, where it is stored in the `$DBDIR/inventory/data_handler/logfiles/0/endpoint_name` directory. When used alone, this option does *not* initiate a scan of the endpoint; it only sends the scan data and log file.

- l Does not send the INV_SA.LOG file to the inventory data handler. This option can be used only in conjunction with the -s option.

Authorization

This command is run locally at the endpoint. No Tivoli authorization is required.

Notes

This command does not work on OS/400 systems, OS/2 and Netware endpoints, and endpoints logged into Netware or OS/2 gateways.

Examples

The following example scans an endpoint using the scan options specified in the configuration file on the endpoint, and then stores the scan data on the endpoint:

```
wepscan
```

The following example sends scan data stored on the endpoint to the configuration repository and sends the INV_SA.LOG file to the inventory data handler:

```
wepscan -s
```

The following example prints the contents of the configuration file, config.dmp, to sa_config.log:

```
wepscan -c
```

The following example runs a scan, sets the debugging level to maximum, creates the log files, and sends the scan data and INV_SA.LOG file:

```
wepscan -d 3 -s
```

The following example runs an isolated scan, saves the scan data in the file TQUINN.DAT, and logs error messages:

```
wepscan -i -n TQUINN.DAT -d 1
```

wgetinvdh

Returns configuration information about the inventory data handler.

Syntax

wgetinvdh

wgetinvdh [-a] [-d] [-n] [-q] [-r] [-s] [-t] [-u] [-z *debug_level*]

Description

The **wgetinvdh** command returns information about the inventory data handler, including how notification occurs, the number of retries, and the timeout period for writing data to a RIM object.

You can either run this command with no options or with the **-a** option to return all configuration information about the inventory data handler.

Options

- a** Returns all available information about the inventory data handler.
- d** Returns the value of *status_directory* as set by the **wcrtinvdh** or **wsetinvdh** commands. This directory stores status information so that it can be restored in case of a system failure. The default value of *status_directory* is `$DBDIR/inventory/stat_dir`; however, this default value is not returned by this option. This option only returns data if the value has been changed from the default.
- n** Returns the value of *bundle_every_n_minutes* as set by the **wcrtinvdh** or **wsetinvdh** commands, which specifies the interval in minutes at which scan completion notifications are sent.
- q** Returns the value of *bundle_every_n_targets* as set by the **wcrtinvdh** or **wsetinvdh** commands, which specifies the maximum number of targets in a bundle.

A bundle is a group of targets about which status information is sent to the Inventory notice group at one time. Status refers to the success or failure of a scan for each target of a particular inventory profile. This option as set by the **wcrtinvdh** or **wsetinvdh** commands, in combination with the **-n** option of the **wsetinvglobal** command, configures how status information is sent in bundles.
- r** Returns the value of *max_RDBMS_retries* as set by the **wcrtinvdh** or **wsetinvdh** commands, which specifies the number of times the inventory data handler tries to write data to a RIM object before returning a failure for the target to which the data is associated.
- s** Returns information about whether the inventory data handler stores status information in case of a system failure. One of the following options is returned:
 - YES** Status information is stored.
 - NO** Status information is not stored.
- t** Returns the value of *RDBMS_retry_delay_time* as set by the **wcrtinvdh** or **wsetinvdh** commands, which specifies a value used to calculate the timeout period in seconds between retries of writes to a RIM object. This

timeout period works according to the algorithm *timeout*retry_count*. For example, on the first retry, with a timeout value of 30 seconds, the algorithm sets the timer to 30 * 1 or 30 seconds. On the second retry, the timer sets to 30 * 2 or 1 minute.

- u Returns a value that indicates whether the inventory data handler is configured to send a notice to the Inventory notice group when an unsolicited update of scan data occurs. One of the following options is returned:
 - YES** A notice is sent to the Inventory notice group when an endpoint-initiated scan completes.
 - NO** A notice is not sent to the Inventory notice group when an endpoint-initiated scan completes.
- z Enables or disables the logging activity of the status collector. Returns the value of *debug_level* where *debug_level* can assume one of the following values:
 - 0** Disables the logging activity of the status collector.
 - A value bigger than 0** Enables the logging activity of the status collector.

Authorization

user, admin, senior, super, or Inventory_view

Examples

The following example returns all available information about the inventory data handler (status_directory is set to its default value):

```
wgetinvdh -a
```

This command returns the following output:

```
Save status in the directory:
Send bundled notification every: 10 minutes
Send bundled notification every: 10 targets
Max retries for RDBMS errors:      5
Save status in case of failure: YES
Retry delay time for RDBMS errors: 30 seconds
Send notice for unsolicited scans:NO
Managed Node:                      fuji
```

See Also

wcollect, wcrtinvdh, wmvinvdh, wsetinvdh, wsetinvglobal

wgetinvglobal

Lists the global properties that are set for an inventory profile.

Syntax

wgetinvglobal [-d] [-e] [-f] [-h] [-l] [-m] [-n] [-s] [-t] [-u] [-w] *profile_name*

Description

The **wgetinvglobal** command returns the global properties that are set for an inventory profile. The information that this command returns depends on the options that you specify when you run it.

Run this command with no options to return all global property information for the specified inventory profile.

Options

- | | |
|-----------|---|
| -d | Returns information about the distribution options for the specified inventory profile. The following values can be returned:
CONFIG
Distributes the configuration file only.
ALL Distributes the configuration file and run the scan. |
| -e | Returns the number of seconds that a profile attempts to scan a target before it times out. |
| -f | Returns the name of the log file to which Inventory status information is logged. |
| -h | Returns the name of the managed node to which Inventory status information is logged. |
| -l | Returns one of the following values, which indicates where Inventory logs status information:
NOTICE_GROUP
Sends status information to the Inventory notice group.
LOG_FILE
Logs status information to the log file specified with wsetinvglobal command and the -h and -l options.
TEC Sends status information to the Tivoli Enterprise Console console.
OFF Logs no status. |
| -m | Returns a value that indicates whether a distribution to a mobile or roaming endpoint (such as a laptop) is hidden. Refer to the wsetinvglobal command for more information about hidden distributions. The following values can be returned:
Y The distribution is hidden.
N The distribution is not hidden. |
| -n | Returns information about when notification is sent when a scan completes on each target. The following values can be returned: |

IMMEDIATE

Indicates that notification is sent as the scan on each target completes.

DONE

Indicates that notification is sent only when scans on all targets are complete.

BUNDLE

Indicates that notification is sent periodically, based on the settings for the inventory data handler. Refer to the **wsetinvdh** command for more information about bundling.

-s

Returns a value that indicates whether the specified profile can be distributed to targets that are not subscribers to the profile manager that contains the inventory profile. The following values can be returned:

YES The profile can be distributed to endpoints that are not subscribers to the profile manager that contains the inventory profile.

NO The profile cannot be distributed to non-subscribers.

-t

Returns information about what type of notification is sent for a target depending on the completion status of a scan. The following values can be returned:

NONE

Indicates that notification is not sent.

SUCCESS

Indicates that notification is sent for targets on which scans complete successfully.

FAIL Indicates that notification is sent for targets on which scans failed.

ALL Indicates that notification is sent for all targets.

-u

Returns information about the data options for this profile. The following values can be returned:

DIFFS Update with differences. Only data that has been added or changed since the last scan is stored in the configuration repository.

REPLACE

Replace with current results. Data in the configuration repository is replaced with the data from the scan.

-w

Specifies whether the profile distribution will switch on a system that has a Wake on LAN network card installed and enabled. The following values can be returned:

Y The profile distribution will switch on the system.

N The profile distribution will not switch on the system.

profile_name

Specifies the inventory profile about which you want information. Enter the profile name in the format **@InventoryConfig:profile_name**.

Authorization

user, admin, senior, super, or Inventory_view

Examples

The following example returns all information about the global properties of inventory profile TestProfile:

```
wgetinvglobal @InventoryConfig:TestProfile
```

This command returns the following output:

```
Distribution option: Distribute configuration file and run scan (ALL)
Endpoint timeout:   1800 seconds
Log file pathname:  /usr/logs/TestProfile.log
Log file host:      fuji
Notice location:    NOTICE_GROUP
Distribution shown to mobile users: NO
Notice interval:    DONE
Distribution allowed to non-subscribers: NO
Notice type:        ALL
Data option:        Replace with current results (REPLACE)
Distribution uses wake on LAN technology: YES
```

The following example returns the location to which notification information is sent and the interval at which it is sent:

```
wgetinvglobal -l -n @InventoryConfig:MyProf
```

This command returns the following output:

```
Notice location:    NOTICE_GROUP
Notice interval:    DONE
```

The following example returns a value that indicates whether the profile MyProfile can be distributed to targets that are not subscribers to the profile manager that contains the profile:

```
wgetinvglobal -s @InventoryConfig:MyProfile
```

This command returns the following output:

```
Distribution allowed to non-subscribers: NO
```

See Also

wgetinvpcfiles, wgetinvpchw, wgetinvpcsw, wgetinvunixfiles, wgetinvunixhw, wgetinvunixsw, wsetinvglobal

wgetinvpcfiles

Lists information about files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of PC endpoints.

Syntax

wgetinvpcfiles [-b] [-d] [-f] [-m] [-s] *profile_name*

Description

The **wgetinvpcfiles** command can return the following information about an inventory profile:

- The files or file types that are included or excluded during a software scan.
- The directories that are included or excluded during a software scan.
- Scripts that are run during the profile distribution before or after the scan is performed.
- The custom MIF files that are collected during the profile distribution after the scan completes.

Run **wgetinvpcfiles** with no options to return all the information.

Options

-b	Returns the contents of the script to be run on targets during the profile distribution before the scan is performed.
-d	Returns the list of directories to be included or excluded during the scan.
-f	Returns the list of files or file types to be included or excluded during the scan.
-m	Returns the list of MIF files to be read during the scan.
-s	Returns the contents of the script to be run on targets during the profile distribution after the scan is performed.
<i>profile_name</i>	The name of the inventory profile about which you want information. Enter the profile name in the format @InventoryConfig:profile_name .

Authorization

user, admin, senior, super, or Inventory_view

Examples

The following example returns all information about files and directories to be scanned, scripts to be run, and custom MIF files to be collected by the inventory profile MyProfile:

```
wgetinvpcfiles @InventoryConfig:MyProfile
```

The output from this command is as follows:

```
No Include Directories
Exclude Directories:
    */TMP/
    */TEMP/
Include File Types:
    *.EXE
```

*.DLL
*.COM
*.NLM
No Custom Mif Files
No script to run before scan
No script to run after scan

See Also

**wgetinvglobal, wgetinvpchw, wgetinvpcsw, wgetinvunixfiles, wgetinvunixhw,
wgetinvunixsw, wsetinvpcfiles, wsetinvunixfiles**

wgetinvpchw

Lists the options that are set for the hardware scan of a PC endpoint.

Syntax

wgetinvpchw [-a] [-d] [-s] [-t] [-u] *profile_name*

Description

The **wgetinvpchw** command returns the options for PC hardware scans that are configured for an inventory profile. This command can return the following information about a profile:

- Whether the Tivoli hardware scanner is run during scans of PC endpoints.
- Whether the scan data is sent to the configuration repository or stored on the endpoint to be collected later.
- Which hardware components are scanned for.

Options

- a** Lists all the hardware components that you can scan for, and indicates whether the inventory profile is configured to scan for each component.
- d** Returns a value that indicates whether the DMI scanner is enabled. The following values can be returned:
- YES** The DMI scanner is enabled.
- NO** The DMI scanner is disabled.
- s** Returns a value that indicates whether the data collected during DMI hardware scans of PC systems is sent to the configuration repository. The following values can be returned:
- YES** DMI scan data is sent to the configuration repository after the scan completes.
- NO** DMI scan data is not sent to the configuration repository. It is stored on the endpoint.
- t** Returns a value that indicates whether the Tivoli hardware scanner is enabled. The following values can be returned:
- YES** The Tivoli hardware scanner is enabled.
- NO** The Tivoli hardware scanner is disabled.
- u** Returns a value that indicates whether the data collected during hardware scans of PC systems is sent to the configuration repository. The following values can be returned:
- YES** Scan data is sent to the configuration repository after the scan completes.
- NO** Scan data is not sent to the configuration repository. It is stored on the endpoint.

profile_name

The inventory profile about which you want information. Enter the profile name in the format **@InventoryConfig:profile_name**.

Authorization

user, admin, senior, super, or Inventory_view

Examples

The following example returns all information about the options for PC hardware scans that are configured for inventory profile MyProfile:

```
wgetinvpchw @InventoryConfig:MyProfile
```

This command returns the following output:

```
Run Tivoli Scanner: YES
Update the hardware scan results in the configuration repository: YES
Update the DMI scan results in the configuration repository: YES
Data to scan:
  Processor: YES
  Memory: YES
  MemoryModules: YES
  Operating System: YES
  Storage: YES
  IP Address: YES
  Modem: YES
  Network Adapter: YES
  Partition: YES
  PC System Params: YES
  PCI Device: YES
  Pointing Device: YES
  SMBIOS: YES
  Keyboard: YES
  IPX Address: YES
  Video: YES
  Printer: YES
  USB Device: YES
  Service Info: YES
  Lpar: YES
```

The following example indicates whether inventory profile MyProfile is configured to run the Tivoli hardware scanner:

```
wgetinvpchw -t @InventoryConfig:MyProfile
```

This command returns the following output:

```
Run Tivoli Scanner: YES
```

See Also

wgetinvglobal, wgetinvpcfiles, wgetinvpcsw, wgetinvunixfiles, wgetinvunixhw, wgetinvunixsw, wsetinvpchw

wgetinvpcsw

Lists the options that are set for the software scan of a PC endpoint.

Syntax

wgetinvpcsw [-b] [-c] [-f] [-h] [-r] [-s] [-x] [-m] [-d] *profile_name*

Description

The **wgetinvpcsw** command returns the options for PC software scans that are configured for an inventory profile. This command can return information about whether a profile is configured to perform the following actions:

- Scan for basic file information.
- Send scan data to the configuration repository.
- Generate checksum values for scanned files.
- Apply the custom filter to scans for basic information.
- Scan files for header information.
- Scan the Windows registry for information about installed products.
- Scan for signature data.
- Scan executable files only.
- Download the swsigs.txt file to the endpoint.

Run this command with no options to return all information about the options for PC software scans that are configured for the specified inventory profile.

Options

- b** Returns information about scans for basic file information. The following values can be returned:
- SCAN** The endpoint is scanned for basic file information, and then the scan data is stored on the endpoint.
- UPDATE**
The endpoint is not scanned. Data from a previous scan for basic file information is collected and sent to the configuration repository.
- BOTH** The endpoint is scanned for basic file information, and then the scan data is collected and sent to the configuration repository.
- NO** No scan for basic file information is performed. No data for basic file information is collected.
- c** Returns information about the configuration of the checksum options. The following values can be returned:
- NONE**
The scan does not generate checksum values of scanned files.
- QUICK**
The scan collects the Quick checksum value of scanned files.
- FULL** The scan collects the Full checksum value of scanned files.

- MD5** The scan collects the MD5 checksum value of scanned files.
- f** Returns information about whether the custom filter is applied to scans for basic file information. The following values can be returned:
- YES** The scan applies the custom filter to scans for basic file information.
- NO** The scan does not apply the custom filter to scans for basic file information.
- h** Returns information about scans for header information. The following values can be returned:
- SCAN** The endpoint is scanned for header information, and then the scan data is stored on the endpoint.
- UPDATE**
The endpoint is not scanned. Data from a previous scan for header information is collected and sent to the configuration repository.
- BOTH** The endpoint is scanned for header information, and then the scan data is collected and sent to the configuration repository.
- NO** No scan for header information is performed. No data for header information is collected.
- r** Returns information about scans of the Windows registry for information about installed products. The following values can be returned:
- SCAN** The Windows registry is scanned for information about installed products, and then the scan data is stored on the endpoint.
- UPDATE**
The endpoint is not scanned. Data from a previous scan for registry information is collected and sent to the configuration repository.
- BOTH** The Windows registry is scanned for information about installed products, and then the scan data is collected and sent to the configuration repository.
- NO** No scan for registry information is performed. No data for registry information is collected.
- s** Compares scanned files against the list of signatures. The following values can be returned:
- SCAN** Scanned files are compared to the list of signatures. Data for matching files is stored on the endpoint.
- UPDATE**
A scan for signature matching is not run. Data from a previous scan for signature matching is collected and sent to the configuration repository.
- BOTH** Scanned files are compared to the list of signatures, and then the data for matching files is collected and sent to the configuration repository.

	NO	No scan for signature matching is performed. No data from signature matching is collected.
-x		Scans executable files only. The following values can be returned:
	YES	Only executable files are scanned.
	NO	The scan is not limited to executable files.
-m		Displays information related to the patch scan setting. The following values can be returned:
	YES	The patch scan is enabled.
	NO	The patch scan is not enabled.
-d Y N		Specifies whether the swsign.txt file must be downloaded to the endpoint.
<i>profile_name</i>		The name of the inventory profile about which you want information. Enter the profile name in the format @InventoryConfig:profile_name .

Authorization

user, admin, senior, super, or Inventory_view

Examples

The following example returns all information about the options for PC software scans that are configured for the inventory profile MyProfile:

```
wgetinvpcsw @InventoryConfig:MyProfile
```

This command returns the following output:

```
Scan for and update basic file information: NO
Cyclic Redundancy Check: No CRC
Scan for and update header information: BOTH
Scan for and update PC registry information: BOTH
Scan for and update matching software signatures: NO
Apply custom filter to basic file information scan results: NO
Executable files only: YES
```

The following example returns the checksum option configured for the inventory profile MyProfile:

```
wgetinvpcsw -c @InventoryConfig:MyProfile
```

This command returns the following output:

```
Cyclic Redundancy Check: No CRC
```

See Also

wgetinvglobal, wgetinvpcfiles, wgetinvpchw, wgetinvunixfiles, wgetinvunixhw, wgetinvunixsw, wsetinvpcsw

wgetinvpvdconfig

Returns a value that indicates whether an inventory profile is set to collect configuration data for pervasive devices.

Syntax

wgetinvpvdconfig -t *profile_name*

Description

You use the **wgetinvpvdconfig** command to determine whether an inventory profile is set to collect configuration data for pervasive devices such as personal digital assistants (PDAs). Configuration data includes language and alarm settings and time and date format.

Options

-t	Returns a value that indicates whether an inventory profile is set to collect configuration data for devices. The following values can be returned: YES Devices are scanned for configuration data. NO Devices are not scanned for configuration data.
<i>profile_name</i>	The inventory profile about which you want information. Enter the profile name in the format @InventoryConfig:profile_name .

Authorization

user, admin, senior, super, or Inventory_view

Examples

The following example returns a value that indicates whether the profile named MyProfile is set to collect configuration data during scans of devices:

```
wgetinvpvdconfig -t @InventoryConfig:MyProfile
```

This command returns the following output:

```
Retrieve Pervasive Configuration Information:YES
```

See Also

wgetinvpvdhw, wgetinvpvdsw, wsetinvpvdconfig, wsetinvpvdhw, wsetinvpvdsw

wgetinvpvdhw

Returns a value that indicates whether hardware scanning of pervasive devices is enabled.

Syntax

wgetinvpvdhw -t *profile_name*

Description

The **wgetinvpvdhw** command returns a value that indicates whether hardware scanning of pervasive devices is enabled for the specified profile.

Options

-t	Returns a value that indicates whether an inventory profile is set to collect hardware data during scans of devices. The following values can be returned: YES Hardware scanning of devices is enabled. NO Hardware scanning of devices is disabled.
<i>profile_name</i>	The inventory profile about which you want information. Enter the profile name in the format @InventoryConfig:profile_name .

Authorization

user, admin, senior, super, or Inventory_view

Examples

The following example returns a value that indicates whether the profile named MyProfile is set to collect hardware data during scans of devices:

```
wgetinvpvdhw -t @InventoryConfig:MyProfile
```

This command returns the following output:

```
Retrieve Pervasive Hardware Information:YES
```

See Also

wgetinvpvdconfig, wgetinvpvdsw, wsetinvpvdconfig, wsetinvpvdhw, wsetinvpvdsw

wgetinvpvdsw

Returns a value that indicates whether software scanning of pervasive devices is enabled.

Syntax

wgetinvpvdsw -t *profile_name*

Description

The **wgetinvpvdsw** command returns a value that indicates whether software scanning of pervasive devices is enabled for the specified profile.

Options

-t	Returns a value that indicates whether an inventory profile is set to collect software data during scans of devices. The following values can be returned: YES Software scanning of devices is enabled. NO Software scanning of devices is disabled.
<i>profile_name</i>	The inventory profile about which you want information. Enter the profile name in the format @InventoryConfig:<i>profile_name</i> .

Authorization

user, admin, senior, super, or Inventory_view

Examples

The following example returns a value that indicates whether the profile named MyProfile is set to collect software data during scans of devices:

```
wgetinvpvdsw -t @InventoryConfig:MyProfile
```

This command returns the following output:

```
Retrieve Pervasive Software Information:YES
```

See Also

wgetinvpvdconfig, wgetinvpvdhw, wsetinvpvdconfig, wsetinvpvdhw, wsetinvpvdsw

wgetinvswd

Returns a value that indicates whether Inventory has been enabled to integrate with Software Distribution.

Syntax

wgetinvswd

Description

The **wgetinvswd** command returns a value that indicates whether Inventory has been enabled to integrate with Software Distribution using the **wsetinvswd** command. The following values can be returned:

- y** Inventory has been enabled to integrate with Software Distribution.
- n** Inventory has not been enabled to integrate with Software Distribution.

Options

There are no options for this command.

Authorization

user, **admin**, **senior**, **super**, or **Inventory_view**

Examples

The following example returns a value that indicates whether Inventory has been enabled to integrate with Software Distribution:

```
wgetinvswd
```

This command returns the following output:

```
Inventory-Software Distribution Integration is enabled: n
```

See Also

wsetinvswd, **wswdmgr** in the *Reference Manual for Software Distribution*

wgetinvunixfiles

Lists information about the specified inventory profile regarding files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of UNIX and OS/400 endpoints.

Syntax

wgetinvunixfiles [-b] [-d] [-f] [-m] [-s] *profile_name*

Description

The **wgetinvunixfiles** command returns the following information about the specified inventory profile regarding scans of UNIX and OS/400 endpoints:

- The files or file types that are included or excluded during a software scan.
- The directories that are included or excluded during a software scan.
- Scripts to be run during the profile distribution before or after the scan is performed.
- The MIF files to be read during the scan.

Run this command with no options to return all information about files and directories to be scanned, scripts to be run, and custom MIF files to be collected for the specified inventory profile.

Options

-b	Returns the contents of the script to be run on targets during the profile distribution before the scan is performed.
-d	Returns the list of directories to be included or excluded during the scan. This option does not apply to OS/400 endpoints.
-f	Returns the list of files or file types to be included or excluded during the scan. This option does not apply to OS/400 endpoints.
-m	Returns the list of MIF files to be read during the scan.
-s	Returns the contents of the script to be run on targets during the profile distribution after the scan is performed.
<i>profile_name</i>	The name of the inventory profile about which you want information. Enter the profile name in the format @InventoryConfig:profile_name .

Authorization

user, admin, senior, super, or Inventory_view

Examples

The following example returns all information about files and directories to be scanned, scripts to be run, and custom MIF files to be collected by the inventory profile MyProfile:

```
wgetinvunixfiles @InventoryConfig:MyProfile
```

The output from this command is as follows:

```
No Include Directories
Exclude Directories:
    */tmp/
Exclude File Types:
```

```
*.C  
*.txt  
No Custom Mif Files  
No script to run before scan  
No script to run after scan
```

See Also

**wgetinvglobal, wgetinvpcfiles, wgetinvpchw, wgetinvpcsw, wgetinvunixhw,
wgetinvunixsw, wsetinvpcfiles, wsetinvunixfiles**

wgetinvunixhw

Lists the options that are set for the hardware scan of UNIX and OS/400 endpoints.

Syntax

wgetinvunixhw [-a] [-t] [-u] *profile_name*

Description

The **wgetinvunixhw** command returns the options for the hardware scan of UNIX and OS/400 endpoints. This command can return the following information about a profile:

- Whether the Tivoli hardware scanner is run during scans of UNIX and OS/400 endpoints.
- Whether scan data is sent to the configuration repository or stored on the endpoint to be collected later.
- Which hardware components are scanned for.

Options

-a Lists all the hardware components that you can scan for, and indicates whether the inventory profile is configured to scan for each component.

-t Returns a value that indicates whether the Tivoli hardware scanner is enabled. The following values can be returned:

YES The Tivoli hardware scanner is enabled.

NO The Tivoli hardware scanner is disabled.

-u Returns a value that indicates whether the data collected during hardware scans of UNIX and OS/400 systems is sent to the configuration repository. The following values can be returned:

YES Scan data is sent to the configuration repository after the scan completes.

NO Scan data is not sent to the configuration repository. It is stored on the endpoint.

profile_name

The name of the inventory profile about which you want information. Enter the profile name in the format **@InventoryConfig:profile_name**.

Authorization

user, **admin**, **senior**, **super**, or **Inventory_view**

Examples

The following example returns all information about the options for UNIX and OS/400 hardware scans that are configured for inventory profile MyProfile:

```
wgetinvunixhw @InventoryConfig:MyProfile
```

This command returns the following output:

```
Run Tivoli Scanner: YES
Update the hardware scan results in the configuration repository: YES
Data to scan:
Processor: YES
```

```
Memory: YES
MemoryModules: YES
Operating System: YES
Storage: YES
IP Address: YES
Network Adapter: YES
Partition: YES
Pointing Device: YES
Keyboard: YES
UNIX System Params: YES
SMBIOS: YES
Lpar: YES
```

The following example indicates whether inventory profile MyProfile is configured to run the Tivoli hardware scanner:

```
wgetinvunixhw -t @InventoryConfig:MyProfile
```

This command returns the following output:

```
Run Tivoli Scanner: YES
```

See Also

wgetinvglobal, wgetinvpcfiles, wgetinvpchw, wgetinvpcsw, wgetinvunixfiles, wgetinvunixsw, wsetinvunixhw

wgetinvunixsw

Lists the options that are set for the software scan of UNIX and OS/400 endpoints.

Syntax

wgetinvunixsw [-b] [-c] [-f] [-p] [-s] [-x] [-d] *profile_name*

Description

The **wgetinvunixsw** command returns the options for UNIX software scans that are configured for an inventory profile. This command can return information about whether a profile is configured to perform the following actions:

- Scan for basic file information.
- Generate the checksum value of scanned files.
- Apply the custom filter to scans for basic information.
- Scan the operating system for information about installed products.
- Compare scanned files against the list of signatures.
- Scan executable files only.
- Download the swsigs.txt file to the endpoint.

Run this command with no options to return all information about the options for UNIX software scans that are configured for the specified inventory profile.

Options

- b** Returns information about scans for basic file information. The following values can be returned:
- SCAN** The endpoint is scanned for basic file information, and then the scan data is stored on the endpoint.
- UPDATE**
The endpoint is not scanned. Data from a previous scan for basic file information is collected and sent to the configuration repository.
- BOTH** The endpoint is scanned for basic file information, and then the scan data is collected and sent to the configuration repository.
- NO** No scan for basic file information is performed. No data for basic file information is collected.
- c** Returns information about the configuration of the checksum options. The following values can be returned:
- NONE**
The scan does not generate checksum values of scanned files.
- QUICK**
The scan collects the Quick checksum value of scanned files.
- FULL** The scan collects the Full checksum value of scanned files.
- MD5** The scan collects the MD5 checksum value of scanned files.

-f	Returns information about whether the custom filter is applied to scans for basic file information. The following values can be returned: YES The scan applies the custom filter to scans for basic file information. NO The scan does not apply the custom filter to scans for basic file information.
-p	Returns information about scans of the operating system for information about installed products. The following values can be returned: SCAN The operating system is scanned for information about installed products, and then the scan data is stored on the endpoint. UPDATE The endpoint is not scanned. Data from a previous scan for information about installed products is collected and sent to the configuration repository. BOTH The operating system is scanned for information about installed products, and then the scan data is collected and sent to the configuration repository. NO No scan for information about installed products is performed. No data for information about installed products is collected.
-s	Compare scanned files against the list of signatures. The following values can be returned: SCAN Scanned files are compared to the list of signatures. Data for matching files is stored on the endpoint. UPDATE A scan for signature matching is not run. Data from a previous scan for signature matching is collected and sent to the configuration repository. BOTH Scanned files are compared to the list of signatures, and then the data for matching files is collected and sent to the configuration repository. NO No scan for signature matching is performed. No data from signature matching is collected.
-x	Scan executable files only. The following values can be returned: YES Only executable files are scanned. NO The scan is not limited to executable files.
-d Y N	Specifies whether the swsigs.txt file must be downloaded to the endpoint.
<i>profile_name</i>	The name of the inventory profile about which you want information. Enter the profile name in the format @InventoryConfig:profile_name .

Authorization

user, admin, senior, super, or Inventory_view

Examples

The following example returns all information about the options for UNIX software scans that are configured for the inventory profile MyProfile:

```
wgetinvunixsw @InventoryConfig:MyProfile
```

This command returns the following output:

```
Scan for and update basic file information: NO
Cyclic Redundancy Check: No CRC
Scan for and update Unix software package information: BOTH
Scan for and update matching software signatures: NO
Apply custom filter to basic file information scan results: NO
Executable files only: YES
```

The following example returns the checksum option configured for the inventory profile MyProfile:

```
wgetinvunixsw -c @InventoryConfig:MyProfile
```

This command returns the following output:

```
Cyclic Redundancy Check: No CRC
```

See Also

wgetinvglobal, wgetinvpcfiles, wgetinvpchw, wgetinvpcsw, wgetinvunixfiles, wgetinvunixhw, wsetinvunixsw

wgetscanstat

Returns information about current inventory scans.

Syntax

wgetscanstat

wgetscanstat -a [-s] [-f] [-p]

wgetscanstat -i scan_ID [-i scan_ID]... [-s] [-f] [-p]

Description

The **wgetscanstat** command returns information about current inventory scans. If you run this command without any options, it returns a list of scan IDs of current inventory scans along with the names of the inventory profiles that initiated those scans. With the available options, you can retrieve information about success, failure, or pending status of all or specific scans.

Options

- a** Returns detailed information about all current scans, including the scan ID, the profile name, the start time, elapsed time, and the number of targets on which scans are completed or pending. (A scan is considered complete after the data for the scanned endpoints is written to the configuration repository.)
- s** Returns a list of all the targets on which scans have completed successfully.
- f** Returns a list of all the targets on which scans have failed.
- p** Returns a list of all the targets on which scans are still pending.
- i scan_ID**
Returns information about the scan with the specified scan ID, including the scan ID, the profile name, the start time, elapsed time, and the number of targets on which scans are completed or pending. You can specify multiple scan IDs. To get the ID of a scan, run the **wgetscanstat** command with no options or with the **-a** option.

Authorization

user, admin, senior, super, or Inventory_view

Examples

The following example returns detailed information about all running scans and lists failed, successful, and pending scans.

```
wgetscanstat -a -f -s -p
```

This command returns the following output:

```
Scan ID:          144
Distribution ID:   1929790663.8
Profile Name:     Hardware
Start time:       Thu Apr 05 17:18:57 2002
Elapsed time:     0 Days 0 Hours 0 Minutes 11 Seconds
Clients completed: 3
Clients pending:  1
The following clients have successfully completed:
```



```
mckinley
aztlan-1
The following clients have failed:
  alioth-5
The following clients are still pending:
  suntmp18-3
Scan ID:          145
Distribution ID:   1929790663.9
Profile Name:     Hardware
Start time:       Thu Apr 05 17:18:57 2002
Elapsed time:     0 Days 0 Hours 0 Minutes 11 Seconds
Clients completed: 0
Clients pending:  1
The following clients have successfully completed:
The following clients have failed:
The following clients are still pending:
  suntmp11
```

See Also

wcollect, wcstat, wcancelstat.

winvdeps

Enables and disables the Common Inventory Technology and Tivoli License Manager dependencies.

Syntax

winvdeps -s *depset*

winvdeps -a *depset*

winvdeps -r *depset*

Description

You can specify if Common Inventory Technology and Tivoli License Manager software package blocks are added to the dependency set of Inventory to be downloaded at the endpoint, when an Inventory scan is requested. The **winvdeps** command enables you to distribute the Common Inventory Technology and Tivoli License Manager software package blocks using Software Distribution. The dependency set (*depset*) is either a set of Common Inventory Technology or Tivoli License Manager dependencies.

Options

- | | |
|-------------------------|---|
| -s <i>depset</i> | Verifies if Common Inventory Technology and Tivoli License Manager software package blocks are contained in the Inventory dependency set. |
| -a <i>depset</i> | Adds Common Inventory Technology and Tivoli License Manager software package blocks to the Inventory dependency set. |
| -r <i>depset</i> | Removes Common Inventory Technology and Tivoli License Manager software package blocks from the Inventory dependency set. |

where **depset** is either **CIT** or **TLM**.

Authorization

admin, senior, orsuper

Examples

The following example adds to the dependency set the files needed for the installation of the Tivoli License Manager (TLM) agent.

```
sh winvdeps -a TLM
```

The following example reports if the dependencies needed to automatically download Common Inventory Technology (CIT) are enabled or not.

```
sh winvdeps -s CIT
```

winvfilter

Modifies files from the custom filter list in the configuration repository.

Syntax

winvfilter -a -f *input_file*

winvfilter -r -f *input_file*

winvfilter -a -n *filter_name*

winvfilter -r -n *filter_name*

winvfilter -a -i

winvfilter -r -i

Description

You can specify that scans for basic information include only files listed in the custom filter. The **winvfilter** command enables you to add or delete file names from the custom filter list. The custom filter list is stored in the configuration repository. You can specify one file name at a time using the -n option, or you can specify multiple file names using the -f option. For more information about the custom filter, see the Inventory online help.

Note: On UNIX systems, file names are case sensitive.

You enable the custom filter using the **wsetinvpcsw** or **wsetunixsw** command and -f option, or by selecting the Apply custom filter option on the Software Scan Configuration window in the Inventory graphical user interface (GUI).

Options

- | | |
|-----------------------|--|
| -a | Adds the specified file or list of files to the custom filter. |
| -f <i>input_file</i> | Specifies the name of a text file that contains a list of file names to be added to, or removed from, the custom filter. Within the input file, you must list each file name on a separate line. |
| -i | Specifies filter data to be read from standard input. You enter data by piping input from a command. |
| -n <i>filter_name</i> | Specifies a single file name to be added to, or removed from, the custom filter. You cannot specify multiple file names using the -n option. |
| -r | Removes the specified file or list of files from the custom filter. |

Authorization

RIM_update, user, admin, senior, or super

Examples

The following example adds the list of file names from the file filterlist.txt to the custom filter.

```
winvfilter -a -f filterlist.txt
```

The following example removes the file name myapp.exe from the custom filter.

```
winvfilter -r -n myapp.exe
```

The following example loads the file names from the signature table (SIGNATURE) into the custom filter list. This example is for a DB2 RDBMS and assumes that the DB2 environment is sourced and a connection to the database has been established:

```
db2 "select sware_name from sware_sig" | winvfilter -a -i
```

Ensure that after the installation of fix pack 2, the sql select is modified as follows:

```
db2 "select distinct name from swcat.signature where sig_type=0" | winvfilter -a -i
```

Notice that the Common Inventory Technology (CIT) scanner component does not accept more than 10000 filters. If you really need to use more than 10000, plan to use the "Scan for installed products using signature matching" scan.

See Also

winvsig, wsetinvpcsw, wsetinvunixsw

winviso

Installs files on an endpoint that you can use to scan disconnected systems.

Before running the **winviso** command, you must perform an Inventory scan on the endpoint.

Syntax

winviso @InventoryConfig:profile_name @Endpoint:endpoint_name

Description

The **winviso** command enables you to run isolated scans. An isolated scan is a scan of a system that is not in your Tivoli region. The system could even be disconnected from the network. You can run isolated scans on supported Windows and UNIX systems except Linux for S/390.

Using the **winviso** command, you can copy to an endpoint all of the files necessary to scan a disconnected system (libraries, executables, signatures, and so on). Files for each supported Windows and UNIX platform are provided. You can then copy files from the endpoint to the disconnected system and run an isolated scan on the disconnected system using the **wepscan** command. This creates a .DAT file that contains the scan data. You must manually move the .DAT file from the disconnected system to the endpoint. You can then run the **wloadiso** command on the endpoint to send the scan data to the configuration repository.

When you run the **winviso** command, the following directories are created on the specified endpoint:

\$LCFROOT/inv/ISOLATED/common/platform_type

This directory contains all the dependencies (libraries, executable files, and so on) needed to scan each supported platform type.

\$LCFROOT/inv/ISOLATED/common/codeset

This directory contains codeset.zip, a compressed file that contains the codeset files needed to scan disconnected systems.

\$LCFROOT/inv/ISOLATED/profile_name

This directory contains the file config.iso. This encoded and compressed file contains information about the inventory profile and other files needed to scan disconnected systems.

\$LCFROOT/inv/ISOLATED/depot

This directory is a depot for the scan data (the DAT file) from the disconnected system.

See “Scanning disconnected systems” on page 56 for the procedure to run isolated scans. For more information about isolated scans, see “wepscan” on page 127 and “wloadiso” on page 178.

Options

@InventoryConfig:profile_name

Specifies the inventory profile that you want to distribute to a disconnected system.

@Endpoint:*endpoint_name*

Specifies the endpoint that you want to copy the profile and related files to.

Authorization

Inventory_scan, senior, or super

Examples

The following example copies the inventory profile Software_Scan and related files to the endpoint duilio.

```
winviso @InventoryConfig:Software_Scan @Endpoint:duilio
```

See Also

wepscan, wloadiso

winvmgr

Creates, changes, deletes, and lists the default values of keys for inventory.

Syntax

winvmgr -l

winvmgr -p *policy_region* -d [*key*]

winvmgr -p *policy_region* -c *key=value*

winvmgr -p *policy_region* -e *key*

Description

The **winvmgr** command enables you to create or change the default value of a key, delete a key, or list the available keys for inventory.

Options

-p *policy_region*

Specifies the policy region in which you want to create, change, or delete default keys. This parameter is mandatory with the **-d**, **-c**, and **-e** options.

-l Lists all available keys for inventory.

-d [*key*]

If no policy region is specified by the **-p** option, lists all default values for all policy regions. If a policy region is specified by the **-p** option, this option lists all default values for that policy region. If a key is specified, this option lists the value for the specified key only.

-c *key=value*

Sets the default value for a key. You can use the following keywords and values:

deadline=*mm/dd/yyyy hh:mm*

Specifies the date and time at which a distribution expires (the date that it fails for unavailable target systems, including pervasive devices). If a target has not received a distribution by this time, MDist 2 no longer attempts the distribution. The default value is 72 hours from the time the profile is distributed. You cannot set both this value and the **dist_timeout** value.

dist_timeout=*hours*

Specifies the number of hours after the profile is distributed that the distribution expires (it fails for unavailable target systems, including pervasive devices). For the value *hours* you must specify a positive integer value. The default value is 72 hours. You cannot set both this value and the **deadline** value.

dist_validate_db_conn=*y* | *n*

Specifies if the MDist 2 RIM database must be running during the distribution. If set to *y*, the distribution stops if the database is down. If set to *n* (the default), the distribution goes through whether the database is up or down, but no tracing message is returned if the database is down.

escalate_date_n=mm/dd/yyyy hh:mm

Specifies the date on which a reminder message must be sent to mobile or roaming targets (such as laptops) that have not yet received the profile. For the value *n*, you can specify a number, 0 through 9, which enables you to create up to ten messages. For *n*, you must specify the values in ascending numerical order, and you cannot skip a number. For each escalation date, you must also specify an escalation message using **escalate_msg**.

escalate_msg_n="message text"

Specifies a message that must be sent to roaming targets that have not received the profile by the associated escalation date. For the value *n* you can specify a number, 0 through 9, which enables you to create up to ten messages. For *n*, you must specify the values in ascending numerical order and you cannot skip a number. For each escalation message, you must also specify an escalation date using **escalate_date**. Enter a message using the following format:

```
escalate_msg_n="message text"
```

execute_timeout=seconds

Specifies the number of seconds that MDist 2 waits for an endpoint to return scan status after the profile has been distributed. For scans of endpoints, this value includes the time that it takes to run the scan and any custom scripts and make an upcall to SCS to request data collection. If the upcall to SCS fails, this value includes the time that it takes to send data to the inventory callback object. For scans of pervasive devices, the execution timeout is the length of time MDist 2 waits for a job to be created on the Web Gateway component. If you plan to run a scan that will take a long time to complete, for example if you must scan a large number of drives, you would set this value to allow for that time.

force_mandatory=y | n

Controls the way in which mandatory distributions on roaming endpoints are treated once the mandatory date is passed. Specify one of the following options:

- **y**—The mandatory distribution is automatically started as soon as the roaming user connects. This is the default option.
- **n**—The roaming user has the choice of not starting the mandatory distribution. However, the user will not be able to perform any other operations until the mandatory distribution has been performed.

hidden={y | n}

Indicates whether a distribution to a roaming endpoint is hidden. Hidden distributions are not revealed to the roaming user and are automatically distributed to the roaming endpoint. Distributions that are not hidden are visible to the roaming user, and the roaming user can defer the distribution. Use one of the following options:

- **y**—Hides the distribution.
- **n**—Does not hide the distribution.

If you set this option to **y**, you must not set values for **mandatory_date**, **escalate_date**, or **escalate_msg**.

This option performs the same function as the **wsetinvglobal** command and the **-m** option. If this option and the **wsetinvglobal** command and the **-m** option are set to conflicting values, the **wsetinvglobal** command and the **-m** option override this option.

If you set this option in an InventoryConfig profile, its value overrides the hidden keyword value.

inv_ep_debug={1 | 2 | 3}

Enables debugging of scanned endpoints and pervasive devices, and specifies the level of debugging information to be logged. When this option is enabled, information is collected from each scanned target and saved to a log file. Use one of the following options:

- 1** Logs error messages.
- 2** Logs error and warning messages.
- 3** Logs error and warning messages and debugging information. (Scanner debugging information is not available from NetWare or OS/2 endpoints.)

If you do not specify a value, debugging information for endpoints is not logged.

Log files for each scanned endpoint are collected by the scan and saved on the inventory data handler in the `$DBDIR/inventory/data_handler/logfiles/scan_ID/endpoint_name` directory, where:

- *scan_ID* is the ID of the scan.
- *endpoint_name* is the name of the endpoint for which debugging information was collected.

For scans of pervasive devices, the log file is stored in `$LCFROOT/inv/TWG/INVscan_ID` on the endpoint that hosts the Web Gateway component. A single log file is created for all of the devices processed on the endpoint.

The name of the log file for both endpoints and pervasive devices is **INV000xx.log**, where *xx* is the scan ID.

inv_tlm_enabled = y | n

Specifies whether the integration with Tivoli License Manager is enabled or not. All the Inventory profiles defined inside a policy region can be enabled for this integration. The default value is no.

is_exclusive = y | n

Specifies whether the distribution to the target is sent exclusively.

is_fail_unavailable = y | n

Specifies if the status of the distribution must be set to failed, if the endpoint is unavailable. When this option is enabled, for example MDist 2 reports the status of the distribution as failed, when the endpoint is powered off. If this option is not enabled, MDist 2 keeps contacting the endpoint until the distribution deadline is reached.

is_multicast = y | n

Specifies whether the distribution is multicast enabled.

label=description_string

Specifies a description string for the distribution. The default value is the name of the inventory profile.

mandatory_date=mm/dd/yyyy hh:mm

Specifies the date by which the distribution must be made. Distributions to roaming targets can be deferred up to this date. When the date is reached, the profile is automatically distributed to all roaming targets that have not yet accepted it. The default value is 60 hours from the time the profile is distributed.

mc_upcall_timeout=seconds

Specifies the length of time, in seconds, that an endpoint collector waits for a down call from a gateway collector before it sends a .DAT file through the callback object.

notify_interval=minutes

Specifies the interval, in minutes, in which each repeater bundles and returns the completed results of the MDist 2 distribution. Use a positive integer. The default value is 1 minute.

PM_WSUS_enabled={y | n}

Specifies if you want to use the Microsoft WSUS Server for the Automated Patch Management solution provided by Tivoli Configuration Manager. Use one of the following options:

- y** You intend to use the Microsoft WSUS Server for your patch management operations.
- n** You intend to use a different server for your patch management operations.

priority={h | m | l}

Specifies the priority for distributions to endpoints and pervasive devices. Priority is the order in which distributions are handled by repeaters. You can set priority to one of the following values:

- h** Highest priority.
- m** Medium priority. This is the default value.
- l** Low priority.

roam_endpoints={y | n}

Indicates whether the operation defined in the **wdistinv** command supports roaming endpoints. Use one of the following options:

- y** The distribution can be transferred to any gateway where the roaming endpoint connects.
- n** After the profile is queued at a gateway it cannot be transferred to another gateway.

send_timeout=seconds

Specifies the number of seconds that a repeater waits for a target system to receive a block of data (for example, a repeater distributing an inventory profile to an endpoint). Each time a packet is sent over the network, the timeout for that packet is set to this value. This timeout is used to detect network or endpoint failures. If a timeout occurs, the distribution remains in the repeater's queue and a retry occurs according to the **conn_retry_interval** value set with the Tivoli Management Framework **wmdist** command.

wake_on_lan={y | n}

Indicates whether the operation sends a Wake on LAN message to trigger the start of a system that is not available when an inventory profile is distributed. For scans of pervasive devices, this option starts the endpoint on which the Web Gateway component resides. This option works only on systems that have a network card that is enabled for Wake on LAN. Use one of the following options:

- **y**—Sends a Wake on LAN message.
- **n**—Does not send a Wake on LAN message.

If you set this option in an InventoryConfig profile, its value overrides the wake_on_lan keyword value.

-e key Deletes the default key and value.

Authorization

admin

Examples

The following example sets the key **dist_timeout** in policy region **region1** to 72 hours.

```
winvmgr -p region1 -c dist_timeout=72
```

winvmigrate

Imports the new IBM signatures and migrates the signature packages.

Syntax

winvmigrate **-s** | **-c** *new_IBM_catalog_path*

Description

The **winvmigrate** command enables you to import the new IBM signatures file, and to migrate the signature packages. At the end of the migration process, the command updates the state of the old IBM signatures from **2** to **0**. It also adds the following entry to the SWCAT.CONTROL table:

TCM_DB_MIGRATED=yes

Options

- s** Returns the list of the signature packages that will be automatically migrated.
- c** *new_IBM_catalog_path*
Imports the file containing the new IBM signatures, and migrates the signature packages. The signature catalog can be downloaded from the following IBM web site:
<http://www-1.ibm.com/support>

Authorization

RIM_update, admin, senior, or super

Examples

The following examples import the signatures contained in the catalog and migrate the signature packages.

```
winvmigrate -c C:\temp\IBM_SoftwareCatalog.xml
```

or

```
winvmigrate -C itlcm22-allProducts-fullSwCat-YYYYMMDD.xml
```

winvpackage

Logically groups two or more signatures so that you can use them together to identify software.

Syntax

winvpackage **-d** *package_description* **-v** *package_version* **-f** *file...*

winvpackage **-d** *package_description* **-v** *package_version* **-i** *signature*

winvpackage **-l**

winvpackage **-r** **-s** *signature_package_ID*

winvpackage **-t**

Description

The **winvpackage** command logically groups multiple signatures so that you can use them to identify a single software application or a collection of applications. A group of signatures is called a *signature package*.

Unlike signatures, signature packages are not distributed to endpoints during scans. To use signature packages, you must first scan the systems in your enterprise using a signature scan. Then you can use the SIG_PACKAGE_QUERY to view the scan data in the configuration repository that matches a signature package.

Options

-d *package_description*

Use this option to enter a description of the signature package. You can enter any alphanumeric value. If the value contains spaces, you must enclose it in double quotation marks ("").

-f *file* Specifies the signatures that you want to add to the signature package. You must specify the signatures in the following format: *file_name/file_size*, where *file_name* is the name of the file, and *file_size* is the size of the file in bytes. The file name and size of each signature that you enter must exactly match existing signatures, including case. To view existing signatures, see the Signatures window of the Inventory GUI. If a file name contains spaces, you must enclose both the file name and size in double quotation marks (""). The file size must be a positive integer value. The separator between name and size must be a forward slash (/). You can specify multiple signatures. Separate each signature with a space.

-i *signature*

Indicates the ID of the signature that you want to add to the signature package. A signature ID is the 32-character value that Inventory assigns to each signature. This value is stored in the configuration repository in the SIGNATURE table. Enter the value exactly as it appears in the SIGNATURE table.

-l Lists the signature packages that you have created. The data for the signature packages is stored in the SIG_PACKAGE table. This command option lists the following information:

- Signature package ID. This is a unique 32-character value that Inventory assigns to each signature package.
 - Signature package description.
 - Signature package version.
 - Name and size of each file that the signature package matches. This is the information that you added using the **-f** option.
- r** Deletes a signature package. When you delete a signature package, the data for the package in the SIG_PACKAGE table is deleted, and the logical link between the signatures is removed. However, the signatures themselves are retained.
- s *signature_package_ID***
Specifies the ID of the signature package that you want to remove.
- t** Tests the validity of the signatures in the signature packages that you have created. The file name and size of the signatures in each package must exactly match existing signatures, which are stored in the SWARE_SIG table. This command lists the ID of any packages that contain signatures that are not valid as well as the ID of the signatures.
- v *package_version***
Use this option to enter a version for the signature package that you are creating. You can enter any alphanumeric value. If the value contains spaces, you must enclose it in double quotation marks ("").

Authorization

RIM_update, admin, senior, or super

Notes

Signature packages work only with data that is collected from signature scans. Before you use SIG_PACKAGE_QUERY, run a signature scan on all of the appropriate endpoints and make sure all scan data is in the configuration repository.

If you use **winvpackage** to edit an existing signature package, the existing signature package is not deleted, and a new signature package is created. You must manually delete the original signature package if you do not want it.

Examples

The following command creates a signature package called Standard Configuration, assigns it a version of 1.0, and adds three signatures to the package.

```
winvpackage -d "Standard Configuration" -v 1.0 -f \
coreldr.exe/11864576 FrameMaker+SGML.exe/3792949 \
WINWORD.EXE/8798720
```

The following example lists the signature packages that you have created.

```
winvpackage -l
```

The output from this command is as follows.

30a74228d2661c2f3577a2e927b5ece4	Pkg 1	1	app1.exe	6081550
30a74228d2661c2f3577a2e927b5ece4	Pkg 1	1	app2.exe	3194843
ce0d83efb20a72cfc8f11595e039f62a	Pkg 2	2	app3.exe	5901848
ce0d83efb20a72cfc8f11595e039f62a	Pkg 2	2	abmx.dll	21504
ce0d83efb20a72cfc8f11595e039f62a	Pkg 2	2	rfst.dll	9758

Successfully processed 5 lines.

See Also

`winvsig`

winvrnode

Removes from the configuration repository all scanned information about an endpoint or pervasive device.

Syntax

winvrnode **-i** *computer_system_ID* ...

winvrnode **-l** *object_label* ...

winvrnode **-o** *object_ID* ...

Description

The **winvrnode** command removes hardware and software scan information for a specified endpoint or pervasive device from the configuration repository.

This command does not remove data from history tables or custom tables unless you add the table names to the INVENTORY_TABLES table.

You can specify an endpoint by providing either the name (*object_label*), object ID (*object_ID*), or computer system ID (*computer_system_ID*) of the system. You can specify a pervasive device by providing the computer system ID (*computer_system_ID*) of the system.

If more than one endpoint has the same name, using *object_label* removes the inventory information for all endpoints with that name, and using *object_ID* or *computer_system_ID* removes the information for just the specified endpoint.

You can remove information for more than one system by providing a list of names.

Note: The object ID of a system is the value of TME_OBJECT_ID. To find the object ID for a system, run the INVENTORY_HWARE query with the TME_OBJECT_ID and TME_OBJECT_LABEL as chosen columns. The results show system names and their corresponding object IDs. The computer system ID of a system is the value of COMPUTER_SYS_ID. To find the computer system ID for a system, run the INVENTORY_HWARE query.

Options

-i *computer_system_ID*

Removes from the configuration repository all hardware and software information for the endpoint or pervasive device with the specified computer system ID.

-l *object_label*

Removes from the configuration repository all hardware and software information for the endpoint with the specified name. If multiple instances of the same label exist, this option removes all instances.

-o *object_ID*

Removes from the configuration repository all hardware and software information for the endpoint with the specified object ID.

Authorization

super

Notes

For information about using **winvrmtree** to remove scan data from custom tables, see “Adding a custom table to the configuration repository” on page 69.

Examples

The following command removes all hardware and software information for the endpoints with the object labels sly and robbie.

```
winvrmtree -l sly robbie
```

The following command removes all hardware and software information for the endpoints with the specified computer system IDs.

```
winvrmtree -i T293HCKDDG30QKX1GCPF0000052F \
5YFC7F221T9GCM71KJS20000051F
```

See Also

winupdatecsid

winvsig

Modifies the list of signatures in the configuration repository.

Syntax

winvsig -a [-t IBM | CUSTOM] -f *file_name*

winvsig -a -i

winvsig -a -n *name* -s *size* -d *description* -v *version*

winvsig -r -f *file_name*

winvsig -r -i

winvsig -r -n *name* -s *size* [-d *description*] [-v *version*] [-o *operating_system*]

Description

The **winvsig** command adds, modifies, or removes signatures from the configuration repository. During a scan that uses signature matching, Inventory matches the files found on the endpoint with signature data. If the name and size of the scanned file match the file name and size values of a signature, Inventory sends the signature data for the matched file to the configuration repository.

Note: On UNIX systems, the file name value is case sensitive. On NetWare, OS/2, and Windows systems, you must enter the file name in uppercase.

Options

-a Adds one or more signatures to the configuration repository, and modifies existing signatures. Each signature must have a unique file name and size. If the specified signature already exists, the information for the existing signature (for example, the description, version, or checksum value) is modified.

-d *description* Specifies a name or description for the software that is associated with the file you enter with the **-n** option. If the description includes spaces, enclose it in double quotation marks (" "). This information appears on the Inventory GUI in the **Software Name** column of the Signatures window.

-f *file_name* Specifies the name of a file that contains the data for one or more signatures. The signature data in the file must be in the following format:

```
<I>,name,size,description,version,quick_checksum,  
full_checksum,md5_checksum
```

The *quick_checksum*, *full_checksum*, and *md5_checksum* values are optional. You can specify one or more of the values. However, you must specify the values in the order shown. Moreover, if you skip one of the values, you must provide a placeholder in the form of two double quotation marks (""). For example, to specify values for *full_checksum* and *md5_checksum* but not *quick_checksum*, enter the following data:

```
<l>,name,size,description,version,"",  
full_checksum,md5_checksum
```

The signature file supports also the XML file format.

- i** Specifies that you want signature data to be read from standard input. You enter data by piping input from another command.
- n name** Specifies the name of the file that you want to use to identify the software product. This file is usually the primary executable for a software product, for example notepad.exe. The combination of the file name entered with the **-n** option and file size entered with the **-s** option must be unique in the list of signatures.

Specify a file name that matches the file name exactly, including case. If the file name differs across operating systems, you might need to create a signature for every operating system. For example, a file named SAMPLE.EXE is not the same as a file named sample.exe.

- r** Removes a signature from the list of current signatures. The signature that you specify with the **-r** option will not be used in signature scans. If history tracking is disabled, the signature is deleted from the configuration repository. If history tracking is enabled, the signature is *not* deleted from the configuration repository. Instead, the SIG_STATUS flag for the signature in the SWARE_SIG table is set to 0, which indicates that the signature is not current. Signatures with SIG_STATUS='0' are not used in signature scans, but they are kept so the history tracking feature can use them to match records in the H_MATCHED_SWARE table.

To delete the signature from the MATCHED_SWARE table, you need to run an inventory scan on the endpoints. The scan uses the new signatures and updates the MATCHED_SWARE table. On the contrary, the INST_SWARE_VIEW immediately shows the different results after running **winvid -r**, because it gets only signature entries having SIG_STATUS = '1'. The same behavior occurs when you add a new signature, for example by using the **winvsig -a** command. After running this command, the SWARE_SIG table is updated but the MATCHED_SWARE table will be updated only when an Inventory scan finds the signature file on an endpoint.

If you use history tracking and you want to delete signatures from the configuration repository, have a qualified database administrator (DBA) run a SQL command to delete them. For example, you can first run **winvsig -r** and then run the following command:

```
"delete from SWARE_SIG where SIG_STATUS='0'"
```

Note: Before you remove a signature, make sure that the signature you want to remove is not being used in any signature packages. To view the contents of signature packages, use the Inventory GUI or the **winvpackage** command.

- s size** Specifies the size of the file, in bytes, that you entered with the **-n** option. You can also specify any size using wildcards, by running the following command on Windows:

```
winvsig -a -n name -s * -d description -v version
```

and the following command on UNIX:

```
winvsig -a -n name -s '*' -d description -v version
```

-t IBM | CUSTOM

Specifies the type of imported signatures. The allowed values are IBM or CUSTOM. The default value is IBM. When specifying this type of signatures, the value of the IBM_SOURCE field in the SIGNATURE table is set in the following way:

0	CUSTOM
1	IBM

Note: When re-importing the signature catalog with a different value, the IBM_SOURCE field value is updated.

-v *version* Specifies the version of the software application that is associated with the file you entered with the **-n** option. If the version includes spaces, enclose it in double quotation marks (" ").

-o *operating_system*

Specifies the platform of the software application that is associated with the file you entered with the **-n** option. Supported values are: Windows, Linux, AIX, HP-UX, Solaris, i5/OS, JVM, Unix.

Authorization

RIM_update, admin, senior, or super

Notes

For all hexadecimal values that you supply using this command, you must include any zeroes:

Incorrect:

ff1234

Correct:

00ff1234

Examples

The following example adds a signature to the configuration repository for a version of the Windows NT WordPad software application.

```
winvsig -a -n WORDPAD.EXE -s 204800 -d "WordPad" -v 1.0
```

The following example removes from the configuration repository the signature data listed in a file named files.txt.

```
winvsig -r -f files.txt
```

See Also

winvfilter, winvpackage, wqueryinv

winvupdatecsid

Resolves the data for duplicate endpoints in the configuration repository.

Syntax

winvupdatecsid -c

winvupdatecsid -f *file_name* [-k]

winvupdatecsid -o *old_computer_system_id* -n *new_computer_system_id* [-k]

winvupdatecsid -q

winvupdatecsid -s

Description

The **winvupdatecsid** command enables you to check for and resolve duplicate endpoint records. Certain situations can cause one endpoint to have multiple computer system IDs, and therefore multiple records in the configuration repository. For example, when you reinstall the operating system of an existing endpoint and then reinstall the endpoint code, a new computer system ID might be issued for that endpoint. The next time you scan the endpoint, it will be treated as a new endpoint, and data for the endpoint could be stored in the configuration repository that duplicates existing data.

You can use the **winvupdatecsid** command to check for duplicate endpoint records in the configuration repository. To locate duplicate endpoint records, the **winvupdatecsid** command searches the configuration repository for duplicate MAC (medium-access control) addresses or serial numbers. Such a duplication would indicate that two sets of data exist for a single endpoint. If a duplicate value is found, a notice is sent to the Inventory notice group.

If duplicate records are found, you can use the **winvupdatecsid** command to resolve them. Duplicate records are resolved as follows:

- In operational data tables (tables that contain current data), data associated with the old computer system ID is deleted.
- In history tables, the old computer system ID is replaced with the new computer system ID. No data is deleted.

When resolving computer system IDs you can listing them individually or specify a text file that contains a list of computer system IDs to be resolved.

Options

- | | |
|----------------------------|---|
| -c | Turns off (clears) matching of duplicate endpoint records. This is the default option. |
| -f <i>file_name</i> | Specifies a text file that contains a list of computer system IDs to be resolved. This text file must meet the following criteria: <ul style="list-style-type: none">• You must list the old computer system ID followed by the new computer system ID.• Each pair of IDs must be separated by a space.• Each pair of IDs must appear on a separate line. |
| -k | Specifies that non-duplicate records are kept and the old computer |

system ID is replaced with the new one. Non-duplicate records are records in the configuration repository that are associated with the old computer system ID but not the new one. If you do not specify the **-k** option, non-duplicate records are deleted. This option does not affect data in history tables.

- n** *new_computer_system_id*
Specifies the new computer system ID.
- o** *old_computer_system_id*
Specifies the old computer system ID.
- q**
Returns a value that indicates whether inventory scans are configured to check for duplicate endpoint records. (You configure scans to do this using the **winvupdatecsid** command and **-s** and **-c** options.) The following values can be returned:
- YES** Inventory scans are configured to check for duplicate endpoint records.
- NO** Inventory scans are not configured to check for duplicate endpoint records. This is the default value.
- s**
Specifies that scans should check for duplicate endpoint records in the configuration repository. In other words, it turns on (sets) matching of duplicate endpoint records.

Authorization

super

Notes

- This command option applies to all inventory scans; you cannot apply this option to an individual inventory profile.
- The **winvupdatecsid** command does not locate duplicate records for pervasive devices. However, you can use this command to resolve duplicate computer system IDs for pervasive devices.

Examples

The following example deletes records related to the old computer system ID T293HCKDDG30QKX1GCPF0000052F in operational data tables. In history tables, the old computer system ID is replaced with the new value 5YFC7F221T9GCM71KJS20000051F. If non-duplicate records are found in operational data tables, the records are kept and the old computer system ID is replaced with the new one.

```
winvupdatecsid -o T293HCKDDG30QKX1GCPF0000052F -n \  
5YFC7F221T9GCM71KJS20000051F -k
```

The following example returns a value that indicates whether inventory scans are configured to check for duplicate endpoint records:

```
winvupdatecsid -q
```

The output for this command is as follows:

```
Multiple endpoint record matching is ON
```

The following example sets (turns on) matching of duplicate endpoint records:

```
winvupdatecsid -s
```

See Also

`winvrmnode`

wloadiso

Uploads data from isolated scans.

Syntax

```
wloadiso [-d {1 | 2 | 3}] -f file_name | -l list_file_name
```

Description

The **wloadiso** command sends data from isolated scans to the configuration repository. An isolated scan is a scan of a system that is not in your Tivoli region.

Using the **winviso** command, you can copy to an endpoint all of the files necessary to scan a disconnected system. You can then copy files from the endpoint to the disconnected system and run an isolated scan on the disconnected system using the **wepscan** command. This creates a .DAT file that contains the scan data. You must manually move the .DAT file from the disconnected system to the \$LCFROOT/inv/ISOLATED/depot directory on the endpoint. You can then run the **wloadiso** command from this directory to send the scan data to the configuration repository.

The **wloadiso** command stores the data from all .DAT files in the depot directory to a file called INV_ISO.DAT. The **wloadiso** command renames the existing .DAT files by appending .OLD to each file. SCS collects the INV_ISO.DAT file and send the data to the configuration repository.

Before you run **wloadiso**, you must first set up your environment to access shared libraries needed by the command. See “wepscan” on page 127 for this procedure.

See “Scanning disconnected systems” on page 56 for the procedure to run isolated scans. For more information about isolated scans, see “wepscan” on page 127 and “winvmgr” on page 161.

Options

- | | | | | | | | |
|--------------------------|---|----------|----------------------|----------|----------------------------------|----------|--|
| -d | Creates the log file WLOADISO.LOG. This log file contains debugging information about the wloadiso command. This log file is saved in the directory from which you ran wloadiso .

You must specify one of the following options with the -d option. These options specify the level of debugging information that is written to WLOADISO.LOG:

<table border="0"><tr><td style="padding-right: 10px;">1</td><td>Logs error messages.</td></tr><tr><td style="padding-right: 10px;">2</td><td>Logs error and warning messages.</td></tr><tr><td style="padding-right: 10px;">3</td><td>Logs error and warning messages and debugging information.</td></tr></table> | 1 | Logs error messages. | 2 | Logs error and warning messages. | 3 | Logs error and warning messages and debugging information. |
| 1 | Logs error messages. | | | | | | |
| 2 | Logs error and warning messages. | | | | | | |
| 3 | Logs error and warning messages and debugging information. | | | | | | |
| -f file_name | Specifies the name of the .DAT file to be sent to the configuration repository. You can specify more than one .DAT file. | | | | | | |
| -l list_file_name | Specifies the name of a file containing a list of .DAT files. | | | | | | |

Authorization

This command is run locally at the endpoint. No Tivoli authorization is required.

Examples

The following example logs error and warning messages to the WLOADISO.LOG file and sends the data from the KAT1.DAT and KAT2.DAT files to the configuration repository.

```
wloadiso -d 3 -f KAT1.DAT KAT2.DAT
```

See Also

wepscan, **winviso**

wmvinvcb

Moves the inventory callback object to the managed node that you specify.

Syntax

wmvinvcb [-t *timeout_value*] *managed_node*

Description

The **wmvinvcb** command moves the inventory callback object from the managed node on which it currently resides to the managed node that you specify.

You can move the inventory callback object to any managed node in your Tivoli region on which Inventory is installed.

You can run this command from any managed node that can run Inventory commands.

Options

-t *timeout_value*

Specifies the number of seconds after which the command will time out if it cannot contact the managed node to which you want to move the inventory callback object. The default value is 120 seconds.

managed_node

Specifies the managed node to which you want to move the inventory callback object.

Authorization

admin, **senior**, or **super**

Notes

Before moving the inventory callback object, make sure that no inventory scans are active.

Examples

The following example moves the inventory callback object to the managed node chengfu and specifies a timeout value of 240 seconds.

```
wmvinvcb -t 240 chengfu
```

See Also

wcrtinvcb

wmvinvdh

Moves the inventory data handler to the managed node that you specify.

Syntax

```
wmvinvdh [-t timeout_value] managed_node
```

Description

The **wmvinvdh** command moves the inventory data handler from the managed node on which it currently resides to the managed node that you specify. The settings of the inventory data handler are preserved when it is moved with the **wmvinvdh** command.

You can run this command from any managed node that can run Inventory commands.

Options

-t <i>timeout_value</i>	Specifies the number of seconds after which the command will time out if it cannot contact the managed node to which you want to move the inventory data handler. The default value is 120 seconds.
<i>managed_node</i>	Specifies the managed node to which you want to move the inventory data handler.

Authorization

admin, senior, or super

Notes

Before moving the inventory data handler, make sure that no inventory scans are active.

Inventory Server must be installed on the managed node to which you move the inventory data handler.

Examples

The following example moves the inventory data handler to the managed node mckinley and specifies a timeout value of 360 seconds.

```
wmvinvdh -t 360 mckinley
```

See Also

wcrtinvdh, wgetinvdh, wsetinvdh

wqueryinv

Queries the configuration repository for information about an endpoint in the Tivoli environment.

Syntax

```
wqueryinv [-d "delimiter"] [-f file_name] [-h host_name] [-n] [{-q @query_name | -s}] endpoint
```

Description

The **wqueryinv** command retrieves inventory scan information for an endpoint in the Tivoli environment. Using this command, you can run any query against the configuration repository that includes the TME_OBJECT_LABEL and TME_OBJECT_ID columns for any endpoint in your Tivoli environment.

Note: This command does not apply to queries of scan data for pervasive devices.

Options

- d "delimiter"** Specifies a delimiter for the query results file. Enclose the delimiting character in double quotation marks (" "). If you do not specify a delimiter, a comma will be used.
- f file_name** Specifies both the directory and the file name in which to save the results of the query. If you do not use the **-f** option, the query results are sent to standard output.
- h host_name** Specifies the name of the managed node on which you want to save the query results file. If you do not use the **-h** option, the query results are saved to the local system.
- n** Indicates that headers should not be included in the query results file.
- q @query_name** Specifies the query to be run. If you do not specify a query, the command runs the INVENTORY_HWARE query for the specified client.
- s** Runs the INVENTORY_SWARE query, which returns software information for the specified endpoint. If you do not use the **-s** or **-q** options, the INVENTORY_HWARE query is run.
- endpoint** Specifies the endpoint on which to run the query. Specify one of the following options:
 - o endpoint_oid**
The Tivoli object ID of the endpoint on which to run the query. You can view the object ID of an endpoint using the **wep** command and **ls** option. For more information about the **wep** command, see the *Tivoli Management Framework: Reference Manual*.
 - endpoint_name**
The host name of the endpoint on which to run the query.

Authorization

RIM_view, Query_view, Query_execute, Query_edit, senior, or super

Examples

The following example runs the query `HDISK_QUERY` for an endpoint named `alister`, specifies a semicolon (;) for the delimiter in the query results, sends the output to a file named `/tmp/inv.out` on a system named `mckinley`, and excludes headers from the file.

```
wqueryinv -n -d ";" -h mckinley -f /tmp/inv.out \  
-q @HDISK_QUERY alister
```

The following example runs the `INVENTORY_SWARE` query for an endpoint with the object ID `1059481916.3.517`, and sends the output to a file named `/tmp/inv.out` on the local system.

```
wqueryinv -s -f /tmp/inv.out -o 1059481916.3.517
```

See Also

`wrunquery` (in the *Tivoli Management Framework Reference Manual*)

wsetinvdh

Changes the settings of the inventory data handler.

Syntax

```
wsetinvdh [-d status_directory] [-n bundle_every_n_minutes] [-q  
bundle_every_n_targets] [-r max_RDBMS_retries] [-s {YES | NO}]  
[-t RDBMS_retry_delay_time] [-u {YES | NO}] [-z debug_level]
```

Description

The **wsetinvdh** command enables you to make changes to the inventory data handler that you created using the **wcrtinvdh** command. The options are identical to the **wcrtinvdh** command except that you do not specify a managed node. The **wsetinvdh** command automatically makes the changes to the inventory data handler instance @InvDataHandler:inv_data_handler in the Tivoli object database.

Options

-d *status_directory*

Specifies where the inventory data handler stores status information that can be restored in case of a system failure. By default, the location is \$DBDIR/inventory/stat_dir on the managed node where the inventory data handler resides.

-n *bundle_every_n_minutes*

Specifies the interval at which scan completion notifications are sent (when the **-n** option for the **wsetinvglobal** option is set to **BUNDLE**). If you set this number, Inventory sends a notice with a list of the targets on which scans have completed during the specified time period. The default value is 10 minutes.

If no scans complete in the specified time period, no notification is sent.

If you set this option to 0, notification occurs according to the value you set for the **-q** option of the **wcrtinvdh** or **wsetinvdh** command.

-q *bundle_every_n_targets*

Specifies the maximum number of targets in a bundle. A bundle is a group of targets about which status information is sent at one time. Status refers to the success or failure of a scan for each target of a particular inventory profile.

The default value is 10 targets.

This option, in combination with the **wsetinvglobal** command and the **-n** option, configures how status information is sent in bundles.

Notes:

1. If you set both **-q bundle_every_n_targets** and **-n bundle_every_n_minutes** to 0, no bundling occurs, and you are not notified until the scans on all targets are completed.
2. If you set both **-q bundle_every_n_targets** and **-n bundle_every_n_minutes** to a positive value, bundling occurs when either value (the specified number of targets or minutes) is reached.
3. Bundling occurs only when the **-n** option for the **wsetinvglobal** command is set to **BUNDLE**.

-r *max_RDBMS_retries*

Specifies the number of times the inventory data handler tries to write data to the RDBMS server. After the maximum number of retries is reached, a failure notification is sent.

The default value is 5.

-s *option*

Specifies whether the inventory data handler stores status information that can be restored in case of a system failure. Use one of the following options:

YES Status information is stored. This is the default option.

NO Status information is not stored.

-t *RDBMS_retry_delay_time*

Specifies a value from which to calculate the timeout period in seconds between retries of writes to a RIM object. This timeout period works according to the algorithm *timeout*retry_count*. For example, on the first retry, with a timeout value of 30 seconds, the algorithm sets the timer to 30 * 1 or 30 seconds. On the second retry, the timer sets to 30 * 2 or 1 minute.

The default value is 30 seconds.

-u {YES | NO}

Specifies whether the inventory data handler sends a notice to the Inventory notice group when an unsolicited update of scan data occurs. An unsolicited update is an update that is not initiated by distributing a scan to an endpoint remotely from another system. Examples include an endpoint-initiated scan using **wepscan** or an update of configuration repository data by an application or component other than Inventory. If you set this option to **YES**, a notice is sent. If you set this option to **NO**, a notice is not sent. By default, a notice is not sent.

-z Enables or disables the logging activity of the status collector. Returns the value of *debug_level* where *debug_level* can assume one of the following values:

0 Disables the logging activity of the status collector.

A value bigger than 0

Enables the logging activity of the status collector.

Authorization

admin, senior, super, or Inventory_edit

Notes

You use the **wcollect** command and the **-o** option to specify the number of output threads for the inventory data handler to write to RIM objects. It is strongly recommended that this value match the total number of RDBMS connections set for all RIM objects used by the inventory data handler. For example, if the inventory data handler uses two RIM objects to connect to the RDBMS, and each RIM object has 5 RDBMS connections, set the output threads for the inventory data handler to 10.

Examples

The following example changes the maximum number of targets in a notification bundle to nine and the RIM object retry delay time to 20 seconds.

```
wsetinvdh -q 9 -t 20
```

See Also

wcollect, wcrtinvdh, wgetinvdh, wmvinvdh, wsetinvglobal

wsetinvglobal

Sets the global properties for an inventory profile.

Syntax

```
wsetinvglobal [-d {CONFIG | ALL}] [-e seconds] [-f log_file_name] [-h  
log_file_host] [-l {OFF | option_list}] [-m {Y | N}] [-n {IMMEDIATE | BUNDLE |  
DONE}] [-s {Y | N}] [-t {NONE | SUCCESS | FAIL | ALL}] [-u {REPLACE |  
DIFFS}] [-w {Y | N}] profile_name
```

Description

The **wsetinvglobal** command enables you to set the global properties for an inventory profile. This command sets the following properties:

- Distribution options—whether the inventory profile distributes the configuration file, runs the scan, or both; the timeout value for the scan; whether a distribution to a mobile endpoint is visible to the mobile user; whether the specified profile can be distributed to targets that are not subscribers to the profile manager that contains the profile; and whether the profile distribution powers up Wake on LAN enabled systems. Unless otherwise specified, these options do not apply to scans of pervasive devices.
- Data options—whether the scan sends only new or changed data to the configuration repository, or replaces existing data in the configuration repository with the data gathered by the scan. Unless otherwise specified, these options do not apply to scans of pervasive devices.
- Status options—where and under what conditions scan status notification is sent. You can also specify the name and path of the status log file and the managed node on which it is stored. These options apply to scans of pervasive devices.

Options

-d *option*

Specifies the distribution option. Use one of the following options:

CONFIG

Distributes the configuration file only.

ALL Distributes the configuration file and runs the scan. This is the default option.

-e *seconds*

Specifies the number of seconds that the profile attempts to scan an endpoint before it times out. The default value is 1800 seconds.

-f *log_file_name*

Specifies the location and name of a file to which you want to log Inventory status information. The path must be a full path, not a relative path. If a null string ("") is specified, the log is written to a file named *\$TMPDIR/inv_scan_nn.log* where *nn* is the scan ID of the scan and *\$TMPDIR* is a temporary directory. The temporary directory is usually one of the following paths:

- For UNIX: /tmp, /usr/tmp, or /var/tmp
- For Windows NT: c:\temp

-h *log_file_host*

Specifies the name of a managed node to which you want to save

Inventory status information. If a null string ("") is specified, the log file is saved on the managed node on which the inventory data handler was created.

-I {OFF | *option_list*}

Specifies the location or locations to which notifications are sent. Use one or more of the following options:

NOTICE_GROUP

Sends status information to the Inventory notice group.

LOG_FILE

Sends status information to the log file specified with the **-f** and **-h** options.

TEC Sends information to the Tivoli Enterprise Console console as specified by the **-t** option. See the **-t** option for more details.

OFF Sends no status.

The default value is NOTICE_GROUP. You can write to more than one location by specifying multiple options. Separate the options with a comma (,).

Note: If an error occurs when Inventory tries to write to a log file, an error message is sent to the Inventory notice group, and all status information is written only to the Inventory notice group.

-m *option*

Specifies whether a distribution to a mobile or roaming endpoint (such as a laptop) is visible to the roaming user. If a distribution is visible, the roaming user can defer the distribution. If a distribution is hidden, it is automatically distributed to the roaming endpoint without notifying the mobile user.

This option performs the same function as the **wdistinv** command and the **hidden** keyword. If this option and the **hidden** keyword are set to conflicting values, this option overrides the **hidden** keyword.

Specify one of the following options:

Y The distribution is visible.

N The distribution is hidden. This is the default value.

-n *option*

Specifies when notification is sent when a scan completes on each target. Choose one of the following options:

IMMEDIATE

Notification is sent when the scan on each target completes.

BUNDLE

Notification is sent periodically, based on the settings for the inventory data handler. Refer to the **wsetinvdh** commands for more information on bundling.

DONE

Notification is sent only when scans on all targets are complete.

The default value is DONE.

-s *option*

Specifies whether the specified profile can be distributed to targets

(endpoints, profile managers, or resource groups of users or pervasive devices) that are not subscribers to the profile manager that contains the inventory profile. Specify one of the following options:

- Y** The profile can be distributed to targets that are not subscribers to the profile manager that contains the inventory profile.
- N** The profile cannot be distributed to non-subscribers. This is the default option.

-t option

Sets the circumstances under which notification is sent for the specified profile. Specify one of the following options:

NONE

Notification is not sent.

SUCCESS

Notification is sent for targets on which scans complete successfully. If the **-I** option is set to **TEC**, a Tivoli Enterprise Console event is sent at the end of the distribution only if all targets are successful.

FAIL Notification is sent for targets on which scans failed. If the **-I** option is set to **TEC**, a Tivoli Enterprise Console event is sent at the end of the distribution if one or more targets failed.

ALL Notification is sent for all targets. If the **-I** option is set to **TEC**, two Tivoli Enterprise Console events are always sent at the start and end of the distribution.

The default value is **ALL**.

Note: No notification is sent to the Tivoli Enterprise Console if the scan is canceled. Canceling the scan does not affect whether notification is sent to the notice group or log file.

-u option

Specifies the data option. Use one of the following options:

REPLACE

Replaces existing data in the configuration repository with the data gathered by this scan.

DIFFS Sends to the configuration repository only the data that has been added or changed since the last scan. Data from previous scans that is not present in the current scan is deleted from the configuration repository. No record of changes is kept unless history tables are used. This is the default option.

-w option

Specifies whether the profile distribution will automatically power up a system that has a Wake on LAN network card installed and enabled. Specify one of the following options:

- Y** The profile distribution will power up the system. This is the default option.
- N** The profile distribution will not power up the system.

profile_name

The profile for which you want to set the global properties. Enter the profile name in the format **@InventoryConfig:profile_name**.

Authorization

admin, senior, super, or Inventory_edit

Notes

If you want to send Inventory events to one or more Tivoli Enterprise Console servers, you must create an Inventory rule base and import the BAROC files into Tivoli Enterprise Console. Inventory attempts to send events to the Tivoli Enterprise Console servers in the order that they are specified. You do not need to use the following procedure if you do not want to send events to a Tivoli Enterprise Console server.

To import the BAROC files into Tivoli Enterprise Console, you must perform the following steps:

1. Create a new rule base named Inventory.
2. Copy the default classes from the Tivoli Enterprise Console rules base to the new rule base named Inventory.
3. Import the Inventory BAROC file `tecad_inv.baroc` from the `$BINDIR/TME/INVENTORY` directory.
4. Compile the Inventory rule base.
5. Load the Inventory rule base into the Tivoli Enterprise Console server.
6. Stop the Tivoli Enterprise Console server.
7. Restart the Tivoli Enterprise Console server.

See the *Tivoli Enterprise Console: Rule Builder's Guide* for details about this procedure.

If your environment is complex (containing multiple Tivoli regions or Tivoli Enterprise Console servers), perform the following steps:

1. If the Tivoli Enterprise Console server and the system on which the Inventory component is installed are in different Tivoli regions, make the EventServer class visible from Inventory by running the following command before connecting the two Tivoli regions:

```
wregister -i -r EventServer
```

2. Copy the BAROC file as described in the previous procedure.
3. If more than one EventServer is present in the environment, create the file `$DBDIR/tecad_inv.conf` on the inventory server. This file should contain the following line:

```
ServerLocation=EventServer#server-region
```

where *server-region* is the event server to which the events must be sent.

4. Run the **wsetinvglobal** command. See “wsetinvglobal” on page 187 for more information.

Examples

The following example configures the inventory profile `SW_Profile` to replace existing data in the configuration repository with the data gathered by the scan and send status information as each target completes.

```
wsetinvglobal -u REPLACE -n IMMEDIATE @InventoryConfig:SW_Profile
```

The following example configures the inventory profile SW_Profile to send notification to the log file, the Inventory notice group, and the Tivoli Enterprise Console console.

```
wsetinvglobal -l LOG_FILE,NOTICE_GROUP,TEC \  
@InventoryConfig:SW_Profile
```

See Also

wgetinvglobal, winvupdatecsid, wsetinvpcfiles, wsetinvpchw, wsetinvpcsw, wsetinvunixfiles, wsetinvunixhw, wsetinvunixsw

wsetinvpcfiles

Specifies options for files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of PC endpoints.

Syntax

```
wsetinvpcfiles [-b script_path] [-t {INCLUDE | EXCLUDE} -d {+ | -}directory] [-e {INCLUDE | EXCLUDE}] [-f {+ | -}file_type...] [-m {+ | -}MIF_path ...] [-s script_path] profile_name
```

Description

The **wsetinvpcfiles** command specifies options for files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of PC endpoints. Using this command, you can set the following options:

- Files or file types to include or exclude during the scan.
- Directories to include or exclude during the scan.
- Scripts to be run during the profile distribution before the scan is performed or after the scan completes.
- Custom MIF files to be collected during the profile distribution after the scan completes.

Options

-b *script_path*

Specifies the path and name of a script that you want to run on an endpoint during the profile distribution before the scan begins. You can specify an absolute or relative path. You can specify text files only. You cannot specify binary files.

Notes:

1. If you edit a script after adding it, you must then add the script again using the **-b** option. If you do not, the revised script will not be run.
2. To remove a script, specify double quotation marks with no characters in between ("").

-d *option directory*

Specifies a directory that you want to add to, or remove from, the list of directories to include or exclude during the scan. By default, the `*/tmp` and `*/temp` directories are excluded from scans. You must precede this option with the **-t** option to indicate whether the directory is to be included or excluded during the scan.

Use one of the following options. Do not type a space between the option and the directory you specify.

- + Adds this directory to the list.
- Removes this directory from the list.

When specifying a directory to *include* in a scan, observe the following guidelines:

- You cannot use wildcard characters. All characters are treated as literals, including asterisks (*) and question marks (?).
- If you do not specify a drive (for example `c:/` or `sys:/`), Inventory scans the specified directory on all scanned drives.

- All directories are considered absolute paths. For example, if you specify `abc`, `c:/abc` will be scanned, but `c:/temp/abc` will not.

When specifying a directory to *exclude* from a scan, observe the following guidelines:

- You can use an asterisk as a wildcard at the beginning of a path name, for example `*/abc`. This example excludes any path that contains a directory named `abc`.
- You can use an asterisk as a wildcard after a drive designation, for example `c:*/abc`, or `sys:*/abc`. These examples exclude any path on the specified drive that contains a directory named `abc`.
- You cannot use a wildcard within a path name, for example `*temp`, `temp*`, `/*/temp`, `c:/*/temp`, `te*mp`, or `/user*/temp`). All characters within paths are treated as literals.
- You cannot use a wildcard in place of a drive letter.
- You must enclose directories that have a wildcard asterisk character in double quotation marks (" ").

-e option

Specifies whether the list of files or file types created with the `-f` option is included or excluded during the scan. Use one of the following options:

INCLUDE

Includes the files or file types in the scan. This is the default option.

EXCLUDE

Excludes the files or file types from the scan.

-f option *file_type*

Specifies a file or file type that you want to add to, or remove from, the list of files or file types to include or exclude during the scan.

Use one of the following options. Do not type a space between the option and the value it modifies.

- +** Adds the file or file type to the list.
- Removes the file or file type from the list.

Observe the following guidelines:

- You can specify a file by typing the file name, or a file type by using an asterisk (*) as a wildcard.
- You can place the asterisk at the beginning or end of the text string, or both (for example `*.bat`, `abc.*`, `*abc.bat`, `abc.ba*`, `*abc`, `abc*`, `*abc*`).
- You cannot place the asterisk within the text string (for example `ab*c`, `ab*c.bat`, `abc.*at`).
- You must enclose file types in double quotation marks (" ").

You can repeat the `-f` option multiple times to add or remove multiple files or file types.

-m option *MIF_path*

Specifies the path and name of a MIF file that you want to add to, or remove from, the list of MIF files to collect during the profile distribution. Specify the absolute path.

Use one of the following options. Do not type a space between the option and the value it modifies.

- + Add the MIF file to the list.
- Remove the MIF file from the list.

For *MIF_path*, you must specify the full path and name of the MIF file. You can repeat the **-m** option multiple times to add or remove multiple MIF files.

-s *script_path*

Specifies the path and name of a script that you want to run on an endpoint after the scan completes and before MIF files are read. You can specify an absolute or relative path. You can specify text files only. You cannot specify binary files.

Notes:

1. If you edit a script after adding it, you must then add the script again using the **-s** option. If you do not, the revised script will not be run.
2. To remove a script, specify double quotation marks with no characters in between ("").

-t *option*

Specifies whether the directory specified with the **-d** option that follows this option is to be included or excluded during software scans of PC systems. Use one of the following options:

INCLUDE

Includes the specified directory in the scan.

EXCLUDE

Excludes the specified directory from the scan.

profile_name

Specifies the inventory profile that you want to edit. Enter the profile name in the format **@InventoryConfig:profile_name**.

Authorization

admin, senior, super, or Inventory_edit

Notes

The directory lists you specify using the **-t** and **-d** options must be mutually exclusive. Do not list the same directory in both the include and exclude lists.

Although you can create separate lists of directories to be excluded and directories to be included, you cannot do this with the list of files or file types (the **-f** option). For files and file types, you use the **-e** option to specify whether the entire list will be either included or excluded during a scan.

Examples

The following example adds the `/cache` directory to the list of directories to exclude during a software scan. This change is applied to an inventory profile named `SW_Profile`.

```
wsetinvpcfiles -t EXCLUDE -d +/cache @InventoryConfig:SW_Profile
```

The following example removes the `/tmp` and `/temp` directories from the list of directories to exclude during a software scan. This change is applied to an inventory profile named `SW_Profile`.


```
wsetinvpcfiles -t EXCLUDE -d -"*/tmp" -d-"*/temp" \  
@InventoryConfig:SW_Profile
```

The following example removes the /usr directory from the list of directories to include during a software scan. This change is applied to an inventory profile named SW_Profile.

```
wsetinvpcfiles -t INCLUDE -d -/usr @InventoryConfig:SW_Profile
```

The following example specifies that the list of files and file types specified by the -f option should be excluded from the scan. This change is applied to an inventory profile named SW_Profile.

```
wsetinvpcfiles -e EXCLUDE @InventoryConfig:SW_Profile
```

The following example adds the .bat and .sys file types to the files to be filtered during the scan. These files are either included or excluded during the scan depending on the setting of the -e option. This change is applied to an inventory profile named SW_Profile.

```
wsetinvpcfiles -f +"*.bat" -f +"*.sys" @InventoryConfig:SW_Profile
```

The following example specifies that the file TEST.BAT is run during the profile distribution. This change is applied to an inventory profile named SW_Profile.

Note: Because the path for TEST.BAT is not specified in this example, TEST.BAT must be in the directory from which this command is run.

```
wsetinvpcfiles -s "TEST.BAT" @InventoryConfig:SW_Profile
```

See Also

wgetinvpcfiles,, wgetinvunixfiles, wsetinvglobal,, wsetinvpchw,, wsetinvpcsw,, wsetinvunixfiles,, wsetinvunixhw,, wsetinvunixsw

wsetinvpchw

Sets the options for the hardware scan of PC endpoints.

Syntax

wsetinvpchw **-a** *component* [**-a** *component*]... *profile_name*

wsetinvpchw [**-d** {**Y** | **N**}] [**-s** {**Y** | **N**}] *profile_name*

wsetinvpchw **-r** *component* [**-r** *component*]... *profile_name*

wsetinvpchw [**-t** {**Y** | **N**}] [**-u** {**Y** | **N**}] *profile_name*

Description

The **wsetinvpchw** command sets the options for the hardware scan of a PC endpoint. You can specify the following options:

- Whether to run the Tivoli hardware scanner during scans of PC endpoints.
- Whether the scan data is sent to the configuration repository or stored on the endpoint to be collected later.
- Which hardware components you want to scan for.

Options

-a *component* Adds the specified component to the list of hardware components to scan for. Not all options are available from every platform type.

These component values are not case sensitive. Moreover, you do not need to enter the entire component name. You can enter only the number of characters necessary to uniquely identify that component.

You can specify multiple components by repeating **-a** *component* multiple times.

You can specify the following components:

IPAddress

Lists the TCP/IP configuration information.

IPXAddress

Lists IPX configuration information.

Keyboard

Describes the connected keyboard, if any.

Lpar Lists the number of processors allocated on each logical partition.

Memory

Describes the installed memory. This option can list details such as the amount of actual memory, virtual memory, and paging space.

MemoryModules

Lists each available memory slot and provides information about installed memory module size and type.

Modem

Lists the installed modems recognized by the operating system and properties of the modems.

NetworkAdapter

Describes the installed network adapter or adapters.

OperatingSystem

Describes only the operating system running at the time of the scan. This option does not list all installed operating systems on dual-boot systems.

Partition

Describes each of the drive partitions on a system. A system might have only one partition. General information listed can include the drive mapping, the size and available size of each partition, the file system, and the media type.

PCIDevice

Lists the expansion cards installed on the PCI and AGP buses.

PCSystemParams

Lists PC-specific information such as user name, domain name, and BIOS characteristics.

PointingDevice

Describes the type of pointing device (such as a mouse or trackball) attached to the system.

Printer

Describes any local and network printers connected to the system.

Processor

Describes the processor or processors installed on the system.

ServiceInfo

Lists the Windows services running on the system. This component applies to Windows platforms only.

SMBIOS

Lists system-specific identification information, such as manufacturer, model, and serial number of the system, chassis, and BIOS. This information can be collected from SMBIOS-compliant Linux systems only.

Storage

Describes all local storage devices such as hard-disk drives, CD-ROM drives, and removable-media drives. This option does not list network drives.

USBDevice

Describes all USB devices and hubs attached to the system at the time of the scan.

Video Describes the video adapter and the current video settings. This option does not list the maximum possible settings.

-d option

Specifies whether to run the DMI scanner. Use one of the following options:

	Y	Runs the DMI scanner.
	N	Does not run the DMI scanner.
-r component		Removes the specified component from the list of hardware components to scan for. The possible values for <i>component</i> are the same as for the -a option. You can specify multiple components by repeating -r component multiple times.
-s option		Specifies whether to return DMI information to the configuration repository. Use one of the following options: Y Sends the DMI scan data to the configuration repository. N Does not send the DMI scan data to the configuration repository. Stores the scan data on the endpoint.
-t option		Specifies whether to run the Tivoli hardware scanner. Use one of the following options: Y Runs the Tivoli hardware scanner. N Does not run the Tivoli hardware scanner.
-u option		Specifies whether to update the configuration repository with the scan data. Use one of the following options: Y Sends the scan data to the configuration repository. N Does not send the scan data to the configuration repository. Stores the scan data on the endpoint.
profile_name		The profile that you want to edit. Enter the profile name in the format @InventoryConfig:profile_name .

Authorization

admin, senior, super, or Inventory_edit

Notes

You must collect processor, memory, and operating system data if you want to use the INVENTORY_HWARE query.

Examples

The following command configures the inventory profile HW_Profile to run the Tivoli hardware scanner, exclude processor information from the scan, and include memory and drive partition information in the scan.

```
wsetinvpchw -r Processor -a Memory -a Partition -t Y \
@InventoryConfig:HW_Profile
```

The following command configures the inventory profile HW_Profile to run the DMI scanner and return the DMI scan data to the configuration repository.

```
wsetinvpchw -d Y -s Y @InventoryConfig:HW_Profile
```

See Also

wgetinvpchw, wsetinvglobal, wsetinvpcfiles, wsetinvpcsw, wsetinvunixfiles, wsetinvunixhw, wsetinvunixsw

wsetinvpcsw

Sets the options for software scans of PC endpoints.

Syntax

```
wsetinvpcsw [-b {SCAN | UPDATE | BOTH | NO}] [-c {QUICK | FULL | MD5 | NONE}] [-f {Y | N}] [-h {SCAN | UPDATE | BOTH | NO}] [-r {SCAN | UPDATE | BOTH | NO}] [-s {SCAN | UPDATE | BOTH | NO}] [-x {Y | N}] [-m {Y | N}] [-d {Y | N}] [-n file_name]]] profile_name
```

Description

The **wsetinvpcsw** command sets the options for software scans of PC endpoints. You can specify whether to scan the files on a PC endpoint, scan the Windows registry for information about installed programs, or both.

For file scans, you can configure whether the scan performs the following actions:

- Scans for basic file information.
- Sends scan data to the configuration repository.
- Generates checksum values for scanned files.
- Applies a custom filter to scans for basic file information.
- Scans files for header information.
- Scans for signature data.
- Scans executable files only.

For Windows registry scans, you can configure whether to send scan data to the configuration repository.

Options

-b option

Specifies whether to scan for basic file information. This type of scan collects the name and path of all scanned files. Other information, such as date of creation, modification, and last access, is collected on some platforms. Use one of the following options:

SCAN Scans for basic file information, and then stores the scan data on the endpoint.

UPDATE

Does not scan. Sends the data from the last scan for basic file information to the configuration repository.

BOTH Scans for basic file information, and then sends the scan data to the configuration repository.

NO Does not scan for basic file information, and does not send data to the configuration repository. This is the default option.

-c option

Specifies whether to generate a checksum value for scanned files. Use one of the following options:

QUICK

Generates the Quick checksum value. This option produces a 32-bit value based only on the first 1 KB of each file. This is the quickest checksum option, but produces the least reliable value.

- FULL** Generates the Full checksum value. This option produces a 32-bit value based on the full contents of each file. This option takes longer than the QUICK option, but produces a more reliable value.
- MD5** Generates the MD5 checksum value. This option produces a 128-bit value based on the full contents of each file. The MD5 option provides a more reliable value than the QUICK or FULL options, but takes longer to complete.
- NONE**
Does not generate checksum values. This is the default option.
- f option** Specifies whether to filter the scan for basic file information using a custom list of files. You create this list of files using the Custom Filter window or **winvfilter** command. When this option is set to Y, a scan for basic file information includes only the files specified in the custom filter. Use one of the following options:
- Y** Filters scans for basic file information using the custom filter.
- N** Does not filter scans for basic file information using the custom filter. This is the default option.
- h option** Specifies whether to search the header of scanned files for the company name, product name, and product version. This option has no effect on NetWare and OS/2 systems. Use one of the following options:
- SCAN** Scans for header information, and then stores the scan data on the endpoint.
- UPDATE**
Does not scan. Sends the data from the last scan for header information to the configuration repository.
- BOTH** Scans for header information, and then sends the scan data to the configuration repository.
- NO** Does not scan for header information, and does not send data to the configuration repository. This is the default option.
- r option** Specifies whether to scan the Windows registry for information about installed products. This option has no effect on NetWare and OS/2 systems. Use one of the following options:
- SCAN** Scans the Windows registry, and then stores the scan data on the endpoint.
- UPDATE**
Does not scan. Sends the data from the last Windows registry scan to the configuration repository.
- BOTH** Scans the Windows registry, and then sends the scan data to the configuration repository. This is the default option.
- NO** Does not scan the Windows registry, and does not send data to the configuration repository.
- s** Specifies whether to scan using signatures. In this type of scan, the

name and size of each scanned file is compared against the list of signatures. Use one of the following options:

SCAN Scans using signatures, and then stores the scan data on the endpoint.

UPDATE

Does not scan. Sends the data from the last signature scan to the configuration repository.

BOTH Scans using signatures, and then sends the scan data to the configuration repository.

NO Does not scan using signatures, and does not send data to the configuration repository. This is the default option.

-x option

Specifies whether to filter a file scan by scanning for executable files only. For PC systems, an executable file is defined as a file with one of the following extensions: .EXE, .COM, .CMD, .BAT. Use one of the following options:

Y Scans for executable files only.

N Does not scan only for executable files. This is the default option.

-m

Sets information related to the patch scan setting. The following values can be returned:

YES The patch scan is enabled.

NO The patch scan is not enabled.

-a

Specifies if the patch scan uses the ApprovedItems.txt file. The following values can be returned:

Y The patch scan uses the ApprovedItems.txt file.

N The patch scan does not use the ApprovedItems.txt file.

-d Y|N

Specifies whether the swsigs.txt file must be downloaded to the endpoint. The default value is N, which means that the file is downloaded to the endpoint with every profile distribution. To prevent the file from being downloaded, set the option to Y. You can use the **-n** option to select a different file to be downloaded.

-n file_name

Specifies the name of the file to be downloaded to the endpoint. You can choose the following two files:

swsigs.txt

Contains Inventory signatures.

wsusscan.cab

Contains the security policy catalog.

This option can be used only with the **-d** option.

profile_name

Specifies the inventory profile that you want to edit. Enter the profile name in the format **@InventoryConfig:profile_name**.

Authorization

admin, senior, super, or Inventory_edit

Examples

The following command configures the inventory profile SW_Profile to generate an MD5 checksum of scanned files, scan the Windows registry, and send the registry scan data to the configuration repository.

```
wsetinvpcsw -c MD5 -r BOTH @InventoryConfig:SW_Profile
```

See Also

wgetinvpcsw, wsetinvglobal, wsetinvpcfiles, wsetinvpchw, wsetinvunixfiles, wsetinvunixhw, wsetinvunixsw

wsetinvpvdconfig

Enables or disables scans of pervasive devices for configuration data.

Syntax

```
wsetinvpvdconfig -t {Y | N} profile_name
```

Description

You use the **wsetinvpvdconfig** command to specify whether a profile scans pervasive devices for configuration data. Examples of configuration data include language and alarm settings and time and date format.

Options

-t	Specifies whether to scan devices for configuration data. Specify one of the following options: Y Scan devices for configuration data. N Do not scan devices for configuration data.
<i>profile_name</i>	Specifies the inventory profile that you want to edit. Enter the profile name in the format @InventoryConfig:profile_name .

Authorization

admin, senior, super, or Inventory_edit

Examples

The following example enables the profile named MyProfile to collect configuration data from devices:

```
wsetinvpvdconfig -t Y @InventoryConfig:MyProfile
```

See Also

wgetinvpvdconfig, wgetinvpvdhw, wgetinvpvdsw, wsetinvpvdhw, wsetinvpvdsw

wsetinvpvdhw

Enables or disables hardware scans of pervasive devices.

Syntax

```
wsetinvpvdhw -t {Y | N} profile_name
```

Description

You use the **wsetinvpvdhw** command to specify whether a profile scans pervasive devices for hardware data.

Options

-t	Sets the hardware scan options for devices. Specify one of the following options: Y Scan devices for hardware data. N Do not scan devices for hardware data.
<i>profile_name</i>	Specifies the inventory profile that you want to edit. Enter the profile name in the format @InventoryConfig:profile_name .

Authorization

admin, **senior**, **super**, or **Inventory_edit**

Examples

The following example configures the profile named MyProfile to collect hardware data during scans of devices:

```
wsetinvpvdhw -t Y @InventoryConfig:MyProfile
```

See Also

wgetinvpvdconfig, **wgetinvpvdhw**, **wgetinvpvdsw**, **wsetinvpvdconfig**, **wsetinvpvdsw**

wsetinvpvdsw

Enables or disables software scans of pervasive devices.

Syntax

```
wsetinvpvdsw -t {Y | N} profile_name
```

Description

You use the **wsetinvpvdsw** command to specify whether a profile scans pervasive devices for software data.

Options

-t	Sets the software scan options for devices. Specify one of the following options: Y Scan devices for software data. N Do not scan devices for software data.
<i>profile_name</i>	Specifies the inventory profile that you want to edit. Enter the profile name in the format @InventoryConfig:profile_name .

Authorization

admin, **senior**, **super**, or **Inventory_edit**

Examples

The following example configures the profile named MyProfile to collect software data during scans of devices:

```
wsetinvpvdsw -t Y @InventoryConfig:MyProfile
```

See Also

wgetinvpvdconfig, **wgetinvpvdhw**, **wgetinvpvdsw**, **wsetinvpvdconfig**, **wsetinvpvdhw**

wsetinvswd

Controls whether Inventory is enabled to integrate with Software Distribution.

Syntax

wsetinvswd {y | n}

Description

The **wsetinvswd** command controls whether Inventory is enabled to integrate with Software Distribution. Software Distribution must also be enabled to integrate with Inventory using the **wswdmgr** command. For more information about the **wswdmgr** command, see the *Reference Manual for Software Distribution*.

When the Inventory and Software Distribution components have been enabled to integrate with each other, you can perform the following tasks:

- When you create a software package, you can optionally specify to have a signature added to the Inventory signature collection if the signature does not currently exist. This signature can represent the software distributed by the software package. Within a software package definition (SPD) file, you can define which files are to be considered signatures. When the package is imported into the Tivoli environment and built, Software Distribution creates the new signature file entries in the configuration repository. This capability is especially useful when distributing in-house software for which Inventory has no signatures.
- Automatically update the list of installed software when Software Distribution successfully delivers a software package.
- Automatically update the software catalog on the endpoint if a software signature-matching scan is requested.

Note: If the Software Distribution component is not installed on the endpoint, Inventory stores the software package information in the file
\$LCF_BASE_DIR/inv/SCAN/swdpackage.

For more information about these tasks, see the *User's Guide for Software Distribution*.

Options

- | | |
|---|--|
| y | Enables integration with Software Distribution. |
| n | Disables integration with Software Distribution. This is the default option. |

Authorization

admin, senior, super, or Inventory_edit

Examples

The following example enables Inventory to integrate with Software Distribution:

```
wsetinvswd y
```

See Also

wgetinvswd, **wswdmgr** in the *Reference Manual for Software Distribution*.

wsetinvunixfiles

Specifies options for files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of UNIX and OS/400 endpoints.

Syntax

```
wsetinvunixfiles [-b script_path] [-e {INCLUDE | EXCLUDE}] [-f {+ |  
-} file_type]... [-m {+ | -} MIF_path]... [-s script_path] [-t {INCLUDE | EXCLUDE}  
-d {+ | -} directory] profile_name
```

Description

The **wsetinvunixfiles** command specifies the following options for scans of UNIX and OS/400 endpoints:

- Files or file types to include or exclude during the scan.
- Directories to include or exclude during the scan.
- Scripts to be run during the profile distribution before the scan is performed or after the scan completes.
- Custom MIF files to be collected during the profile distribution after the scan completes.

Options

-b *script_path*

Specifies the path and name of a script that you want to run on an endpoint during the profile distribution before the scan begins. You can specify an absolute or relative path. You can specify text files only. You cannot specify binary files.

Notes:

1. If you edit a script after adding it, you must then add the script again using the **-b** option. If you do not, the revised script will not be run.
2. To remove a script, specify double quotation marks with no characters in between ("").

-d *option directory*

Specifies a directory that you want to add to, or remove from, the list of directories to include or exclude during the scan. This option does not apply to OS/400 endpoints. By default, the `*/tmp` directory is excluded from scans. You must precede this option with the **-t** option to indicate whether the directory is to be included or excluded during the scan.

Use one of the following options. Do not type a space between the option and the value it modifies.

+ Adds this directory to the list.

- Removes this directory from the list.

When specifying a directory to *include* in a scan, observe the following guidelines:

- You cannot use wildcard characters. All characters are treated as literals, including asterisks (*) and question marks (?).
- All directories are considered absolute paths. For example, if you specify `abc`, `/abc` will be scanned, but `/tmp/abc` will not.

When specifying a directory to *exclude* from a scan, observe the following guidelines:

- You can use an asterisk as a wildcard at the beginning of a path name, for example */abc. This example excludes any path that contains a directory named abc.
- You cannot use a wildcard within a path name, for example *temp, temp*, /*/tmp, t*mp, or /usr*/tmp). All characters within paths are treated as literals.
- You must enclose directories that have an asterisk character (*) in double quotation marks (" ").

-e option

Specifies whether the list of files or file types created with the -f option is included or excluded during the scan. Use one of the following options:

INCLUDE

Includes the files or file types in the scan. This is the default option.

EXCLUDE

Excludes the files or file types from the scan.

-f option *file_type*

Specifies a file or file type that you want to add to, or remove from, the list of files or file types to include or exclude during the scan. This option does not apply to OS/400 endpoints.

Use one of the following options. Do not type a space between the option and the value it modifies.

- + Adds the file or file type to the list.
- Removes the file or file type from the list.

Observe the following guidelines:

- You can specify a file by typing the file name, or a file type by using an asterisk (*) as a wildcard.
- You can place the asterisk at the beginning or end of the text string, or both (for example *.txt, abc.*, *abc.txt, abc.tx*, *abc, abc*, *abc*).
- You cannot place the asterisk within the text string (for example ab*c, ab*c.txt, abc.t*t).
- You must enclose file types in double quotation marks (" ").

You can repeat the -f option multiple times to add or remove multiple file types.

-m option *MIF_path*

Specifies the path and name of a MIF file that you want to add to, or remove from, the list of MIF files to collect during the profile distribution. Specify the absolute path.

Use one of the following options. Do not type a space between the option and the value it modifies.

- + Adds the MIF file to the list.
- Removes the MIF file from the list.

For *MIF_path*, you must specify the full path and name of the MIF file. You can repeat the -m option multiple times to add or remove multiple MIF files.

-s *script_path*

Specifies the path and name of a script that you want to run on an

endpoint after the scan completes and before MIF files are read. You can specify an absolute or relative path. You can specify text files only. You cannot specify binary files.

Notes:

1. If you edit a script after adding it, you must then add the script again using the **-s** option. If you do not, the revised script is not run.
2. To remove a script, specify double quotation marks with no characters in between ("").

-t option

Specifies whether the directory specified with the **-d** option that follows this option is to be included or excluded during software scans of UNIX systems. Use one of the following options:

INCLUDE

Includes the specified directory in the scan.

EXCLUDE

Excludes the specified directory from the scan.

profile_name

The name of the inventory profile that you want to edit. Enter the profile name in the format **@InventoryConfig:profile_name**.

Authorization

admin, senior, super, or Inventory_edit

Notes

The directory lists you specify using the **-t** and **-d** options must be mutually exclusive. Do not list the same directory in both the include and exclude lists.

Although you can create separate lists of directories to be excluded and directories to be included, you cannot do this with the list of files or file types (the **-f** option). For files and file types, you use the **-e** option to specify whether the entire list will be either included or excluded during a scan.

Examples

The following example adds the `/cache` directory to the list of directories to exclude during a software scan. This change is applied to an inventory profile named `SW_Profile`.

```
wsetinvunixfiles -t EXCLUDE -d +/cache @InventoryConfig:SW_Profile
```

The following example removes the `/tmp` directory from the list of directories to exclude during a software scan. This change is applied to an inventory profile named `SW_Profile`.

```
wsetinvunixfiles -t EXCLUDE -d -"/tmp" \
@InventoryConfig:SW_Profile
```

The following example removes the `/usr` directory from the list of directories to include during a software scan. This change is applied to an inventory profile named `SW_Profile`.

```
wsetinvunixfiles -t INCLUDE -d -/usr @InventoryConfig:SW_Profile
```

The following example specifies that the list of files and file types specified by the `-f` option should be excluded from the scan. This change is applied to an inventory profile named `SW_Profile`.

```
wsetinvunixfiles -e EXCLUDE @InventoryConfig:SW_Profile
```

The following example adds the `.txt` and `.gif` file types to the files to be filtered during the scan. These files are either included or excluded during the scan depending on the setting of the `-e` option. This change is applied to an inventory profile named `SW_Profile`.

```
wsetinvunixfiles -f +"*.txt" +"*.gif" @InventoryConfig:SW_Profile
```

The following example specifies that the script `create_mif.sh` is run during the profile distribution. This change is applied to an inventory profile named `SW_Profile`.

Note: Because the path for `create_mif.sh` is not specified in this example, `create_mif.sh` must be in the directory from which this command is run.

```
wsetinvunixfiles -s "create_mif.sh" @InventoryConfig:SW_Profile
```

See Also

wgetinvpcfiles, wgetinvunixfiles, wsetinvglobal, wsetinvpcfiles, wsetinvpchw, wsetinvpcsw, wsetinvunixhw, wsetinvunixsw

wsetinvunixhw

Sets the options for hardware scans of UNIX and OS/400 endpoints.

Syntax

wsetinvunixhw **-a** *component* [**-a** *component*]... *profile_name*

wsetinvunixhw **-r** *component* [**-r** *component*]... *profile_name*

wsetinvunixhw **-t** {Y | N} *profile_name*

wsetinvunixhw **-u** {Y | N} *profile_name*

Description

The **wsetinvunixhw** command sets the options for the hardware scan of UNIX and OS/400 endpoints. You can specify the following options:

- Whether to run the Tivoli hardware scanner during scans of UNIX and OS/400 endpoints.
- Whether the scan data is sent to the configuration repository or stored on the endpoint to be collected later.
- Which hardware components you want to scan for.

Options

-a *component* Adds the specified component to the list of hardware components to scan for. Not all components are available from every platform type.

These component values are not case sensitive. Moreover, you do not need to enter the entire component name. You can enter only the number of characters necessary to uniquely identify that component.

You can specify multiple components by repeating **-a** *component* multiple times.

You can specify the following components.

IPAddress

Lists the TCP/IP configuration information.

Keyboard

Describes the connected keyboard, if any.

Lpar Gets information for logical partitioned systems.

Memory

Describes the installed memory. This option can list details such as the amount of actual memory, virtual memory, and paging space.

MemoryModules

Lists each available memory slot and provides information about installed memory module size and type. This information can be collected only from PC systems running a Linux operating system.

NetworkAdapter

Describes the installed network adapter or adapters.

OperatingSystem

Describes only the operating system running at the time of the scan. This option does not list all installed operating systems on dual-boot systems.

Partition

Describes each of the drive partitions on a system. A system might have only one partition. General information listed can include the drive mapping, the size and available size of each partition, the file system, and the media type.

PCIDevice

Lists any expansion cards installed on the SBus (Solaris) or PCI and AGP (Linux on PC systems) buses.

PointingDevice

Describes the type of pointing device (such as a mouse or trackball) attached to the system.

Processor

Describes the processor or processors installed on the system.

SMBIOS

Lists system-specific identification information, such as manufacturer, model, and serial number of the system, chassis, and BIOS. This information can be collected from SMBIOS-compliant Linux systems only.

Storage

Describes all local storage devices such as hard-disk drives, CD-ROM drives, and removable-media drives. This option does not list network drives.

UnixSystemParam

Describes the current state of the system, such as boot time and run level.

- r component** Removes the specified component from the list of hardware components to scan for. The possible values for *component* are the same as for the **-a** option.
- You can specify multiple components by repeating **-r component** multiple times.
- t option** Specifies whether to run the Tivoli hardware scanner. Use one of the following options:
- Y** Runs the Tivoli hardware scanner.
 - N** Does not run the Tivoli hardware scanner.
- u option** Specifies whether to update the configuration repository with the scan data. Use one of the following options:
- Y** Sends the scan data to the configuration repository.
 - N** Does not send the scan data to the configuration repository. Stores the scan data on the endpoint.
- profile_name** The name of the inventory profile that you want to edit. Enter the profile name in the format **@InventoryConfig:profile_name**.

Authorization

admin, senior, super, or Inventory_edit

Notes

You must collect processor, memory, and operating system data if you want to use the INVENTORY_HWARE query.

Examples

The following command configures the inventory profile HW_Profile to run the Tivoli hardware scanner, include UNIXSystemParam in the scan, and store the scan data on the endpoint.

```
wsetinvunixhw -a UNIXSystemParam -t Y -u N \  
@InventoryConfig:HW_Profile
```

See Also

wgetinvunixhw, wsetinvglobal, wsetinvpcfiles, wsetinvpchw, wsetinvpcsw, wsetinvunixfiles, wsetinvunixsw

wsetinvunixsw

Sets the options for software scans of UNIX and OS/400 endpoints.

Syntax

```
wsetinvunixsw [-b {SCAN | UPDATE | BOTH | NO}] [-c {QUICK | FULL | MD5 | NONE}] [-f {Y | N}] [-p {SCAN | UPDATE | BOTH | NO}] [-s {SCAN | UPDATE | BOTH | NO}] [-x {Y | N}] [-d {Y|N}] profile_name
```

Description

The **wsetinvunixsw** command sets the options for software scans of UNIX endpoints. You can specify whether to scan the files on an endpoint, scan the operating system for information about installed programs, or both.

For file scans, you can configure whether the scan performs the following actions:

- Scans for basic file information.
- Sends scan data to the configuration repository.
- Generates checksum values for scanned files.
- Applies a custom filter to scans for basic file information.
- Scans for signature data.
- Scans executable files only.

Note: File scans do not apply to OS/400 endpoints.

For operating system scans, you can configure whether to send scan data to the configuration repository.

Options

- b option** Specifies whether to scan for basic file information. This type of scan collects the name and path of all scanned files. Other information, such as date of creation, modification, and last access, is collected on some platforms. Use one of the following options:
- SCAN** Scans for basic file information, and then stores the scan data on the endpoint.
- UPDATE** Does not scan. Sends the data from the last scan for basic file information to the configuration repository.
- BOTH** Scans for basic file information, and then sends the scan data to the configuration repository.
- NO** Does not scan for basic file information, and does not send data to the configuration repository. This is the default option.
- c option** Specifies whether to generate a checksum value for scanned files. Use one of the following options:
- QUICK** Generates the Quick checksum value. This option produces a 32-bit value based only on the first 1 KB of each file. This is the quickest checksum option, but produces the least reliable value.

	<p>FULL Generates the Full checksum value. This option produces a 32-bit value based on the full contents of each file. This option takes longer than the QUICK option, but produces a more reliable value.</p> <p>MD5 Generates the MD5 checksum value. This option produces a 128-bit value based on the full contents of each file. The MD5 option provides a more reliable value than the QUICK or FULL options, but takes longer to complete.</p> <p>NONE Does not generate checksum values. This is the default option.</p>
-f option	<p>Specifies whether to filter the scan for basic file information using a custom list of files. You create this list of files using the Custom Filter window or winvfilter command. When this option is set to Y, a scan for basic file information includes only the files specified in the custom filter. Use one of the following options:</p> <p>Y Filters scans for basic file information using the custom filter.</p> <p>N Does not filter scans for basic file information using the custom filter. This is the default option.</p>
-p option	<p>Specifies whether to scan the operating system for information about installed products and patches. Use one of the following options:</p> <p>SCAN Scans the operating system, and then stores the scan data on the endpoint.</p> <p>UPDATE Does not scan. Sends the data from the last operating system scan to the configuration repository.</p> <p>BOTH Scans the operating system, and then sends the scan data to the configuration repository. This is the default option.</p> <p>NO Does not scan the operating system, and does not send data to the configuration repository.</p>
-s	<p>Specifies whether to scan using signatures. In this type of scan, the name and size of each scanned file is compared against the list of signatures. Use one of the following options:</p> <p>SCAN Scans using signatures, and then stores the scan data on the endpoint.</p> <p>UPDATE Does not scan. Sends the data from the last signature scan to the configuration repository.</p> <p>BOTH Scans using signatures, and then sends the scan data to the configuration repository.</p> <p>NO Does not scan using signatures, and does not send data to the configuration repository. This is the default option.</p>
-x option	<p>Specifies whether to filter a file scan by scanning for executable files only. For UNIX systems, an executable file is defined as a file that has the execute permission set. Use one of the following options:</p>

	Y	Scans for executable files only. This is the default option.
	N	Does not scan only for executable files.
-d Y N		Specifies whether the swsigs.txt file must be downloaded to the endpoint. The default value is N, which means that the file is downloaded to the endpoint with every profile distribution. To prevent the file from being downloaded, set the option to Y.
<i>profile_name</i>		The name of the inventory profile that you want to edit. Enter the profile name in the format @InventoryConfig:profile_name .

Authorization

admin, senior, super, or Inventory_edit

Examples

The following command configures the inventory profile SW_Profile to generate a Quick checksum of scanned files, scan the operating system for information about installed products and patches, and send the operating system scan data to the configuration repository.

```
wsetinvunixsw -c QUICK -p BOTH @InventoryConfig:SW_Profile
```

See Also

wgetinvunixsw, wsetinvglobal, wsetinvpcfiles, wsetinvpchw, wsetinvunixfiles, wsetinvunixhw, wsetinvpcsw

wtransfer

Copies files and directories from one managed node to another.

The **wtransfer** command uses TAR software to compress the files to be transferred from one managed node to another. Some HP-UX platforms do not allow long file names, the size limit depends on the specific HP-UX version. So if the TAR software on the source workstation allows long file names, and the TAR software on an HP-UX target workstation does not, the files are not transferred. This problem can be solved only if a patch for the specific HP-UX version is available.

Syntax

```
wtransfer [-h source_host] -s source_file {-t target_label... | -t all_gw | -t all_mn | -T targets_list_file} [-d destination_dir] [-f]
```

Description

The **wtransfer** command copies files and directories from one managed node to another. It can be used only on the directories managed by Tivoli.

Options

-h *source_host*

The name of the managed node where the file or directories to be copied are stored.

-s *source_file*

The absolute path of the files or directories on the source host. You can use wild cards and the Tivoli variables \$(DBDIR), \$(BINDIR), \$(TEMP) and \$(TMP). The Tivoli variables are solved on the source host machine.

-t *target_label*

The name of the target managed node or gateway

-t all_gw | **-t all_mn**

Specifies either all the gateways (**all_gw**) of the Tivoli management region as the targets of the operation, or all the managed nodes (**all_mn**) of the Tivoli management region as the targets of the operation.

-T *targets_list_file*

The absolute name of the file containing the list of target managed node or gateway names.

-d *destination_dir*

The relative path that is added to the managed node installation path (%BINDIR%\..) to determine the target destination path. If the specified destination directory does not exist, it is not created unless you specify the **-f** option to force the creation of the destination directory. You can use the \$(DBDIR), \$(BINDIR), \$(TMP) and \$(TEMP) Framework variables to specify this option. If you use these variables, the path is considered absolute.

-f Forces the creation of the destination directory specified by the **-d** *destination_dir* option if the directory does not exist.

Authorization

super

Return Values

The **wtransfer** command returns one of the following:

0 Indicates that **wtransfer** completed successfully.

other than 0

Indicates that **wtransfer** failed due to an error.

Examples

1. To move the `to_transfer.txt` file to all gateways in the Tivoli Management region forcing the creation the destination path if it does not exist, enter the following command:

```
wtransfer -h lab145864 -s /test/to_transfer.txt -t all_gw -d /statistics -f
```

2. To move all files and directories matching the `c:\te*p` pattern, enter the following command:

```
wtransfer -h lab145864 -s c:\te*p -t all_mn -d /statistics -f
```

See Also

None.

wwaitscan

Identifies the completion of a scan.

Syntax

```
wwaitscan {-e endpoint_name [-e endpoint_name]... | -i scan_ID [-i scan_ID]...} [-t timeout_value]
```

Description

The **wwaitscan** command enables you to determine when a scan has completed by displaying the command prompt when scan data has reached the configuration repository. In scripts, you can use this command to ensure that the script does not proceed until a scan has completed.

The **wwaitscan** command also enables you to monitor scans by displaying information about the scans or endpoints you specify.

Options

-e *endpoint_name*

Specifies the scanned endpoint that you want to monitor. The command displays each endpoint name, the endpoint status, the ID of the scan for each endpoint, and the status of the scan.

-i *scan_ID*

Specifies the ID of the scan that you want to monitor. For endpoint targets, the command displays each scan ID, the status of the scan, the number of endpoints being scanned, and the status of the scan on each endpoint. For pervasive device targets, the command displays the resource label of the pervasive device. For user targets, the command displays the endpoint name and the user name.

-t *timeout_value*

Specifies the number of minutes that the **wwaitscan** command waits before timing out. When you use this option, the **wwaitscan** command times out after the specified time elapses, regardless of the status of the specified scans. If you do not specify a timeout value, the **wwaitscan** command does not terminate until all scans have completed or all targets have been scanned. (A scan is considered complete after the data for the scanned targets is written to the configuration repository.)

Authorization

user, admin, senior, super, or Inventory_view

Examples

The following command displays information about scans 35 and 36. It exits when both scans have completed, or when the timeout period of 15 minutes is reached, whichever comes first.

```
wwaitscan -i 35 -i 36 -t 15
```

The output for this command is as follows:

```
-----  
WWAITSCAN  --  Wait for inventory scans to complete  
-----  
Timeout   : 15 minutes
```

```

-----
Waiting for the following scan IDs to complete.
-----
Scan ID      Activity      Endpoints      Status
-----
35           Not active    0              None
36           Active        1              1 Pending
                                     0 Successful
                                     0 Failed
-----
Scans are complete.

```

The following command displays information about the scans of endpoints fuji and everest. It exits when the scans of both endpoints complete.

```
wwaitscan -e fuji -e everest
```

The output from this command is as follows:

```

-----
WWAITSCAN - Wait for inventory scans to complete
-----
Indefinite. Waiting for specified scans to complete.
-----
Waiting for the following Endpoints to complete.
-----
Endpoint      Activity      Scan Id      Status
-----
everest       Active        35           Active
fuji          Active        35           Active
-----

Endpoint scans are complete.

```

See Also

wgetscanstat

Appendix C. Policy

This appendix includes an explanation of policies along with a description of each policy included with Inventory.

Policies are rules that govern the resources in policy regions. There are two kinds of policies that set guidelines in a policy region:

- *Default policy*—Enables you to set the defaults for managed resources.
- *Validation policy*—Enables you to create rules that are enforced when administrators add or change managed resources.

See the *Tivoli Management Framework: User's Guide* for detailed information about policies and policy regions.

This appendix provides information about the default policy methods provided with Inventory and an example of how to create new policy methods and policy objects.

The Inventory default policy methods described in this appendix set the defaults for **InventoryConfig** resources. In other words, the Inventory default policy methods determine what options are set by default when you create a new inventory profile. (Inventory does not provide validation policies.)

You can change the Inventory default policy methods if you want to preset inventory profile properties with specific values. For example, if you want all of your inventory profiles within a particular policy region to update the configuration repository with only the differences since the last scan, you can define a policy method that always sets this option for newly created profiles.

Default policy methods

Default policy methods are shell scripts or programs—UNIX scripts (such as Bourne, K, and Perl shell scripts), awk programs, C programs, and so on—that are invoked by Inventory when a new inventory profile is created. By creating scripts or programs and replacing the contents of these policy methods, you can control the options that are automatically set in newly created inventory profiles. When creating your own policy method, use the following guidelines:

- It is recommended that you write your policy programs in an interpreter language. Policies are stored in the database, and interpretive programs usually require less space than executable files (compiled programs). Also, an interpretive program can be used across multiple platforms; executable files cannot.
- When Inventory invokes a default policy method, the name of the inventory profile being created is passed to the method as the first argument. Inventory expects the default policy methods to exit with the code **0**—you must write your policy methods to do so. Reserve other exit codes for hard errors, such as insufficient memory, incorrect usage, and so on.
- Do not use C shell scripts because file descriptors are not handled as needed for Tivoli Management Framework.

Policy objects

A policy object is a set of policy methods for a specific resource class. The default policy methods that govern the **InventoryConfig** resource are defined in a default policy object called **BasicInventoryConfig**.

The **BasicInventoryConfig** policy object and its policy methods are provided with Inventory. You can also create additional policy objects for the **InventoryConfig** resource. Multiple policy objects enable you to define different policies that have the same policy methods.

For example, suppose you have two policy regions called **Data** and **Software**. To create policies that set the initial values for inventory profile names and inventory profile options in each policy region, you can create separate policy objects, such as **DataPolicy** and **SoftwarePolicy**. After you define the policies for the methods in each policy object and assign the policy objects to the policy regions, any newly created inventory profiles would adhere to the guidelines you defined.

To define a new policy object and its policy methods, you must perform the following procedures:

- Create a new policy object.
- Replace the contents of the new policy object methods.
- Assign the new policy object to the policy region in which the inventory profiles will reside.

The following sections provide detailed instructions on each of these procedures. See the *Tivoli Management Framework: User's Guide* for detailed information about checking policy in a policy region.

Creating a new policy object

To define different policies for multiple policy regions, you must create new policy objects. If you do not define policy for a policy region, Inventory uses the **BasicInventoryConfig** policy object and its policy methods by default.

The following table provides the authorization roles required to create a new policy object:

Operation	Context	Role
Create a new policy object	Tivoli region	senior or super

You must use the command line to create a new policy object.

Enter the following syntax for the **wcrtpol** command to create a default policy object:

```
wcrtpol -d InventoryConfig DataPolicy
```

where:

-d Creates a default policy object.

InventoryConfig

Specifies **InventoryConfig** as the resource type of the new default policy object.

DataPolicy

Specifies **DataPolicy** as the name of the new default policy object.

A new policy object inherits its policy methods from an existing policy object. After you create a new policy object, you can view the existing policy methods to validate inventory profile properties or operations for a particular policy region using the following commands:

wlspolm

Lists the policy methods for the specified resource. You can list the default or validation policy methods with this command.

wgetpolm

Returns the contents of the specified default or validation policy method. Policy methods are inherited from an existing **InventoryConfig** policy object.

For more information about the **wcrtpol**, **wlspolm**, and **wgetpolm** commands, see the *Tivoli Management Framework: Reference Manual*.

Replacing the contents of a policy method

To define different policies from those inherited by the parent policy object, you must create a script or program and replace the existing policy method with it.

The following table provides the authorization roles required to replace the contents of a policy method:

Operation	Context	Role
Replace the content of a policy method	Tivoli region	senior or super

You must use the command line to replace the content of a policy method.

Enter the following syntax for the **wputpolm** command to replace the contents of the **ic_def_global_logfile_path** policy method with the contents of the **Data_logfile_def.sh** script:

```
wputpolm -d InventoryConfig DataPolicy \  
ic_def_global_logfile_path < Data_logfile_def.sh
```

where:

-d Specifies that the method is a default policy method.

InventoryConfig

Specifies **InventoryConfig** as the resource type for which the policy is defined.

DataPolicy

so Specifies **DataPolicy** as the policy object that contains the default policy method being replaced.

ic_def_global_logfile_path

Replaces the contents of the **ic_def_global_logfile_path** default policy method.

< **Data_logfile_def.sh**

Redirects the **Data_logfile_def.sh** script to the command. The contents of this file replace the existing contents of the **ic_def_global_logfile_path** policy method.

For more information about the **wputpolm** command, see the *Tivoli Management Framework: Reference Manual*.

Assigning policy to a policy region

To change the default policy for a policy region, you must assign policy to the policy region after you have created a new policy object and replaced its policy methods.

The following table provides the authorization roles required to assign policy to a policy region:

Operation	Context	Role
Assign policy to a policy region	Policy region	senior or super

You can use either the Tivoli desktop or the command line to assign policy to a policy region. See the *Tivoli Management Framework: User's Guide* for instructions about how to use the Tivoli desktop.

To use the **wsetpr** command to change the default policy in the **Dev** policy region to those methods defined in the **DataPolicy** policy object, enter the following command:

```
wsetpr -d DataPolicy InventoryConfig @PolicyRegion:Dev
```

where:

-d DataPolicy

Changes the default policy to that defined in the **DataPolicy** object.

InventoryConfig

Specifies **InventoryConfig** as the resource type for which the policy is defined.

@PolicyRegion:Dev

Specifies **Dev** as the policy region for which to assign the policy.

For more information about the **wsetpr** command, see the *Tivoli Management Framework: Reference Manual*.

Example—Setting a default policy method

The following example shows how to use the command line to set the policy default for the default list of file extensions to be matched, and how to create and assign policy to a new policy object. Complete the following steps:

1. List the default policy methods for the **InventoryConfig** class by entering the following command from a root prompt:

```
wlspolm -d InventoryConfig
```

where:

-d Lists the default policy methods for the **InventoryConfig** resource type.

InventoryConfig

Specifies **InventoryConfig** as the resource whose policy methods are to be listed.

The following default policies are returned:

```
ic_def_global_action
ic_def_global_ep_timeout
ic_def_global_logfile_host
ic_def_global_logfile_path
ic_def_global_notice_interval
ic_def_global_notice_location
ic_def_global_notice_type
ic_def_global_security
ic_def_global_update_replace
ic_def_pc_custom_before_script
ic_def_pc_custom_mifs
ic_def_pc_custom_script
ic_def_pc_exclude_dirs
ic_def_pc_extensions
ic_def_pc_hw_gran
ic_def_pc_hw_outfile
ic_def_pc_hw_scan
ic_def_pc_include_dirs
ic_def_pc_sw_crc
ic_def_pc_sw_flags
ic_def_pc_sw_outfile
ic_def_pc_sw_scan
ic_def_pvd_config_scan
ic_def_pvd_hw_scan
ic_def_pvd_sw_scan
ic_def_unix_custom_before_script
ic_def_unix_custom_mifs
ic_def_unix_custom_script
ic_def_unix_exclude_dirs
ic_def_unix_extensions
ic_def_unix_hw_gran
ic_def_unix_hw_outfile
ic_def_unix_hw_scan
ic_def_unix_include_dirs
ic_def_unix_sw_crc
ic_def_unix_sw_flags
ic_def_unix_sw_outfile
ic_def_unix_sw_scan
```

2. List the default policy objects that exist for the **InventoryConfig** class. Tivoli Management Framework supports multiple default policy objects, which means that you can have one set of policy defaults in policy region **X** and a different set in policy region **Y**. Use the following command to list the default policy objects for Inventory:

```
wlspol -d InventoryConfig
```

where:

-d Lists the default policy objects for the **InventoryConfig** resource type.

InventoryConfig

Specifies **InventoryConfig** as the resource whose policy objects are to be listed.

This command returns only the **BasicInventoryConfig** object if you have not created additional default policy objects.

3. Extract the current contents of the **ic_def_pc_extensions** default policy method to make sure that another administrator has not modified it.

```
wgetpolm -d InventoryConfig BasicInventoryConfig \
ic_def_pc_extensions
```

where:

-d Lists the contents of a default policy method.

InventoryConfig

Specifies **InventoryConfig** as the resource whose policy is to be returned.

BasicInventoryConfig

Specifies **BasicInventoryConfig** as the policy object whose policy is to be returned.

ic_def_pc_extensions

Specifies the policy method whose contents are to be returned.

The contents of this policy method are output to standard output by default. This command returns the following output:

```
#!/bin/sh
# Component Name:

# $Date: 2000/10/03 15:00:00 $
#
# $Source: ic_def_pc_extensions,v $
#
# $Id: ic_def_pc_extensions,v 1.0 2000/10/03 15:00:00 plw Exp $
#
# Description:
#
# Copyright 2002 by IBM Corporation
# Unpublished Work
# All Rights Reserved
# Licensed Material - Property of IBM Corporation

EXCLUDE=0
INCLUDE=1

# MUST set Include or Exclude first before listing extensions.
# If no extensions are listed then just select on Include or Exclude.
# It will not matter.
echo $INCLUDE

# PC extensions MUST be in uppercase. No validation will be done.
echo "*.EXE
*.DLL
*.COM
*.NLM"

exit 0
```

4. Create a script that changes the extension list to an EXCLUDE list, and sets the list of extensions to exclude. The following script, called /tmp/list.sh, accomplishes this:

```
#!/bin/sh
# Custom policy

EXCLUDE=0
INCLUDE=1

# MUST set Include or Exclude first before listing extensions.
# If no extensions are listed then just select on Include or Exclude.
# It will not matter.
echo $EXCLUDE

echo "*.BAK
*.TMP
```



```
*.SYS  
*.INI"
```

```
exit 0
```

5. Replace the contents of the **ic_def_pc_extensions** policy method with the new script using the following command:

```
wputpolm -d InventoryConfig BasicInventoryConfig \  
ic_def_pc_extensions </tmp/list.sh
```

where:

-d Specifies that the **ic_def_pc_extensions** policy method is a default policy method.

InventoryConfig

Specifies **InventoryConfig** as the resource for which the policy is set.

BasicInventoryConfig

Specifies **BasicInventoryConfig** as the policy object for which the policy is set.

ic_def_pc_extensions

Specifies **ic_def_pc_extensions** as the policy method whose contents are to be replaced.

</tmp/list.sh

Redirects the **/tmp/list.sh** script to the command. This command reads its input from standard input.

6. Associate the new policy method in the **BasicInventoryConfig** policy object with the **Dev** policy region:

```
wsetpr -d BasicInventoryConfig InventoryConfig \  
@PolicyRegion:Dev
```

where:

-d BasicInventoryConfig

Changes the default policy to that defined in the **BasicInventoryConfig** object.

InventoryConfig

Specifies **InventoryConfig** as the resource type for which the policy is defined.

@PolicyRegion:Dev

Specifies **Dev** as the policy region for which to assign the policy.

After setting the policy, every inventory profile created in policy regions whose default policy for the **InventoryConfig** resource type is set to

BasicInventoryConfig will have the list of file extensions set as specified in the sample script.

Policy methods

The following default policy methods enable you to control the default settings in **InventoryConfig** resources. These methods can differ among policy regions. That is, one policy region could have one set of default policy methods and another policy region could have another.

Note: The query default policy methods are also listed here, though they are packaged with Tivoli Management Framework.

Policy methods must return a zero for the exit code if the method was successful. If the method fails, a nonzero exit code is returned. Information that the method returns must be written as a string to standard output (stdout).

ic_def_global_action

Specifies whether to distribute only the configuration file, or to distribute the configuration file and also perform a scan.

Resource

InventoryConfig

Syntax

ic_def_global_action

Description

This method specifies whether an inventory profile distributes only the configuration file, or distributes the configuration file and also performs a scan. For example, you might want to only distribute the configuration file if you are performing an endpoint-initiated scan using the **wepscan** command.

Permissible values for this method are as follows:

- 16** Distribute the configuration file only.
- 17** Distribute the configuration file and perform a scan. This is the default value.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_global_ep_timeout

Specifies the default number of seconds after which a scan of an endpoint times out. For pervasive devices, specifies the time it takes for the execution of the method on the endpoint to create the jobs on the Web Gateway component.

Resource

InventoryConfig

Syntax

ic_def_global_ep_timeout

Description

This method specifies the default value of the `endpt_time_out` attribute. Inventory times the duration of a scan on each endpoint. If the scan does not complete (the endpoint does not return status for the scan) before the time period set by this method, the endpoint is timed out, and Inventory indicates that the scan has failed for that endpoint.

Permissible values for this method are a decimal number greater than 0. The default value is 1800 seconds.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_global_logfile_host

Specifies the name of the managed node on which the log file for Inventory status information is to be stored.

Resource

InventoryConfig

Syntax

ic_def_global_logfile_host

Description

This method specifies the name of the managed node on which to write the log file for Inventory status information. This method is used only if an inventory profile has been configured to write the status information to a log file. If no managed node is specified, the default option is the managed node on which the inventory data handler is installed. If a name is specified, the name must match the name of a valid managed node. An example is as follows:

```
echo "rainier"  
exit 0
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_global_logfile_path

Specifies the path and name for the log file for Inventory status information.

Resource

InventoryConfig

Syntax

ic_def_global_logfile_path

Description

This method specifies the path and name for the log file that holds Inventory status information. This method is used only if an inventory profile has been configured to write the status information to a log file. If no value is specified in this method, the file is named `inv_scan_n`, where *n* is the ID number of the scan. The file is placed in the temporary directory of the managed node specified by `ic_def_global_logfile_host`. This temporary directory is usually one of the following paths:

- For UNIX systems: `/tmp`, `/usr/tmp`, or `/var/tmp`
- For Windows systems: `c:\temp`

An example is as follows:

```
echo "/tmp/pc_scan_hw.log"  
exit 0
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_global_notice_interval

Specifies when notification is sent for a scan that is configured to report status.

Resource

InventoryConfig

Syntax

ic_def_global_notice_interval

Description

This method specifies when notification is sent for a scan that is configured to report status. The options are to send notification as each target completes, to send notification at periodic intervals, or to send notification only when all targets have completed.

Permissible values for this method are as follows:

- 1** Send notification as each target completes.
- 16** Send notification after a specific time period, or send notification each time a specified number of targets complete, or both. The time period and the number of targets that complete is based on the settings for the inventory data handler.
- 256** Send notification only after all targets are complete. This is the default value.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_global_notice_location

Specifies the locations to which notifications are sent.

Resource

InventoryConfig

Syntax

ic_def_global_notice_location

Description

This method indicates where status information for a profile are written during a scan.

Permissible values for this method are as follows:

- 0** Do not write any status information. No value is the same as a value of 0.
- 1** Write status information to the inventory notice group. This is the default value.
- 16** Write status information to a log file.
- 256** Write status information to the Tivoli Enterprise Console console.

You can specify that you want to write to multiple locations by adding multiple values for the method. The following example writes status information to the inventory notice group and Tivoli Enterprise Console:

```
echo 257  
or  
expr 256 + 1  
exit 0
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_global_notice_type

Specifies the circumstances for which notification is sent.

Resource

InventoryConfig

Syntax

ic_def_global_notice_type

Description

This method returns a numeric value indicating the circumstances for which notification is sent.

Permissible values for this method are as follows:

- 0** Do not send any notification.
- 1** Send notification for targets that succeed.
- 16** Send notification for targets that fail.
- 17** Send notification for both successful and failed targets. This is the default value.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_global_security

Specifies whether the specified profile can be distributed to targets that are not subscribers to the profile manager in which the inventory profile resides.

Resource

InventoryConfig

Syntax

ic_def_global_security

Description

This method specifies whether the specified profile can be distributed to targets (endpoints, resource groups, or profile managers) that are not subscribers to the profile manager in which the inventory profile resides.

Permissible values for this method are as follows:

- 0** Do not distribute the profile to targets that are not subscribers to the profile manager in which the inventory profile resides.
- 1** Distribute the profile to non-subscribers.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_global_update_replace

Specifies whether Inventory replaces all data or replaces only differences when the configuration repository is updated with the results of a scan.

Resource

InventoryConfig

Syntax

ic_def_global_update_replace

Description

This method specifies whether Inventory replaces all data in the configuration repository during a scan, or compares the current scan data to the MIF file generated by the previous scan and replaces only the differences. If an endpoint does not yet have scan data in the database, all data for that endpoint is sent to the configuration repository. This method does not apply to pervasive devices; Inventory always replaces all data in the configuration repository for pervasive device scans.

Permissible values for this method are as follows:

- 1** Compare the results of a scan with the results of the previous scan and save the differences in the configuration repository. This is the default option.
- 16** Save the current scan results in the configuration repository.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pc_custom_before_script

Specifies the default scripts to be run before the scan of a PC.

Resource

InventoryConfig

Syntax

ic_def_pc_custom__before_script

Description

This method specifies the custom scripts to be run before a PC scan. Scripts specified by this method are usually .BAT or .CMD scripts. An example follows:

```
echo "C:/MYSCAN.EXE  
COPY C:/MYSCAN.MIF D:/TIVOLI/USERADD.MIF
```

By default, no custom scripts are run.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pc_custom_mifs

Specifies the path and file name of custom MIF files that are read during the scan of a PC endpoint.

Resource

InventoryConfig

Syntax

ic_def_pc_custom_mifs

Description

This method generates the list of custom MIF files that Inventory reads from PC endpoints. By default, no custom MIF files are read. If this method writes out more than one custom MIF file, each file name must be written out on a new line.

Directory names must be delimited using forward slashes (/) as in the following example:

```
echo "useradd.mif  
C:/scandata/data.mif"  
exit 0
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pc_custom_script

Specifies the default scripts to be run during the scan of a PC endpoint.

Resource

InventoryConfig

Syntax

ic_def_pc_custom_script

Description

This method specifies the custom scripts to be run during the scan of a PC endpoint (after the scan completes but before the MIF files are read). Scripts specified by this method are usually .BAT or .CMD scripts. An example follows:

```
echo "C:/MYSCAN.EXE  
COPY C:/MYSCAN.MIF D:/TIVOLI/USERADD.MIF
```

By default, no custom scripts are run.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pc_exclude_dirs

Specifies the default list of directories that are excluded during a software scan of a PC endpoint.

Resource

InventoryConfig

Syntax

ic_def_pc_exclude_dirs

Description

This method generates the default list of directories that an inventory profile does not scan during a software scan of a PC endpoint. The method writes a list of directory names, each on a separate line, to standard output. By default, the `*/TMP`, `*/TEMP`, `/RECYCLED/`, and `/RECYCLER/` directories are not scanned.

Directory names must be uppercase and delimited with forward slashes. An example follows:

```
echo "*/CACHE
*/BACKUP"
exit 0
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pc_extensions

Specifies a list of files or file types to either include or exclude during a software scan of PC endpoints.

Resource

InventoryConfig

Syntax

ic_def_pc_extensions

Description

This method provides a list of file names (such as file.exe), file types (such as *.exe), or both. The method generates a value indicating whether the files or file types are to be included or excluded during a software scan of a PC endpoint.

By default, the *.exe, *.dll, *.com, *.nlm, and *.sig file types are included in software scans of a PC.

The first line of method output is one of the following options:

```
0      Exclude the list of extensions from a PC software scan
1      Include the list of extensions from a PC software scan
```

The lines following this first line of output list file names or file types, each separated by a new line, as in the following example:

```
echo 1
echo "*.EXE
*.DLL
*.DRV
*.NLM
CONFIG.SYS"
exit 0
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pc_hw_gran

Indicates the hardware components that are scanned for on a PC endpoint.

Resource

InventoryConfig

Syntax

ic_def_pc_hw_gran

Description

This method generates a list of the hardware components that can be scanned for during a PC endpoint scan and assigns a value for each component. The value indicates whether or not to collect information for that component. The following is the contents of the method provided with Inventory. The list contains all the potential component names that you can use in your method.

```
$/bin/sh
OFF=0
ON=1
echo "Processor $ON"
echo "Memory $ON"
echo "Operating System $ON"
echo "Storage $ON"
echo "IP Address $ON"
echo "Network Adapter $ON"
echo "Partition $ON"
echo "Pointing Device $ON"
echo "Keyboard $ON"
echo "IPX Address $ON"
echo "Video $ON"
echo "Printer $ON"
echo "USB Device $ON"
exit 0
```

By default, all hardware components in this list are scanned for.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pc_hw_outfile

Specifies the MIF files that are retrieved from PC endpoints during a hardware scan.

Resource

InventoryConfig

Syntax

ic_def_pc_hw_outfile

Description

This method generates a decimal value that indicates which MIF files, and therefore what scan data, is retrieved from a PC endpoint and sent to the configuration repository during a hardware scan.

Permissible values for this method are as follows:

- 0** Do not return any of the MIF files during a hardware scan.
- 1** Return the hardware MIF file produced by the Tivoli hardware scanner. This is the default option.
- 16** Return the hardware MIF file produced by the DMI scanner.
- 17** Return the hardware MIF file produced by both the Tivoli hardware scanner and the DMI scanner.

You can specify that you want both MIF files returned by adding the values for the Tivoli hardware scanner and the DMI scanner. For example:

```
echo 17  
or  
expr 16 + 1
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pc_hw_scan

Specifies which hardware scanners to run when scanning PC endpoints.

Resource

InventoryConfig

Syntax

ic_def_pc_hw_scan

Description

This method generates a decimal value indicating which hardware scanners to run during a scan of PC endpoints.

Permissible values for this method are as follows:

- 0** Do not run any hardware scanners.
- 1** Run the Tivoli hardware scanner. This is the default option.
- 16** Run the DMI scanner.
- 17** Run both the Tivoli hardware scanner and the DMI scanner.

You can run both the Tivoli and DMI scanners by adding the values for the Tivoli scanner and the DMI scanner. For example:

```
echo 17  
or  
expr 16 + 1
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pc_include_dirs

Specifies the default list of directories that are included during a software scan of a PC endpoint.

Resource

InventoryConfig

Syntax

ic_def_pc_include_dirs

Description

This method generates the default list of directories that are scanned during a software scan of a PC endpoint. The method writes a list of directory names to standard output, with each directory separated by a new line. PC directory names must be uppercase, and directories must be delimited by forward slashes (/). For example:

```
echo "*/PROGRAM_FILES
*/APPLICATIONS"
exit 0.
```

By default, no directories are specified (all directories are scanned).

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pc_sw_crc

Specifies which cyclic redundancy check (CRC) value, if any, is obtained for files during a software scan of PC endpoints.

Resource

InventoryConfig

Syntax

ic_def_pc_sw_crc

Description

This method generates a decimal value that indicates which CRC value is obtained for files during a PC software scan. Only one type of CRC value can be collected during a software scan.

Permissible values for this method are as follows:

- 0** Do not compute a CRC for files. This is the default option.
- 1** Produce a 4-byte checksum for the first 1024 bytes of the file.
- 16** Produce a 4-byte checksum for all the bytes in the file.
- 256** Produce a 16-byte MD5 checksum for a file.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pc_sw_flags

Specifies which special options are in effect during software scans of PC endpoints.

Resource

InventoryConfig

Syntax

ic_def_pc_sw_flags

Description

This method generates a decimal value that indicates which special options are in effect during a software scan of a PC endpoint. The options available indicate whether to scan for executables only, and whether to apply a custom filter to the basic information software scan.

Permissible values for this method are as follows:

- 0** No special PC software scan options. This is the default option.
- 1** Scan only for executable files.
- 16** Apply a custom filter to the basic information software scan.
- 17** Scan only for executables and apply a custom filter.

You can specify more than one option by generating the sum of the desired options. For example, you can specify to scan only for executables and apply a custom filter as follows:

```
echo 17  
or  
expr 1 + 16
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pc_sw_outfile

Specifies the MIF files that are retrieved from PC endpoints during a software scan.

Resource

InventoryConfig

Syntax

ic_def_pc_sw_outfile

Description

This method generates a decimal value that indicates which MIF files, and therefore what scan data, is retrieved from a PC endpoint and sent to the configuration repository during a software scan.

Permissible values for this method are as follows:

- 0** Do not return any of the MIF files.
- 1** Return the MIF file that contains basic file information.
- 16** Return the MIF file that contains file header information.
- 256** Return the MIF file that contains registry information.
- 4096** Return the MIF file produced by signature matching.

You can specify that you want multiple MIF files returned by adding the values for desired the MIF files. For example, you can specify to return the basic file information, the file header information, and the PC registry information as follows:

```
echo 273  
or  
expr 256 + 16 + 1
```

The default value is 272 (256 + 16).

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pc_sw_scan

Specifies which software scanners to run when scanning PC endpoints.

Resource

InventoryConfig

Syntax

ic_def_pc_sw_scan

Description

This method generates a decimal value indicating which software scanners to run on a PC endpoint during a software scan.

Permissible values for this method are as follows:

- 0** Do not run any software scanners.
- 1** Scan for basic file information.
- 16** Scan for file header information.
- 256** Scan the PC registry. This is the default option.
- 4096** Scan for file signature information.

You can specify that you want to run multiple scanners by adding the values for the scans you want. For example, you can run scanners for the PC registry, file information, and basic file information as follows:

```
echo 273  
or  
expr 256 + 16 + 1
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pvd_config_scan

Specifies whether to perform a scan of user-configurable settings on pervasive devices. User-configurable settings include language and alarm settings and time and date format.

Resource

InventoryConfig

Syntax

ic_def_pvd_config_scan

Description

This method generates a decimal value indicating whether to perform a scan of user-configurable settings on pervasive devices.

Permissible values for this method are as follows:

- 0** Do not perform a scan of user-configurable settings on pervasive devices.
- 1** Perform a scan of user-configurable settings on pervasive devices. This is the default option.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pvd_hw_scan

Specifies whether to perform a hardware scan on pervasive devices.

Resource

InventoryConfig

Syntax

ic_def_pvd_hw_scan

Description

This method generates a decimal value indicating whether to perform a hardware scan on pervasive devices.

Permissible values for this method are as follows:

- 0** Do not perform a hardware scan on pervasive devices.
- 1** Perform a hardware scan on pervasive devices. This is the default option.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_pvd_sw_scan

Specifies whether to perform a software scan on pervasive devices.

Resource

InventoryConfig

Syntax

ic_def_pvd_sw_scan

Description

This method generates a decimal value indicating whether to perform a software scan on pervasive devices.

Permissible values for this method are as follows:

- 0** Do not perform a software scan on pervasive devices.
- 1** Perform a software scan on pervasive devices. This is the default option.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_unix_custom_before_script

Specifies a user-defined script to be run before the scan of UNIX systems.

Resource

InventoryConfig

Syntax

ic_def_unix_custom_before_script

Description

This method generates a script to be run before a scan. On UNIX systems, this script is normally a .sh script, as in the following example:

```
echo "#!/bin/sh
/usr/local/bin/myscan.sh
cp /tmp/scandata.mif /tivoli/useradd.mif
"
```

By default, no custom scripts are run.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_unix_custom_mifs

Specifies the path and file names of custom MIF files that are read during the scan of UNIX endpoints.

Resource

InventoryConfig

Syntax

ic_def_unix_custom_mifs

Description

This method generates the list of custom MIF files that Inventory reads from UNIX target machines. By default, no custom MIF files are read. If this method writes out more than one custom MIF file, each file name must be written out on a new line, as in the following example:

```
echo "useradd.mif  
/usr/apps/custom.mif"  
exit 0
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_unix_custom_script

Specifies a user-defined script to be run during the scan of UNIX systems.

Resource

InventoryConfig

Syntax

ic_def_unix_custom_script

Description

This method generates a script to be run during a scan (after the scan completes but before the MIF files are read). On UNIX systems, this script is normally a .sh script, as in the following example:

```
echo "#!/bin/sh
/usr/local/bin/myscan.sh
cp /tmp/scandata.mif /tivoli/useradd.mif
"
```

By default, no custom scripts are run.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_unix_exclude_dirs

Specifies the default list of directories that are excluded during a software scan of UNIX systems.

Resource

InventoryConfig

Syntax

ic_def_unix_exclude_dirs

Description

This method generates the default list of directories that an inventory profile will not scan during a profile distribution to UNIX systems. By default, the `*/tmp` directory is excluded.

The method must write a list of directory names to standard output, each separated by a new line. For example:

```
echo "*/tmp
*/temp/"
exit 0
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_unix_extensions

Specifies a list of file names or file types to either include or exclude during a software scan of UNIX systems.

Resource

InventoryConfig

Syntax

ic_def_unix_extensions

Description

This method provides a list of file names (such as readme.txt), file types (such as *.txt), or both. The method generates a value indicating whether the file names and file types are to be included or excluded in a software scan of UNIX system.

The first line of method output is one of the following options:

```
0      Exclude the list of extensions from a UNIX software scan
1      Include the list of extensions in a UNIX software scan
```

The lines following this first line of output list file names or file types, each separated by a new line, as in the following example:

```
echo 1
echo "*.txt
myfile
*.TXT"
exit 0
```

By default, no file names or file types are specified (all files are included in the scan).

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_unix_hw_gran

Indicates the hardware components that are scanned for on UNIX systems.

Resource

InventoryConfig

Syntax

ic_def_unix_hw_gran

Description

This method generates a list of the hardware components that can be scanned for during a scan of UNIX systems and assigns a value for each component. The value indicates whether or not to collect information for that component. The following is the contents of the method provided with Inventory. The list contains all the potential component names that you can use in your method.

```
#!/bin/sh
OFF=0
ON=1
echo "Processor $ON"
echo "Memory $ON"
echo "Operating System $ON"
echo "Storage $ON"
echo "IP Address $ON"
echo "Network Adapter $ON"
echo "Partition $ON"
echo "Pointing Device $ON"
echo "Keyboard $ON"
echo "Unix System Params $ON"
exit 0
```

By default, all components in this list are scanned for.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_unix_hw_outfile

Specifies the MIF files that are retrieved from UNIX systems during a hardware scan.

Resource

InventoryConfig

Syntax

ic_def_unix_hw_outfile

Description

This method generates a decimal value that indicates whether the MIF that contains the hardware scan data is retrieved from UNIX endpoints and sent to the configuration repository during a hardware scan.

Permissible values for this method are as follows:

- 0** Do not return the hardware MIF file during a hardware scan.
- 1** Return the hardware MIF file. This is the default option.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_unix_hw_scan

Specifies whether to perform a hardware scan on UNIX endpoints.

Resource

InventoryConfig

Syntax

ic_def_unix_hw_scan

Description

This method generates a decimal value indicating whether to perform a hardware scan on UNIX endpoints. The following are the permissible values:

- 0** Do not perform a hardware scan on UNIX endpoints.
- 1** Perform a hardware scan on UNIX endpoints. This is the default option.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_unix_include_dirs

Specifies the default list of directories that are included during a software scan of UNIX endpoints.

Resource

InventoryConfig

Syntax

ic_def_unix_include_dirs

Description

This method generates the default list of directories that are scanned during a software scan of UNIX endpoints. The method writes a list of directory names to standard output, with each directory separated by a new line. Directories must be delimited by forward slashes (/). For example:

```
echo "/usr/local/bin
*/bin"
exit 0
```

By default, no directories are specified (all directories are scanned).

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_unix_sw_crc

Specifies which cyclic redundancy check (CRC) value, if any, is obtained for files during a software scan of UNIX systems.

Resource

InventoryConfig

Syntax

ic_def_unix_sw_crc

Description

This method generates a decimal value that indicates which CRC value is obtained for files during a software scan of UNIX systems. Only one type of CRC value can be collected during a software scan. The following values are permissible:

- 0** Do not compute a CRC for files. This is the default option.
- 1** Produce a 4-byte checksum for the first 1024 bytes of the file.
- 16** Produce a 4-byte checksum for all the bytes in the file.
- 256** Produce a 16-byte MD5 checksum for a file.

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_unix_sw_flags

Specifies which special options are in effect during software scans of UNIX systems.

Resource

InventoryConfig

Syntax

ic_def_unix_sw_flags

Description

This method generates a decimal value that indicates which special options are in effect during a software scan of UNIX systems. The options available indicate whether to scan for executables only, and whether to apply a custom filter to a software scan for basic information. The permissible option values are as follows:

- 0** No special software scan options for UNIX systems.
- 1** Scan only for executable files. This is the default option.
- 16** Apply a custom filter to a software scan for basic information.

You can specify more than one option by generating the sum of the desired options. For example, you can specify to scan only for executables and apply a custom filter as follows:

```
echo 17  
or  
expr 1 + 16
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_unix_sw_outfile

Specifies the Tivoli MIF files that are retrieved from UNIX systems during a software scan.

Resource

InventoryConfig

Syntax

ic_def_unix_sw_outfile

Description

This method generates a decimal value that indicates which MIF files, and therefore what scan data, is retrieved from UNIX endpoints and sent to the configuration repository during a software scan. Permissible values for this method are as follows:

- 0** Do not return any of the MIF files.
- 1** Return the MIF file that contains basic file information.
- 256** Return the MIF file that contains installed product and patch information. This is the default option.
- 4096** Return the software MIF file produced by signature matching.

You can specify that you want multiple MIF files returned by adding the values for desired the MIF files. For example, you can specify to return the basic file information and the signature information as follows:

```
echo 4097  
or  
expr 4096 + 1
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

ic_def_unix_sw_scan

Specifies which software scanners to run when scanning UNIX endpoints.

Resource

InventoryConfig

Syntax

ic_def_unix_sw_scan

Description

This method generates a decimal value indicating which software scanners to run on UNIX systems during a software scan. Permissible values are as follows:

- 0** Do not run any software scanners.
- 1** Scan for basic file information.
- 256** Scan for installed product and patch information. This is the default option.
- 4096** Scan for file signature information.

You can specify that you want to run multiple scanners by adding the values for the scans you want. For example, to scan for basic file information and installed product and patch information:

```
echo 257  
or  
expr 256 + 1
```

Return Codes

Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a nonzero exit code if a failure occurs. Information that the method returns must be written as a string to standard output.

Appendix D. XML schema definition for signature catalogs

Use the following IBMSoftwareUpdate.xsd file as a reference to validate XML files you defined, which contain the signature definitions.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Licensed Materials - Property of IBM
5724-D33
IBM Tivoli License Compliance Manager

(C) Copyright IBM Corp. 2002, 2006. All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- ===== -->
  <!-- Global elements definition -->
  <!-- ===== -->
  <xsd:element name="IBMSoftwareUpdate"><xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Platforms" type="PlatformsType"/>
      <xsd:element name="Vendors" type="VendorsType"/>
      <xsd:element name="Signatures" type="SignaturesType"/>
      <xsd:element name="Components" type="ComponentsType"/>
      <xsd:element name="Products" type="ProductsType"/>
      <xsd:element name="Supersedes" type="SupersedesType"/>
    </xsd:sequence>
    <xsd:attribute name="Copyright" type="xsd:string" use="required"/>
    <xsd:attribute name="Date" type="DateType" use="required"/>
    <xsd:attribute name="Description" type="xsd:string"/>
    <xsd:attribute name="Version" type="xsd:string" use="required"/>
    <xsd:attribute name="IBMUseOnly" type="xsd:boolean" default="false"/>
    <xsd:attribute name="Exported" type="xsd:boolean" default="false"/>
  </xsd:complexType>
</xsd:element>
  <!-- ===== -->
  <!-- Container elements definition -->
  <!-- ===== -->
  <xsd:complexType name="PlatformsType">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="Platform" type="PlatformType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="VendorsType">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="Vendor" type="VendorType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SignaturesType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="File" type="FileType"/>
      <xsd:element name="WinReg" type="WinRegType"/>
      <xsd:element name="InstReg" type="InstRegType"/>
      <xsd:element name="AppServer" type="AppServerType"/>
      <xsd:element name="J2EEApp" type="J2EEAppType"/>
      <xsd:element name="XslmId" type="XslmIdType"/>
      <xsd:element name="ExtSig" type="ExtSigType"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="ComponentsType">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="Component" type="ComponentType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ProductsType">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="Product" type="ProductType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SupersedesType">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="Supersedes" type="SupersedesType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

```

</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ProductsType">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="Product" type="ProductType"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SupersedesType">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="Supersede" type="SupersedeType"/>
  </xsd:sequence>
</xsd:complexType><!-- ===== -->
<!-- Signature types definitions -->
<!-- ===== -->
<!-- Each signature type extends an AbstractSignatureType, which collects
      common attributes -->
<xsd:complexType name="AbstractSignatureType" abstract="true">
  <xsd:attribute name="ID" type="SignatureIdType" use="required"/>
  <xsd:attribute name="TargetPlatform" type="xsd:IDREF" use="required"/>
  <xsd:attribute name="IsDeleted" type="xsd:boolean" default="false"/>
  <xsd:attribute name="CustomerDefined" type="xsd:boolean" default="false"/>
  <xsd:attribute name="Description" type="xsd:string" use="optional"/>
  <xsd:attribute name="Version" type="xsd:string" use="optional"/>
</xsd:complexType>
<xsd:complexType name="FileType">
  <xsd:complexContent>
    <xsd:extension base="AbstractSignatureType">
      <xsd:attribute name="Name" type="xsd:string" use="required"/>
      <xsd:attribute name="Size" type="xsd:long" default="0"/>
      <xsd:attribute name="Scope" type="SignatureScopeType" default="Any"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="XslmIdType">
  <xsd:complexContent>
    <xsd:extension base="AbstractSignatureType">
      <xsd:attribute name="PublisherID" type="SixteenBytes" use="required"/>
      <xsd:attribute name="ProductID" type="EightBytes" use="required"/>
      <xsd:attribute name="VersionID" type="EightBytes" use="required"/>
      <xsd:attribute name="FeatureID" type="EightBytes" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="InstRegType">
  <xsd:complexContent>
    <xsd:extension base="AbstractSignatureType">
      <xsd:attribute name="Key" type="xsd:string" use="required"/>
      <xsd:attribute name="Data" type="xsd:string"/>
      <xsd:attribute name="Source" type="InstRegSourceType" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType><xsd:complexType name="WinRegType">
  <xsd:complexContent>
    <xsd:extension base="AbstractSignatureType">
      <xsd:attribute name="Key" type="xsd:string" use="required"/>
      <xsd:attribute name="Data" type="xsd:string"/>
      <xsd:attribute name="Type" type="WinRegDataType" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="AppServerType">
  <xsd:complexContent>
    <xsd:extension base="AbstractSignatureType">
      <xsd:attribute name="Name" type="xsd:string" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

```

<xsd:complexType name="J2EEAppType">
  <xsd:complexContent>
    <xsd:extension base="AbstractSignatureType">
      <xsd:attribute name="Name" type="xsd:string" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ExtSigType">
  <xsd:complexContent>
    <xsd:extension base="AbstractSignatureType">
      <xsd:all>
        <xsd:element name="Body" type="xsd:string"/>
      </xsd:all>
      <xsd:attribute name="Key" type="xsd:string" use="required"/>
      <xsd:attribute name="Scope" type="SignatureScopeType" default="Any"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType><!-- ===== -->
<!-- Catalog elements definitions -->
<!-- ===== -->
<!-- The Component type extends an AbstractComponentType, which collects
common attributes -->
<xsd:complexType name="AbstractComponentType" abstract="true">
  <xsd:attribute name="Name" type="xsd:string" use="required"/>
  <xsd:attribute name="Description" type="xsd:string" use="optional"/>
  <xsd:attribute name="VendorID" type="xsd:IDREF" use="required"/>
</xsd:complexType>
<!-- A concrete Component must have at least one associated Version -->
<xsd:complexType name="ComponentType">
  <xsd:complexContent>
    <xsd:extension base="AbstractComponentType">
      <xsd:sequence maxOccurs="unbounded">
        <xsd:element name="CVersion" type="ComponentVersionType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<!-- A Component is identifiable by an arbitrary set of Signatures (IDREFS)-->
<!-- IsSimple means that the component is not an aggregate component -->
<xsd:complexType name="ComponentVersionType">
  <xsd:attribute name="ID" type="ExternalIdType" use="required"/>
  <xsd:attribute name="Value" type="ComponentVersionValueType"
    use="required"/>
  <xsd:attribute name="SupportedPlatform" type="xsd:IDREF" use="required"/>
  <xsd:attribute name="IsDeleted" type="xsd:boolean" default="false"/>
  <xsd:attribute name="Signatures" type="xsd:IDREFS" use="optional"/>
  <xsd:attribute name="DisabledMonitoringSignatures" type="xsd:IDREFS"
    use="optional"/>
  <xsd:attribute name="DisabledRecognitionSignatures" type="xsd:IDREFS"
    use="optional"/>
  <xsd:attribute name="DisabledAnySignatures" type="xsd:IDREFS"
    use="optional"/>
  <xsd:attribute name="IsSimple" type="xsd:boolean" default="false"/>
</xsd:complexType>
<!-- A Product is the composition of an arbitrary number of software
components; we need the possibility to use the Name, Description,
Vendor attributes at each level of the product version hierarchy,
because a product may change their values starting from a given
Version or Release. This implies that the Name, Description and
VendorID attributes are optional for the Version and Release elements
and, if someone is not specified, the value of the ancestor element is
considered -->
<xsd:complexType name="AbstractProductType" abstract="true">
  <xsd:attribute name="ID" type="ExternalIdType" use="required"/>
  <xsd:attribute name="IsIBM" type="xsd:boolean" default="false"/>
  <xsd:attribute name="LicenseType" type="xsd:int" default="2"/>
  <xsd:attribute name="IsDeleted" type="xsd:boolean" default="false"/>

```

```

        </xsd:complexType>
<xsd:complexType name="ProductType">
  <xsd:complexContent>
    <xsd:extension base="AbstractProductType">
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="Version" type="ProductVersionType"/>
      </xsd:sequence>
      <xsd:attribute name="Name" type="xsd:string" use="required"/>
      <xsd:attribute name="Description" type="xsd:string" use="optional"/>
      <xsd:attribute name="VendorID" type="xsd:IDREF" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<!-- Name and Description are optional attributes; if not specified, use the values
      defined by the ancestor element -->
<xsd:complexType name="ProductVersionType">
  <xsd:complexContent>
    <xsd:extension base="AbstractProductType">
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="Release" type="ProductReleaseType"/>
      </xsd:sequence>
      <xsd:attribute name="Value" type="ProductVersionValueType" use="required"/>
      <xsd:attribute name="Name" type="xsd:string" use="optional"/>
      <xsd:attribute name="Description" type="xsd:string" use="optional"/>
      <xsd:attribute name="VendorID" type="xsd:IDREF" use="optional"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<!-- Name and Description are optional attributes; if not specified, use the values
      defined by the ancestor element -->
<!-- If the Platforms attribute is not specified, All platforms are supported -->
<!-- Managed Components are always chargeable -->
<xsd:complexType name="ProductReleaseType">
  <xsd:complexContent><xsd:extension base="AbstractProductType">
    <xsd:attribute name="Value" type="ProductReleaseValueType" use="required"/>
    <xsd:attribute name="Name" type="xsd:string" use="optional"/>
    <xsd:attribute name="Description" type="xsd:string" use="optional"/>
    <xsd:attribute name="VendorID" type="xsd:IDREF" use="optional"/>
    <xsd:attribute name="SupportedPlatforms" type="xsd:IDREFS" use="required"/>
    <xsd:attribute name="Components" type="xsd:IDREFS" use="optional"/>
    <xsd:attribute name="NoChargeComponents" type="xsd:IDREFS" use="optional"/>
    <xsd:attribute name="ManagedComponents" type="xsd:IDREFS" use="optional"/>
    <xsd:attribute name="Products" type="xsd:IDREFS" use="optional"/>
    <xsd:attribute name="NoChargeProducts" type="xsd:IDREFS" use="optional"/>
    <xsd:attribute name="ManagedProducts" type="xsd:IDREFS" use="optional"/>
  </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<!-- ===== -->
<!-- Other complex types definitions -->
<!-- ===== -->
<xsd:complexType name="PlatformType">
  <xsd:attribute name="ID" type="PlatformIdType" use="required"/>
  <xsd:attribute name="Name" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="VendorType">
  <xsd:attribute name="ID" type="VendorIdType" use="required"/>
  <xsd:attribute name="Name" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="SupersedeType">
  <xsd:attribute name="NewID" type="SupercededIdType" use="required"/>
  <xsd:attribute name="OldID" type="SupercededIdType" use="required"/>
</xsd:complexType><!-- ===== -->
<!-- Symple Types Definitions -->
<!-- ===== -->
<xsd:simpleType name="PlatformIdType">
  <xsd:restriction base="xsd:ID">

```

```

        <xsd:pattern value="p\d{1,9}"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="VendorIdType">
    <xsd:restriction base="xsd:ID">
        <xsd:pattern value="v\d{1,9}"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="SignatureIdType">
    <xsd:restriction base="xsd:ID">
        <xsd:pattern value="s\d{1,9}"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="ExternalIdType">
    <xsd:restriction base="xsd:ID"/>
</xsd:simpleType>
<xsd:simpleType name="SupercededIdType">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="IBM_\d{1,9}"/>
    </xsd:restriction>
</xsd:simpleType>
<!-- A Product Version value is a number optionally followed by a pound and a
    star (n.*) -->
<xsd:simpleType name="ProductVersionValueType">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d+(\.\\*)?"/>
    </xsd:restriction>
</xsd:simpleType>
<!-- A Product Release value is a number, followed by a pound and a second number,
    optionally followed by a pound and a star (n.m.*) -->
<xsd:simpleType name="ProductReleaseValueType">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d+\.\d+(\.\\*)?"/>
    </xsd:restriction>
</xsd:simpleType>
<!-- A Component Version value is a number, optionally followed by an
    arbitrary sequence of dot separated numbers (n.m, n.m.x,...) -->
<xsd:simpleType name="ComponentVersionValueType">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d+((\.\d+)+)?"/>
    </xsd:restriction></xsd:simpleType>
<xsd:simpleType name="SignatureScopeType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Recognition"/>
        <xsd:enumeration value="Monitoring"/>
        <xsd:enumeration value="Any"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="SixteenBytes">
    <xsd:restriction base="xsd:hexBinary">
        <xsd:length value="16"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="EightBytes">
    <xsd:restriction base="xsd:hexBinary">
        <xsd:length value="8"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="InstRegSourceType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="OS"/>
        <xsd:enumeration value="ISMP"/>
        <xsd:enumeration value="Any"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="WinRegDataType">
    <xsd:restriction base="xsd:string">

```

```

    <xsd:enumeration value="binary"/>
    <xsd:enumeration value="dword"/>
    <xsd:enumeration value="expand_string"/>
    <xsd:enumeration value="multi_string"/>
    <xsd:enumeration value="string"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="DateType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="20\d\d-[0-1]\d-[0-3]\d 12:00:00.000000"/>
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>

```

Appendix E. Installing and uninstalling Common Inventory Technology (CIT)

Installing Common Inventory Technology (CIT)

Tivoli Configuration Manager Version 4.3.1 requires Common Inventory Technology (CIT) Version 2.5. When a Tivoli Configuration Manager activity is started on a workstation, Tivoli Configuration Manager checks if Common Inventory Technology (CIT) is already installed and installs it, if it is not found. CIT is installed using default paths, so during the installation the following directories are created:

CIT ini file directory

Where the cit.ini file and other programs needed during the installation are stored. This directory is /etc/cit on UNIX platforms, and %WINDIR%\cit on Windows platforms. The cit.ini file contains the list of each exploiter which installed CIT.

CIT product binaries directory

Where the binary files of the product are stored. This directory is /opt/tivoli on UNIX, and C:\Program Files\Tivoli\cit on Windows platforms.

CIT log and trace files directory

Where the log and trace files of the product are stored. This directory is /usr/ibm/tivoli/common/CIT/logs on UNIX platforms, and C:\Program Files\ibm\tivoli\common\CIT\logs on Windows platforms.

If you update from IBM Tivoli Configuration Manager, Version 4.2.3 Fix Pack 1 to IBM Tivoli Configuration Manager, Version 4.2.3 Fix Pack 3, the data created with CIT 1.1.1 is removed. For example, files such as wscanner*, libInvHW.*, libInvSW.* and libInvReg.* are removed from the lcf_root/inv/SCAN directory.

To enable the Inventory endpoint traces, run the **wdistinv** command using the **inv_ep_debug** option. Every time an Inventory scan is run on the endpoint, a log file called INVxxxxx.LOG (where xxxxx is the scan_id) is created under the inv/SCAN directory.

As an alternative to using default paths, you can install the CIT binary files in other directories by performing the following steps:

1. If you have already installed CIT manually when installing IBM Tivoli Configuration Manager, Version 4.3.1, run the following command to remove the installed software packages:

```
$BINDIR/./generic/inv/SCRIPTS/CIT_import.pl -uninst
```

If you have not previously installed CIT, move to step 2 on page 276.

2. On the Inventory server, run the following command:

```
$BINDIR/./generic/inv/SCRIPTS/CIT_import.pl -d CD_ROM/CIT_SPB
```

where

CD_ROM/CIT_SPB

Specifies the path where the CIT_Preinstall.spb and CIT.spb software packages are located on the IBM Tivoli Configuration Manager, Version 4.3.1 CD 1.

You can also copy the software packages to a local folder and run the command locally.

3. As a result, two software packages are created in a new profile manager called Inventory_CIT_PM.
4. On the Inventory server, run the following command to install the two software packages (CIT_Preinstall.CIT and CIT.2.3.1012) on custom paths:

```
winstsp -f -DCIT_ExploiterID=ITCM -DCIT_DestinationDirectory=/mydir  
@CIT_Preinstall.CIT @Endpoint:ep1
```

where:

/mydir

Is the directory on the endpoint where CIT is installed.

ep1 Is the name of the endpoint.

5. On the Inventory server, run the following command:

```
winstsp -f -ty @CIT.2.5.1003 @Endpoint:ep1  
wcommstsp -f @CIT.2.5.1003 @Endpoint:ep1
```

where:

ep1 Is the name of the endpoint.

Note: This installation method is not supported on OS/400 systems.

This procedure must be completed before performing any Inventory or Tivoli License Manager operation based on CIT, because also Tivoli License Manager can install CIT on the same workstation. By following this procedure you can customize the CIT product binary directory, but you cannot customize the cit.ini directory and the CIT log files directory.

Another way to change the CIT installation directory is to overwrite the default CIT_DestinationDirectory variable by editing the CIT_Preinstall.spb software package. The modified CIT_Preinstall.spb should then be distributed against all the gateway bundle directories replacing the old software package.

After installing any Tivoli Configuration Manager fix pack or interim fix and before performing any Tivoli Configuration Manager activity on the endpoints of your environment, perform the following steps:

1. Make a backup copy of the old CIT_Preinstall.spb software package before replacing it with the new one.
2. Import the software package using the **CIT_import.pl** command. This command creates a profile manager called Inventory_CIT_PM and adds to the profile manager the following software packages:

```
CIT.2.5.1003  
CIT_Preinstall.CIT
```

Note: The CIT version 2.5.1003 might vary depending on which fix pack or interim fix you have installed.

3. Open the Inventory_CIT_PM profile manager and right-click CIT_Preinstall.CIT.

4. Select **Convert → Unbuild**.
5. Specify a path where to unbuild the software package, then select **Convert & Close**.
6. Open CIT_Preinstall.CIT using the software package editor and select **edit → variable list editor**. The default value for CIT_DestinationDirectory is:

```
$(Destination_$(os_family))$(FileSeparator)
tivoli$(FileSeparator)cit
```

Modify this value for example into

```
$(Destination_$(os_family))$(FileSeparator)
..$(FileSeparator)mydir
```

The new CIT installation directory is C:\Program Files\..\my_cit_dir on Windows workstations, or /opt/../../my_cit_dir path on UNIX.

7. Save the modified variable value.
8. Right-click again CIT_Preinstall.CIT.
9. Select **Convert**.
10. Specify the old name and path of the software package, select the **overwrite** check box, then select **Convert & Close**. The default path used by the new CIT_preinstall.spb software package has been updated.
11. Replace the CIT_preinstall.spb software package on all the gateway bundle directories of your environment. All the following software packages must be replaced with the new .spb file:

```
lcf_bundle.41100\lib\aix4-r1\inv\CIT\SPB\CIT_Preinstall.spb
lcf_bundle.41100\lib\hpux10\inv\CIT\SPB\CIT_Preinstall.spb
lcf_bundle.41100\lib\linux-ix86\inv\CIT\SPB\CIT_Preinstall.spb
lcf_bundle.41100\lib\linux-ppc\inv\CIT\SPB\CIT_Preinstall.spb
lcf_bundle.41100\lib\linux-s390\inv\CIT\SPB\CIT_Preinstall.spb
lcf_bundle.41100\lib\solaris2\inv\CIT\SPB\CIT_Preinstall.spb
lcf_bundle.41100\lib\solaris2-ix86\inv\CIT\SPB\CIT_Preinstall.spb
lcf_bundle.41100\lib\w32-ix86\inv\CIT\SPB\CIT_Preinstall.spb
```

Note: It is recommended to test the new variable on some endpoints, before deploying it to the entire environment.

Uninstalling Common Inventory Technology (CIT)

You can uninstall CIT using the Software Distribution component of IBM Tivoli Configuration Manager. This operation requires uninstalling CIT for all its registered exploiters. Before uninstalling CIT, verify that it is no longer required by any of the exploiters listed in the cit.ini file.

You must uninstall CIT software packages in the following order.

1. Uninstall the CIT_Preinstall.CIT software package for each registered exploiter. Each CIT exploiter is registered in the cit.ini file with the **CITExploiterID** key. The string identifying IBM Tivoli Configuration Manager is as follows:
CITExploiterID=ITCM.
2. Uninstall the CIT.2.3.1012 software package

To uninstall CIT using the Software Distribution component of IBM Tivoli Configuration Manager, perform the following steps:

1. Set the environment to run the disconnected Software Distribution command line.

2. Uninstall CIT for each exploiter registered in the cit.ini file. To perform this operation, run the following commands:
 - a. `set CIT_ExploiterID=exploiter_ID`
 where
 exploiter_ID
 Is the ID of the exploiter as registered in the cit.ini file. The ID for IBM Tivoli Configuration Manager is **ITCM**.
 - b. `wdrmsp -f CIT_Preinstall.CIT`
 - c. Repeat steps 2a and 2b for each registered exploiter.
3. After uninstalling the CIT_Preinstall.CIT software package for each registered exploiter, type the following command to complete the uninstallation:
`wdrmsp -f @CIT.2.3.1012`

Table 8 lists the errors which might occur if you do not correctly manage the list of exploiters in the cit.ini file:

Table 8. CIT return codes

Return code	Error reason	Recovery action
11	At least one exploiter is still registered in the cit.ini file.	Remove the exploiter ID as explained in step 2.
12	You have not specified the exploiter name.	Specify the exploiter name.
13	The exploiter you specified is unknown.	Check the exploiter name in the cit.ini file and specify it again.

Managing the Inventory bundle dependency set

You can add, remove, and verify the automatic download of IBM Tivoli License Manager and CIT using the standard Framework bundle dependency set. To perform this operation, use the **winvdeps** command.

winvdeps

Specifies dependency sets that a method needs to run.

Syntax

winvdeps -a *depset*

winvdeps -r *depset*

winvdeps -s *depset*

Description

The **winvdeps** command specifies dependencies that a method needs to run. You can use the command to add, remove, and verify a dependency set.

Options

-a Adds a dependency set.

-r Removes a dependency set.

-s Verifies the presence of a dependency set.

depset The name of the dependency set. Supported values are as follows:

CIT The name of the dependency set for CIT.

TLM The name of the dependency set for TLM.

Return Values

The **winvdeps** command returns one of the following:

0 Indicates that **winvdeps** started successfully.

-1 Indicates that **winvdeps** failed due to a generic error.

Examples

To accept add the CIT dependency set, enter the following command:

```
winvdeps -a CIT
```

See Also

None.

Scanning virtual environments

Inventory leverages the CIT capability to collect LPAR information in a VMware environment.

VMware versions supported by CIT are shown in the table below:

Table 9. Versions of VMware tested on CIT

VMWare versions tested on CIT
ESX Server 3.0
ESX Server 3.01
ESX Server 3.02
ESX Server 2.5.3

Standard Inventory functions are not impacted whether the scans are performed on guest or host operating systems. However, due to a limitation of the data export functionality in the virtualization software, hardware and software discovery operations performed on guest operating systems might report incorrect data.

CIT provides an enabler to export correct data to the guest systems so that CIT instances installed on each operating system partition can discover and return correct data.

The CIT enabler stores the correct hardware information on the guest operating systems. In this way when a hardware Inventory scan starts on a guest operating system, the correct hardware configuration data is retrieved and stored in the Inventory RDBMS. You can query the data by running the LOGICAL_PARTITION_QUERY and the LOGICAL_PARTITIONED_SYSTEMS_QUERY

Starting the enabler

The CIT enabler is installed on the physical workstation which hosts the guest operating systems. The CIT enabler does not require an installation process. To use the CIT enabler, copy the files listed below from the /cit_enabler folder on the IBM Tivoli Configuration Manager, Version 4.3.1 CD 1 to the workstation which hosts the guest operating systems:

Table 10. Enabler executable files

Virtual software	Supported OS	Files
VMware	Microsoft Windows	<ul style="list-style-type: none">• cpuid.exe• retrieve.pl• wenvmw.exe
	Linux	<ul style="list-style-type: none">• cpuid• dispatcher• retrieve.pl• wenvmw.sh

You must install VMware VmPerl Scripting API before starting the enabler for the first time.

To start the enabler, launch the executable file from a shell or command prompt. You can optionally specify the `-v` option to generate a log file named `env_out.txt`, which is created in the same directory where the executable file is located.

Notes:

1. The guest workstations must be active when the enabler is started.
2. Any guest workstations not active when the enabler is running, are not detected.
3. If you run the enabler after the configuration changes are applied to the guest workstation, the updated data is returned by the hardware scan.
4. The information retrieved by the enabler is deleted after a reboot.

Enabler return codes

The following is a list of the return codes returned by the enabler. Return codes help you identify the result of the command.

A return code of 0 indicates that the command completed successfully:

OK: CIT ENABLER return code = 0

while a nonzero return code indicates that an error occurred. The following example shows the case in which the user does not have enough privileges, when running the CIT enabler executable:

OK: CIT ENABLER return code = 11
ERROR: dispatcher return code = 11"

A complete list of all nonzero return codes is provided in the table below:

Table 11. Enabler return codes

Return value	Code	Description
RETRIEVE_NODECAPACITY_ERROR	1	Cannot collect host system processor number.
RETRIEVE_NODEID_ERROR	2	Cannot collect host system serial number.
RETRIEVE_VMCAPACITY_ERROR	3	Cannot collect number of processors assigned to virtual machines.
RETRIEVE_VMID_ERROR	4	Cannot collect virtual machines IDs.
SET_GUESTINFO_ERROR	5	Cannot transfer information from host system to guest systems.
GET_VMLIST_ERROR	6	Cannot retrieve list of registered virtual machines.
CONNECT_VM_ERROR	7	Cannot establish connection with virtual machines.
NO_PRIVILEGES	11	User does not have enough privileges when running the CIT enabler executable.
GENERIC_ERROR	-1	A generic error has occurred. For more information, enter the command again with the <code>-v</code> option. The resulting information is logged into the <code>env_out.txt</code> file.
VMWARENOTFOUND_ERROR	100	VMware Server not found on host system.

Table 11. Enabler return codes (continued)

Return value	Code	Description
VMWARENOTSUPPORTED_ERROR	105	VMware Server version not supported.
VMPERL_NOT_FOUND	120	VmPerl Scripting API not found or incorrectly configured.

Appendix F. Troubleshooting

This appendix describes files and procedures that can help you research and resolve problems that you might encounter while using Inventory. In general, check the Inventory notice group for troubleshooting information before consulting the sources described in this appendix. See the *Tivoli Management Framework: User's Guide* for information about notice groups.

AutoTrace

Inventory features AutoTrace, a process-tracing diagnostic tool. AutoTrace collects information during run-time and stores the information for problem determination. Unlike standard file-based logging where an error is written to a file, AutoTrace stores a record of the functions for each process call in memory buffers. When an error occurs, AutoTrace can write the trace information to a file. You can then send the trace file to Tivoli Customer Support personnel, who use the information to trace a problem to its source or to determine why the error occurred. AutoTrace is available on the following platforms: Solaris, HP-UX, AIX, Linux on PC systems, Windows NT, and Windows 2000.

AutoTrace provides constant, in-depth tracing. Input/output operations such as writing to a file are kept to a minimum. This allows continuous tracing so that the data is available the first time a problem occurs, known as first failure data capture (FFDC). Data collection in the memory buffers wraps so that the latest information is always available.

By default, AutoTrace for Inventory is disabled. Contact Tivoli Customer Support for information about enabling and using AutoTrace.

Configuration repository schema

To troubleshoot problems encountered when creating the configuration repository, check the log file named `inv_rdbms_type_admin.log`, where *rdbms_type* is the type of RDBMS you are using for the configuration repository. This variable can be one of the following values: `db2`, `infx`, `ms_sql`, `ora`, or `syb`. The success or failure of the SQL statements are logged in this log file. Check your RDBMS documentation to determine the location of this log file.

DMI

This section contains information about troubleshooting DMI scans.

Adding a DMI layer

Before you can scan for DMI data, you must configure DMI scanning options. As a part of this process, you must add each type of DMI layer you want to scan to the table in the DMI Scanner Configuration window. Before you add a DMI layer, make sure that you have distributed an inventory profile at least once to the endpoint on which the DMI layer is installed. An inventory profile distributes the following files to an endpoint: `dmiscan.exe`, `wcdmi.dll`, and `wdmiutil.dll`. These files must be present on the endpoint before you can retrieve the DMI layer.

To configure a DMI layer, you must have the **admin** role. This role enables you to write to \$BINDIR on the Tivoli server and \$DBDIR on the local managed node. If the DMI layer on an endpoint does not display in the DMI Scanner Configuration window, you might not have the correct authorization.

The following errors might occur when you attempt to add a DMI layer to the table in the DMI Scanner Configuration window. Troubleshooting information is provided for each error.

- **Could not find endpoint.**

To troubleshoot this error message, run the following command to ensure that the endpoint is available:

```
wadminep endpoint_name view_log_file
```

where:

endpoint_name

Specifies the name of the endpoint.

view_log_file

Displays a log file for the endpoint.

If you cannot view the log file, the endpoint is not available. Troubleshoot the endpoint and the connection between the endpoint and its gateway.

- **Could not examine DMI layer.**

A DMI layer exists on the specified endpoint, but the examination of the DMI layer timed out. Retry the operation later.

- **No DMI layer exists on this endpoint.**

The endpoint you specified does not have a DMI layer. Install a DMI layer on this endpoint, or specify a different endpoint that has the same type of DMI layer.

- **Could not retrieve DMI data.**

There is a communication problem between the endpoint and the gateway to which it belongs. To troubleshoot this error message, run the following command:

```
wadminep endpoint_name view_log_file
```

where:

endpoint_name

Specifies the name of the endpoint.

view_log_file

Displays a log file for the endpoint.

If you cannot view the log file, the endpoint is not available. Troubleshoot the endpoint and the connection between the endpoint and its gateway.

- **DMI data format is not recognized.**

To troubleshoot this error message, perform the following tasks:

- Specify a different endpoint that has the same type of DMI layer.
- Examine the DMI layer on the endpoint using a DMI browser, such as Intel DMI Explorer, to determine whether the data is corrupted or contains an unrecognized value.
- Reinstall the DMI layer on the endpoint.

For more information about using the DMI Scanner Configuration window and configuring DMI scan options, see the Inventory online help.

DMI scans

If a DMI scan fails, perform the following tasks to troubleshoot the scan:

- Verify that a DMI layer exists on the endpoint for which the scan failed. For PCs that do not contain a DMI layer, a DMI scan will return a value of NO_DATA for all attributes. To check whether a DMI layer is installed on an endpoint, follow the procedure to add the DMI layer from the endpoint to the list of DMI layers in the DMI Scanner Configuration window. If no DMI layer is installed on that endpoint, the following message is displayed:

No DMI layer exists on this endpoint.

For more information about the procedure to add a DMI layer, see the Inventory online help.

- Make sure that the file dmi.ini was distributed to the endpoint. This file contains the DMI scan settings. The inventory profile must successfully distribute this file to the endpoint for a DMI scan to complete.
- Run the DMI scan on the endpoint from the command line. This can help you determine whether the scan failed because of a communication problem between the endpoint and the managed node from which you attempted the DMI scan. The DMI scanner is located in \$LCF_INSTDIR/inv/SCAN, where \$LCF_INSTDIR is the directory in which the endpoint is installed. Run the DMI scan using the following command:

```
dmiscan -i dmi.ini -o dmican.mif
```

If this scan fails, check the DMI.INI file to verify that it is formatted correctly.

DMI table and column names

The `dmi_RDBMS_type_schema.sql` script creates custom tables in the configuration repository for DMI scan information. The default name for each custom table is the DMI group name truncated to 18 characters, with underscores (`_`) substituted for blank spaces. The default names for columns in each table are the DMI attribute names truncated to 18 characters, with underscores substituted for blank spaces.

Because these names are generated automatically, it is possible that the script will use SQL-92 reserved words for table and column names. If this happens, you must rename the table or column. For more information about SQL-92 reserved words, consult a database administrator or SQL documentation.

It is also possible that the script will use characters that are not valid in table or column names. You can name tables and columns using the following characters:

- Lowercase a through z
- Uppercase A through Z
- Numeric characters 0 through 9
- Underscores

You cannot use spaces. Table and column names must start with a letter.

Endpoints

This section contains instructions for troubleshooting the Inventory endpoints.

Enabling endpoint log files

Using the `wdistinv` command and the `-l` option and the `inv_ep_debug` keyword, you can configure an inventory profile to collect debugging information from scanned endpoints. You can also set the level of debugging information to be

logged. The debugging information for each endpoint is saved on the endpoint in a log file in the \$LCFROOT/inv/SCAN directory. These log files are stored on the inventory data handler in the \$DBDIR/inventory/data_handler/logfiles/*scan_ID*/*ep_name* directory, where:

- *scan_ID* is the ID of the scan.
- *ep_name* is the name of the endpoint for which debugging information was collected.

The log files are named INV*scan_ID*.LOG, where *scan_ID* is the ID of the scan that generated and collected the log file. On the endpoint, the log files are deleted after each successful scan. On the inventory data handler, these log files persist until you delete them, or until they are overwritten by an identically named log file.

For more information about the **wdistinv** command, see “wdistinv” on page 122.

You can also collect debugging information from endpoints using the **wepscan** command and **-d** option. The debugging information is stored on the endpoint in a log file named INV_SA.LOG. When you run **wepscan** using the **-s** option, this log file is sent to the inventory data handler, where it is stored in the \$DBDIR/inventory/data_handler/logfiles/0/*ep_name* directory. This file is overwritten each time you run the **wepscan** command and the **-d** and **-s** options. You also use the **wepscan** command and **-d** option to set the level of debugging information to be logged.

You can also create log files for endpoints by creating an environment variable on the endpoint named WEPSCAN_DEBUG. This environment variable creates the same log files as the **wepscan** command and **-d** option. This is especially helpful when the **wepscan** command is not available but you want to create log files, for example when you initiate a scan using the Web Interface. The log files are created each time you scan the system using the Web Interface or the **wepscan** command.

For more information about the **wepscan** command and the WEPSCAN_DEBUG environment variable, see “wepscan” on page 127.

Installing multiple endpoints on the same workstation

To install multiple endpoints on the same workstation, it is necessary to ensure that each endpoint is installed on a different root directory and has a different COMPUTER_SYS_ID. The COMPUTER_SYS_ID is associated with the global GUID (Globally Unique Identifier) created during the Tivoli Agent installation.

To ensure that each endpoint has its own COMPUTER_SYS_ID, run for each endpoint installed on the same workstation the following commands:

```
wep endpoint_name set_config GUID="USE_LCF_ID"  
wep endpoint_name set_config lcfid="local_lcf_id_string"
```

where "local_lcf_id_string" is the string that you want to specify as COMPUTER_SYS_ID. It must be different for each endpoint.

For example, if on the same workstation the **ep1** and **ep2** endpoints are installed, you should run:

```
wep ep1 set_config GUID="USE_LCF_ID"  
wep ep1 set_config lcfid="ep1_computer_sys_id"  
wep ep2 set_config GUID="USE_LCF_ID"  
wep ep2 set_config lcfid="ep2_computer_sys_id"
```

By running the following commands:

```
wadminep endpoint_name view_machine_id
```

```
wadminep endpoint_name view_GUID
```

you can check the values which have been assigned to GUID and machine_id.

Endpoint-initiated scans

To troubleshoot an endpoint-initiated scan, it is helpful to understand how it differs from scans initiated by a profile distribution. A scan of an endpoint initiated by a profile distribution is composed of the following steps:

- The endpoint is scanned, and then the scan data is saved on the endpoint in a Management Information Format (MIF) file. The MIF file is named *scan_type.mif*, where *scan_type* is the type of scan. For example, the Tivoli hardware scanner generates *tivhscan.mif*, and the DMI scanner generates *dmiscan.mif*.
- Assuming the inventory profile is configured to send the scan results to the configuration repository, the endpoint method parses the MIF file and renames it with a .BK1 extension. The data parsed from the MIF file is written to a .dat file (an encoded and compressed binary file). The .dat file is named *invscanID.dat*, where *scanID* is the numerical value that identifies the scan.
- The .dat file is sent through the collector hierarchy to the configuration repository.

Endpoint-initiated scans are controlled using the **wepscan** command. With this type of scan, the MIF file is not renamed with a .BK1 extension until you send the data to the configuration repository using the **wepscan** command and **-s** option. In other words, if you run two endpoint-initiated scans consecutively without sending the data to the configuration repository, the second scan overwrites the MIF file generated by the first scan. The data from the first scan is never collected. This is done to ensure that the .BK1 file always contains the data that was last sent to the configuration repository (not necessarily data from the last scan that was run). This design ensures that, if you choose to update with differences, the scan always updates the database with the correct information. (A scan that updates with differences compares the MIF file of the current scan with the .BK1 scan and sends only new or changed data to the configuration repository.)

If you run the **wepscan** command with the **-d** option, the command creates log files called *sa_results.log* and *sa_config.log*. If you run the **wepscan** command with the **-c** option, the command creates a log file called *sa_config.log*. These log files are stored in the directory from which you run the **wepscan** command. The log files contain the following information:

sa_results.log

The contents of the .dat file. Check this file to see what data was collected by the scan. This data will be sent to the configuration repository when the scan data is collected.

sa_config.log

The configuration of the profile. Check this file to see what software and hardware information the profile is configured to collect, and other profile configuration information.

Before you run an endpoint-initiated scan, make sure that you have performed the following tasks:

- Distributed an inventory profile to the endpoint so that the configuration file, *config.dmp*, is installed on the endpoint. This file must be present for the

endpoint-initiated scan to succeed. To verify that config.dmp is installed on the endpoint, check the inv/SCAN directory in the directory in which the endpoint is installed.

- Set up your environment to access the shared libraries needed by the command. For more information about setting up your environment to use **wepscan**, see “wepscan” on page 127.

Graphical user interface

This section contains instructions for troubleshooting the Inventory graphical user interface (GUI).

System requirements

The GUI for Inventory requires the Java Runtime Environment (JRE), which can be installed from the Tivoli Management Framework CD. Not all operating systems that are supported by Inventory are compatible with the JRE. For more information, see the *Release Notes* and the documentation for your operating system.

Log files

The Inventory GUI provides information useful for debugging in log files.

UNIX Systems

On UNIX systems, you can configure the Inventory GUI to write messages to log files. You can then use the log files to debug the GUI. These log files are as follows:

sh.out Contains the path locations of the Java classes. For example, this log file lists the settings of the environment variables used by the GUI, such as the LANG (language) variable.

DEBUG3

Contains the debug information provided by the GUI Java code.

These files contain information about the current session of the GUI only. Each time you start the GUI, the log file from the previous session is overwritten.

To enable log files for the GUI on UNIX systems, use following procedure:

1. On the system on which you are using the GUI, open the inv_gui.sh file in a text editor. This file is located in \$BINDIR/TME/INVENTORY.
2. In the inv_gui.sh file, locate the variable DEBUG. By default, this variable is set to 0, and no log files are created.
3. Set the DEBUG variable to one of the following values:
 - 1 Create the sh.out log file.
 - 2 Create the sh.out and DEBUG3 log files.

By default, sh.out and DEBUG3 are stored in the \$DBDIR directory. To change the directory in which the sh.out and DEBUG3 log files are stored, edit the DEBUGFPATH variable in the inv_gui.sh file.

Note: The user logged into the GUI must have write permission to the directory specified by the DEBUGFPATH variable. If the user of the GUI does not have write permission, the log files will not be created.

PC Systems

On PC systems that use the Tivoli Desktop for Windows from Tivoli Management Framework, Version 4.1 and later, the Inventory GUI creates the log file `InvGuiDebug.log` in the `$DSWIN` directory. This file contains the debug information provided by the GUI Java code. It contains the same information as the `DEBUG3` log file.

On PC systems that use an earlier version of Tivoli Desktop for Windows, you can configure the Inventory GUI to write messages to the `sh.out` and `DEBUG3` log files. See “UNIX Systems” on page 286 for more information about enabling these log files.

Running the GUI on UNIX systems

To run the Inventory GUI on UNIX systems, you must enable connections to the X Window System on the UNIX system. This is necessary even if the GUI runs on the same machine as the X Window System display. To do this, follow these steps:

1. Set the `DISPLAY` environment variable to the X Window System display on which to display the GUI.

For example, to open the GUI on the X Window System display named `zeus:0.0`, a Bourne or Korn shell user would enter the following commands:

```
DISPLAY=zeus:0.0
export DISPLAY
```

In some instances, you might need to specify an IP address for the `DISPLAY` variable rather than the system name.

2. Enable remote connections to the X Window System. For example, to start the GUI on the `zeus` display, enter the following command:
`xhost +zeus`

Ensure that remote logins are enabled on the system on which you start the GUI. To ensure that remote logins are enabled, enter the following command:

```
odadmin odinfo
```

The following message is displayed:

```
Remote client login allowed = value
```

where *value* is either **TRUE** or **FALSE**.

If **FALSE**, enter the following command to enable remote client logins:

```
odadmin set_allow_rconnect TRUE
```

Logging in

On UNIX systems, when you double-click an Inventory profile, you are prompted to log in to the Inventory GUI. You must enter the Tivoli management region server (Tivoli server) name, login name, and password. If the Tivoli server is not on port 94, you must enter the Tivoli server name and the port number in the **Host Machine** field. You must enter this information in the format *server_name:port_number*. For example, if a Tivoli server named `gilberto` uses port 8860, you would enter the server name as follows:

```
gilberto:8860
```

Properties window does not open

The properties window for an inventory scan profile does not start when launched from the Tivoli desktop.

Check whether the **odadmin odlist** command returns the fully qualified hostname. In case the fully qualified hostname is not returned, add it to the aliases list using the **odadmin add_hostname alias** command. For more information on the **odadmin odlist** command, refer to the *Tivoli Management Framework Reference Manual*

Scalable Collection Service

This section contains information about troubleshooting Scalable Collection Service (SCS).

Log and data files

The following sections describe SCS-related log and data files. These files reside in one of the following directories, unless a different directory is specified:

- On the inventory data handler—`$DBDIR/inventory/data_handler/file_name`
- On all other collectors—`$DBDIR/mcollect/file_name`

where:

\$DBDIR

Is the Tivoli object database directory on the managed node

file_name

Represents one of the file names listed in the following sections

Queue Data Files

SCS logs information about the queue data of a collector in the following queue data files:

- `checkpointGL_iqfile.dat`—Lists the collection tables of contents (CTOCs) in the input queue of the collector.
- `checkpointGL_oqfile.dat`—Lists the CTOCs in the output queue of the collector.
- `checkpointGL_eqfile.dat`—Lists the CTOCs in the error queue of the collector. CTOCs move to the error queue of a collector when the collector has made an unsuccessful attempt to send out or take in information. The CTOC remains in the error queue until the collector makes the maximum number of retries. After the collector reaches the maximum number of retries, it sends an error notification and removes the CTOC from all its queues.
- `checkpointGL_dqfile.dat`—Lists the CTOCs in the deferred queue of the collector. CTOCs exist in the deferred queue when links to the collector are turned off. For more information about turning off links, see “Scheduling collections” on page 20 and “wcollect” on page 110.

Note: The .dat files for the queues exist so that if a collector shuts down, SCS can recover and continue processing the collected data where it left off.

You can view these files using the **wcstat -q** command.

These files are 4 bytes in size when empty and grow in size when the corresponding queue contains data. You can check the size of these files to

troubleshoot queues. For example, to verify that the input queue contains data, you can verify that the queue data file `checkpointGL_iqfile.dat` is greater than 4 bytes in size.

Collector Log File

The `mcollect.log` file contains all the activity of a collector as data flows through it. You control the amount of information that is logged in this file by setting a value with the `wcollect -d` option. By default, only fatal errors are logged. You specify the maximum size of this file using the `wcollect -g` option. By default, the maximum size is 1 megabyte (MB). For more information, see “wcollect” on page 110.

Watch the logging activity as it occurs by running the **tail** command on the collector:

```
tail -f mcollect.log
```

After you run this command, the log file displays a scrolling list of logging activity. If the log file display freezes in the middle of a collection, the file has reached its maximum size. Repeat the **tail** command to resume watching the log activity.

As you watch the log activity, watch for the strings **WARNING**, **ERROR**, and **exception**, which indicate error conditions.

Inventory Data Handler Log File

The inventory data handler creates a log file similar to the log file created by each collector. The file name is the same, `mcollect.log`, and you configure and monitor this log file using the same commands that you use for the collector log file.

As you monitor this log file, watch for the string **IR**, which indicates an inventory data handler message. For more information on configuring and monitoring this log file, see “Collector Log File” on page 289.

If the inventory data handler is having trouble connecting to the configuration repository, check `mcollect.log` for messages. The inventory data handler writes messages to this log file that indicate which RIM object it is trying to use to connect to the configuration repository.

Collection Manager Log File

The collection manager is installed on the same system as the Tivoli server. The collection manager log file is `$TMPDIR/mcollect_collmgr.log` on the Tivoli server, where `$TMPDIR` is a temporary directory. This temporary directory is usually one of the following paths:

- For UNIX systems: `/tmp`, `/usr/tmp`, or `/var/tmp`
- For Windows systems: `c:\temp`

This log file contains the routing information that the collection manager provides to the collectors. You can view this log file using the **tail** command or a text editor. To view this file using the **tail** command, run the command as follows:

```
tail -f mcollect_collmgr.log
```

Inventory Status Log File

By default, Inventory sends status information about scans to the Inventory notice group only. Using the `wsetinvglobal -l` command, you can configure Inventory to send status information to a log file.

The default path for this log file is \$TMPDIR/inv_scan_*nn*.log where *nn* is the scan ID and \$TMPDIR is a temporary directory. The temporary directory is usually one of the following paths:

- For UNIX systems: /tmp, /usr/tmp, or /var/tmp
- For Windows systems: c:\temp

Note: On UNIX systems, \$TMPDIR is the temporary directory returned in the **tmpnam** system call. On Windows systems, \$TMPDIR is the temporary directory returned in the **GetTempPath** system call. For more information, consult the documentation for your operating system.

This log file provides the profile name, the start time, elapsed time, and the number of targets on which scans are completed or pending.

You can view this log file using the **tail** command or a text editor. To view this file using the **tail** command, run the command as follows:

```
tail -f inv_scan_nn.log
```

CTOC Completed Status Log File

You can configure a collector to maintain a log file that lists the CTOCs that have completed on that collector. For troubleshooting, you can check this log file to see whether a collection completed successfully on a collector. The path for this file is \$DBDIR/mcollect/CTOC_log.dat. To view the contents of this file, run the following command:

```
wcstat -q c collector
```

where *collector* is of the form **@ManagedNode:collector_name**, **@Gateway:collector_name**, or **@InvDataHandler:inv_data_handler**.

You use the **wcollect** command and **-f** option to specify whether or not the collector writes data to the CTOC_log.dat file. By default, collectors write data to the file. To check whether a collector is configured to write data to this file, run the following command:

```
wcollect collector
```

where *collector* is of the form **@ManagedNode:collector_name** or **@Gateway:collector_name**.

In the resulting output, check the value of log_completed_ctoc. A value of true specifies that the collector writes data to the log file; false specifies that it does not write data to the file.

Activity Planner and Web Interface

Activity Planner enables you to define a group of activities in an activity plan, submit the plan to be executed, and monitor the execution of the plan. Activities are single operations that are performed on a set of targets at specified times. After registering the Inventory plug-in, you can use Activity Planner to create an Inventory scan operation.

Activity Planner is notified about the status of an Inventory scan operation by the Inventory status collector component. The status collector resides on the same managed node where is installed the Inventory Data Handler.

In case of interconnected Tivoli regions, you might want to start an Inventory scan operation defined in an InventoryConfig profile created in a Tivoli region different

than the one where Activity Planner is installed. In this case, the InventoryConfig profile status collector of that Tivoli region is responsible to notify the Activity Planner.

In case the Data Handler and the Tivoli server are installed on different regions, and the Data Handler is on a different managed node than the Inventory server, to allow the status collector to notify the Activity Planner, you need to copy under the \$DBDIR directory of the Data Handler managed node the

default_apm_callback.conf file that contains the following row:

```
1483073514.1.2678APMEngine::APM
```

which represents the Activity Planner OID.

You can obtain this OID by running the **wlookup -ar ActivityPlanner** command on the Tivoli server, where Activity Planner is installed.

Using Activity Planner and Inventory on interconnected Tivoli regions, requires a two_way connection between the two regions. Otherwise, the status collector is not able to notify the Activity Planner.

The IBM Tivoli Configuration Manager Web Interface enables you select a profile and then scan your local system instead of distributing a scan from an inventory profile.

See the *User's Guide for Deployment Services* for information about using log files with Activity Planner and the IBM Tivoli Configuration Manager Web Interface.

Other Log Files

The following log files contain information you might find useful when troubleshooting, such as error messages. These log files are text files that you can view using the **tail** command or a text editor.

- \$LCF_DATDIR/lcfd.log—A Tivoli Management Framework message log file for capturing endpoint messages. This file resides on each endpoint. For SCS troubleshooting, you can check this log file for information about exceptions that SCS has created or connectivity problems. The \$LCF_DATDIR directory is created when you install Tivoli Management Framework. For more information about the lcfd.log file, see the *Tivoli Management Framework: Reference Manual*.
- /tmp/rim_db_log—The log file that contains tracing information for RDBMS interface module (RIM) objects. This file resides on the managed node where the RIM object is installed. You use the **wrimtrace** command to enable or disable tracing and to specify the information written to the log file. You can write interobject message (IOM) packet information, relational database management system (RDBMS) errors, or both to the RIM log file.

By default, the RIM log file is not created. You must enable tracing using the **wrimtrace** command to cause the RIM log file to be created.

Note: The tracing function is intended for debugging purposes. If enabled for extended periods of time, tracing can decrease performance and slow the processing of RIM calls considerably.

See the *Tivoli Management Framework: Reference Manual* for more information about the **wrimtrace** command and the RIM log file.

Depot contents

You can check the contents of a collector's depot by checking the depot/*CTOC_ID* directory, where *CTOC_ID* specifies a CTOC by identification number. The data represented by the CTOC is contained in the directory named with the ID for that CTOC. The default path for the depot directory is \$DBDIR/mcollect/depot on each collector.

You cannot view the files located in the depot directory, but you can check the directory to verify that it exists and see whether it contains data. You can also verify that the files in this directory contain data by verifying that the file size is greater than 0 bytes.

Troubleshooting procedures

This section covers resolutions for situations you might encounter when using SCS. For more information about the commands described in this section, see Appendix B, "Commands," on page 103 and the *Tivoli Management Framework: Reference Manual*.

- **The scan completes but the data is not in the database.**

In an SCS-enabled environment, an inventory profile completes when all targets have been physically scanned, but data has not necessarily been populated in the configuration repository.

The **wgetscanstat** command reports a scan as finished only when all collected data has also been stored in the configuration repository or a failure occurs. To verify that the data for a target has reached the configuration repository, run the command as follows:

```
wgetscanstat -a -s -f
```

You can also check the Inventory notice group or Inventory status log file for scan completion information, if the profile for the scan is configured to send notifications to those sources.

- **The scan seems to take too long to finish.**

First, use the **wgetscanstat** command to view the progress of the targets being scanned. If scans for most of the targets have completed, use the **wadminep** command to check whether the endpoints that have not completed are accessible.

Next, use the **wrimtest** command to check whether the RIM object is able to connect to the database. Run the command as follows:

```
wrimtest -l rim_object_label
```

where *rim_object_label* is the name of RIM object used by the inventory data handler.

Note: By default, only one RIM object, invdh_1, is used by the inventory data handler. However, it is possible to connect the inventory data handler to more than one RIM object. To properly diagnose the problem, you must perform this test on each RIM object used by the inventory data handler. For more information about RIM objects used by the inventory data handler, see "Creating and configuring RIM objects for the inventory data handler" on page 17

To exit the **wrimtest** utility, enter the **x** command option.

Also, check the names of the RIM objects to make sure they are named correctly. The RIM object created during installation for use by the inventory data handler

must be named **invdh_1**. The RIM object used for queries; the Inventory GUI; and the **winvfilter**, **winvrnode**, and **winvsig** commands must be named **inv_query**.

You should also make sure all RIM objects are configured correctly. Run the following command for each RIM object to check its configuration:

```
wgetrim rim_object_label
```

where *rim_object_label* is the name of RIM object. For the procedure to check the configuration of RIM objects configured to work with the inventory data handler, see “Creating and configuring RIM objects for the inventory data handler” on page 17.

- **The profile is not using SCS.**

To troubleshoot this problem, perform the following actions:

- Verify that SCS is installed on all gateways and managed nodes in your repeater hierarchy. First, run the **wrpt** command to verify your repeater hierarchy. Next, run the **wlsinst** command to verify that all repeaters have been installed with SCS.
- Run the **wgetinvdh** command to verify that the inventory data handler exists.
- Run the **wgetrim** command to verify that one or more RIM objects have been created for use with SCS.
- Verify that the platform type of the endpoint is supported by SCS.
- Verify that the platform type of the gateway is supported by SCS. If it is not, Inventory will not use SCS to collect from that gateway.

- **The depot on a collector is full.**

When a depot on a collector is full, the collector adds error information to the CTOC for the attempted collection and moves the CTOC into the error queue. From there the CTOC is passed up to the error queue of each collector in the hierarchy. When the CTOC reaches the inventory data handler, the inventory data handler sends a notification stating that the collection for the affected endpoint failed because the depot of an upstream collector is full.

To troubleshoot this problem, use the **wcollect** command to check the depot size. If the depot is too small, use the **-z** option of the **wcollect** command to increase the size of the depot on that collector, other collectors on the route, and the inventory data handler. For more information, see “wcollect” on page 110.

- **A collector has failed.**

Check the Inventory notice group and Inventory status information log file for errors.

Check to see whether an offlink has been set for the collector. For more information about offlinks see “Scheduling collections” on page 20 and “wcollect” on page 110.

Use the **wcollect** command to display the collector’s configuration. Check to see if the value for *max_input_retries* is set too low. For example, if this value is set to 1, the collector does not retry data collection after the first failure.

Check the network connectivity between all the collectors from the endpoint to the inventory data handler. If connectivity is lost between any two collectors in this route, the inventory data handler cannot receive error messages until the network is restored.

To recover a failed collector, use the **wcollect -h** command to halt the collector if it is still running. Then restart the collector using the **wcollect -s** command.

- **A collection has failed.**

Perform the same checks that you would for a failed collector, but check all the collectors in the route from the scan target to the inventory data handler. To determine the route, type the **wrpt** command with no arguments to determine the repeater hierarchy, and then reverse the hierarchy to determine the collector hierarchy.

Also check connectivity between RIM objects and the configuration repository.

- **The status collector incorrectly indicates that a completed scan is still running.**

If a non-recoverable error occurs for a target during a scan, and the status collector cannot be notified that a failure occurred, the status collector will think that the scan for that target is still pending and that the scan is not complete.

First, consider recording the information about the status of all the targets. You might not get a final notification, depending on the notification options you have set. The following command lists the information for the scan:

```
wgetscanstat -a -s -f -p
```

Next, reset the status collector by performing the following steps:

1. Kill the status collector process. The status collector process runs on the managed node that hosts the inventory data handler and is named `inv_stat_meths`. You can determine the name of this managed node using the **wgetinvdh -a** command. To determine the ID for the `inv_stat_meths` process on UNIX systems, use the **ps** command. To determine the ID for this process on Windows NT systems, use the Task Manager, which you can access by right-clicking the taskbar.

To kill the process on UNIX systems, use the **kill** command. To kill the process on Windows NT systems, select the process in Processes tab of the Task Manager, then click the **End Process** button.

2. Delete the status file for the scan that did not complete. In the directory `$DBDIR/inventory/stat_dir` on the managed node that hosts the inventory data handler, delete the file associated with the scan that failed. The filename is `inv_stat_n.cfg`, where *n* is the scan ID.

To ensure data integrity, the next time you perform a scan on any target that was listed as pending in the output from the **wgetscanstat** command, set the **Replace with current results** option for the profile.

- **The Inventory notification group contains a message stating that a restore operation could not be performed.**

When a scan is performed, the status collector maintains a file on disk so that if the status collector process is interrupted (for example, if the machine crashes), status can be restored. If the file on disk is corrupted when the status collector restarts, an error similar to the following is reported to the Inventory notice group:

```
The restore operation could not be performed for file
~/Tivoli/deploy/1/users/bob/inv_8333/db/fuji.db/inventory/
stat_dir/inv_stat_1.cfg' scan id `1'. The file contains
invalid data.
```

To delete this error, look in the directory `$DBDIR/inventory/stat_dir` on the inventory data handler managed node for files named `inv_stat_n.cfg`, where *n* is an integer. Remove the file that is specified in the error message. In the preceding example, the file is `inv_stat_1.cfg`.

- **A pending inventory scan is stuck in the queue of the inventory data handler.**

First, view information about pending scans and targets using the following command:

```
wgetscanstat -a -p
```

Next, verify that data remains in the inventory data handler queues by checking the queue data files in the \$DBDIR/inventory/data_handler directory on the inventory data handler. If the files are greater than 4 bytes in size, they contain data. For more information about these files, see “Queue Data Files” on page 288.

Check the inventory data handler log file for messages that indicate the inventory data handler could not connect to RIM object. You might see the following messages:

IR: No RIM objects found

Check to see whether the `inv_query` and `invdh_1` RIM objects exist. If they do not exist, you must create them. For more information, see “Creating and configuring RIM objects for the inventory data handler” on page 17, and the description of `wcrtim` in the *Tivoli Management Framework: Reference Manual*.

IR: ERROR: No valid rim objects for InvDataHandler in returned sequence

No existing RIM object is configured to work with the inventory data handler. You must configure one or more RIM objects for this purpose. For more information, see “Creating and configuring RIM objects for the inventory data handler” on page 17.

IR: ERROR failure in connecting to database

Use the `wrimtest` command to check the connection to each RIM object that is configured to work with the inventory data handler.

Next, stop and restart the inventory data handler. Stop the inventory data handler using the following command:

```
wcollect -h immediate @InvDataHandler:inv_data_handler
```

Restart the inventory data handler using the following command:

```
wcollect -s @InvDataHandler:inv_data_handler
```

- **A scan fails with the following error: “The specified segment ID (config.dmp) with version (n) cannot be removed because it is currently in use. Please try again later.”**

This error usually occurs when a managed node is shared by two Tivoli management regions (Tivoli regions). To resolve this error, follow these steps:

1. Change the default MDist 2 depot directory on the managed node using the following command:

```
wmdist -s managed_node_name rpt_dir=new_directory
```

2. Re-execute the object dispatcher using the following command:

```
odadmin reexec
```

This error can also happen if the status collector is reset by removing the `inv_scan_id.cfg` file in the \$DBDIR/status_dir directory. When a new scan is initiated, the version number of the `config.dmp` file will be 1. If there is still a `config.dmp` segment in a managed node MDist 2 depot with that version number, then this error will occur. To resolve this error, clean out the MDist2 depots. To avoid this error, do not remove the `inv_scan_id.cfg` files when removing files from the status collector.

Common Inventory Technology

This section contains information about troubleshooting Common Inventory Technology (CIT).

Common Inventory Technology traces

IBM Tivoli Common Inventory Technology (CIT) is a component technology used to collect hardware, software, file system and registry information from systems in a network. CIT provides a trace file located in the following directories, depending on the operating system:

On Windows systems

C:/Program Files/ibm/tivoli/common/CIT/logs/traceCIT.log

On UNIX systems

/usr/ibm/tivoli/common/CIT/logs/traceCIT.log

Use the **wscancfg** command to activate traces and modify default settings.

You can use the **wscancfg -s key value** command to define the following keys:

trace_level

MAX

trace_file_size

1024000

All traces are enabled. The default trace file size is 1024 Kilobytes.

Common Inventory Technology scanners

If you experience issues related to Inventory hardware and software scans, for example the scan fails or does not collect all the expected information, it is necessary to use a troubleshooting procedure specific for the Common Inventory Technology component.

Collect the configuration files that Tivoli Configuration Manager writes and passes to the Common Inventory Technology scanners as input parameters, and the command line syntax that Tivoli Configuration Manager uses when invoking the Common Inventory Technology scanners. Depending on the different customizations of the InventoryConfig profile and the platform on which the scan runs, the above mentioned commands and configuration files are:

Table 12. Commands and configuration files on Windows platforms

Windows		
Type of scan	Command	Configuration file
Hardware scan	C:\Program Files\tivoli\cit \bin\wscanhw -c C:\win_ep_name\ \inv\SCAN\config.xml -o C:\win_ep_name\inv\SCAN\ tivhscan.mif-m	config.xml
Scan for installed products using signature matching	C:\Program Files\tivoli\cit \bin\wscansw -i C:\win_ep_name\ \inv\SCAN\wscansw.xml -o C:\win_ep_name\inv\SCAN\ swscan.xml -c C:\win_ep_name\ \inv\SCAN\config.xml -e C:\win_ep_name\inv\SCAN\ warning.out	wscansw.xml, config.xml

Table 12. Commands and configuration files on Windows platforms (continued)

Windows		
Type of scan	Command	Configuration file
Scan files for basic information	C:\Program Files\tivoli\cit\bin\wscanfs -c C:\win_ep_name\inv\SCAN\config.xml -o C:\win_ep_name\inv\SCAN\tivfscan.mif -m	config.xml
Scan files for header information	C:\Program Files\tivoli\cit\bin\wscanfs -c C:\win_ep_name\inv\SCAN\config3.xml -o C:\win_ep_name\inv\SCAN\tivwscan.mif -m	config3.xml
Scan registry for product information	C:\Program Files\tivoli\cit\bin\wscanvpd -c C:\win_ep_name\inv\SCAN\config.xml -o C:\win_ep_name\inv\SCAN\tivrscan.mif -m	config.xml

Table 13. Commands and configuration files on UNIX platforms

UNIX		
Type of scan	Command	Configuration file
Hardware scan	/opt/tivoli/cit/bin/wscanhw -c /tivoli/unix_ep_name/inv/SCAN/config.xml -o /tivoli/unix_ep_name/inv/SCAN/tivhscan.mif -m	config.xml
Scan for installed products using signature matching	/opt/tivoli/cit/bin/wscansw -i /tivoli/unix_ep_name/inv/SCAN/wscansw.xml -o /tivoli/unix_ep_name/inv/SCAN/swscan.xml -c /tivoli/unix_ep_name/inv/SCAN/config.xml -e /tivoli/unix_ep_name/inv/SCAN/warning.out	wscansw.xml, config.xml
Scan files for basic information	/opt/tivoli/cit/bin/wscanfs -c /tivoli/unix_ep_name/inv/SCAN/config.xml -o /tivoli/unix_ep_name/inv/SCAN/tivfscan.mif -m	config.xml
Scan registry for product information	/opt/tivoli/cit/bin/wscanvpd -c /tivoli/unix_ep_name/inv/SCAN/config.xml -o /tivoli/unix_ep_name/inv/SCAN/tivrscan.mif -m	config.xml

The Common Inventory Technology scanners might return error codes. Return codes help you identify the result of the command: a return code of 0 indicates that the command completed successfully, while a return code other than zero indicates that an error occurred. A list of all return codes other than zero is given in Table 14 on page 298:

Table 14. Return codes

Return value	Code	Description
WSRC_WRONG_PARMS	1	One or more CLI options are incorrect.
WSRC_INPUT_FILE_PARSE_ERROR	2	An error occurred while parsing the configuration file.
WSRC_SIGNATURE_FILE_PARSE_ERROR	3	An error occurred while parsing the signature file.
WSRC_OUTPUT_FILE_ERROR	4	An error occurred while writing the output file.
WSRC_INPUT_FILE_ERROR	5	An error occurred while reading the input file.
WSRC_MISSING_SIGNATURE_FILE	6	No signature file was specified and no default signature file is available.
WSRC_VALUE_OUT_OF_BOUND	7	One of the values you specified exceeds the assigned limits.
WSRC_INTERNAL_ERROR	8	An internal error has occurred.
WSRC_TIMEOUT_ELAPSED	9	The specified timeout has expired.
WSRC_UPGRADE_IN_PROGRESS	10	CIT is being upgraded and commands are momentarily not responding.
WSRC_FILE_READ_ONLY	11	The output file is read only.
WSRC_INIFILE_NOT_FOUND	12	The cit.ini file was not found.
WSRC_CITFILE_NOT_FOUND	13	The CIT configuration file was not found.
WSRC_CCLOGFILE_NOT_FOUND	14	The CitTrace.properties file was not found.
WSRC_KEY_NOT_FOUND	15	The value you specified is incorrect.
WSRC_VALUE_NOT_VALID	16	The specified value is not valid.
WSRC_KEY_CANNOT_CHANGE	17	The specified key cannot be modified.
WSRC_FILE_CANNOT_OPEN	18	Cannot open the specified file.
WSRC_FILE_CANNOT_RENAME	19	Cannot rename the specified file.
WSRC_FILE_CANNOT_DELETE	20	Cannot delete the specified file.
WSRC_CITFILE_NOT_VALID	21	The cit.ini file is corrupt.
WSRC_CIT_TRACEFILE_NOT_VALID	22	The trace file is corrupt.
WSRC_INVALID_AGE	24	The age you specified is incorrect.
WSRC_INVALID_TIMEOUT	25	The timeout you specified is incorrect.
WSRC_INVALID_ATTRIBUTE	26	The attribute you specified is incorrect.
WSRC_INVALID_OUTPUT_FORMAT	27	The output format you specified is not supported.
WSRC_CANNOT_LOAD_PROVIDER	28	The required .dll or shared library file is not available.
WSRC_QUERY_TIMED_OUT	29	The query has reached the timeout.
WSRC_QUERY_FAILED	30	The query has failed.

Table 14. Return codes (continued)

Return value	Code	Description
WSRC_PROCESS_INTERRUPTED	31	The process was interrupted.
WSRC_NO_CONFIG_NAME	32	No configuration file was specified.
WSRC_NO_CONFIG_OPTION	33	No configuration option was specified.
WSRC_NO_OUTPUT_NAME	34	No output file was specified.
WSRC_NO_PARMS	35	No parameters were specified.
WSRC_EMPTY_CONFIG_ELEMENT	36	The configuration file contains an empty element.
WSRC_FAILURE	37	An internal error has occurred.
WSRC_NO_SORT_FIELD_NAME	38	You specified the sort option in the command without specifying a sort criterion.
WSRC_INVALID_SORT_FIELD_NAME	39	The sort criterion you specified is incorrect.
WSRC_WARNING_FILE_ERROR	40	An error has occurred while attempting to create the warning file during a software scan.
WSRC_UNABLE_TO_INITIALIZE	41	The process initialization failed.
WSRC_MISSING_XSS_SCHEMA_FILE	42	Cannot find the signature catalog schema.
WSRC_UNABLE_INSTALL_DRIVER	45	Cannot install the CITMDRV_IA64.SYS, CITMDRV_AMD64.SYS, CITMDRV.SYS drivers.
WSRC_UNABLE_LOAD_CITMEMDLL	46	Cannot load the CITMEM.DLL library.
WSRC_UNABLE_LOAD_SYMBOL_IN_CITMEM	47	Cannot load the symbols in the CITMEM.DLL library.
WSRC_UNABLE_READ_CITMEMDLL	48	Cannot read the CITMEM.DLL library.
WSRC_FILE_ACCESS_DENIED	49	The user does not have sufficient rights to access the file.
WSRC_NOT_AUTHORIZED	50	The user does not have sufficient rights to perform the operation.
WSRC_FILE_NOT_FOUND	51	The specified file or directory does not exist.

Common Inventory Technology installations

The default installation of Common Inventory Technology is performed on the endpoints using the Tivoli Configuration Manager Software Distribution disconnected command line.

Use the following troubleshooting procedure when installing Common Inventory Technology on Tivoli Configuration Manager workstations:

1. Enable the Software Distribution disconnected CLI traces by opening on the endpoint the Software Distribution swdis.ini file, which can be found under the %WINDIR% directory on Windows and the /etc/Tivoli directory on UNIX, and

set the `trace_level` value to 5 in the MOBILE section of the file. A sample MOBILE section of the `swdis.ini` file follows:

```
[#MOBILE]
product_dir=C:\swdis
working_dir=C:\swdis\work
backup_dir=C:\swdis\backup
profile_dir=C:\swdis\work\profiles
trace_level=0
trace_size=1000000
send_timeout=300
autopack_dir=C:\swdis\autopack
staging_dir=swdis\service
user_file_variables=C:\swdis\swdis.var
import_libraries=spd,libecimp
```

2. Distribute an Inventory profile by running the `wdistinv` command using the `inv_ep_debug` option, so that when the Inventory scan runs on the endpoint, a log file called `INVxxxxx.LOG` (where `xxxxx` is the `scan_id`) is created under the `inv/SCAN` directory.
3. Collect under the `inv/SCAN` directory the file `INVxxxxx.LOG`, and under the product directory specified in the MOBILE section mentioned in step 2 the trace files `*.trc`, and under the working directory the `epsp.cat` file. Collect the `cit.ini` file, which can be found under the `%WINDIR%` directory on Windows or the `/etc/Tivoli` directory on UNIX.

Queries

Problems with queries can result from the following conditions:

- The `invdh_1` RIM object fails, causing scan failures.
- The `inv_query` RIM object fails, causing problems with queries, the Inventory property GUI, or the **`winvsig`**, **`winvfilter`**, or **`winvrnode`** commands.

To test the RIM objects for scan failures, run the following command.

```
wrimtest -l RIM_object
```

where *RIM_object* is the name of the RIM object, either `inv_query` or `invdh_1`.

The following message indicates that the connection succeeded:

```
Opening Regular Session...Session Opened
```

If the connection succeeded, enter `x` to exit **`wrimtest`**.

If the connection fails, a message similar to the following is displayed:

```
Opening Regular Session...FRWRA0012E
The RDBMS server call has failed.
```

If the connection fails, verify that the RIM object passwords are set correctly. Also check the output generated by **`wrimtest`** for incorrect settings. Use the **`wrimset`** command to correct the settings for each RIM object.

If the settings are correct and the problem persists, attempt to connect to the configuration repository using the commands provided with the RDBMS.

RIM objects

When you install Inventory, a RIM object named `invdh_1` is created. The inventory data handler uses this RIM object to write scan data to the configuration repository. This RIM object has the application label `invdh`.

Every RIM object that the inventory data handler uses must have an application label of `invdh`. If you delete the `invdh_1` RIM object and recreate it using the **wcrtrim** command, or if you create additional RIM objects for the inventory data handler, you must use the **wcrtrim** command and the **-a** option to set the value of the application label to `invdh`. You must also use the **wcrtrim** command and **-m** option to configure the maximum number of connections that the RIM object is allowed to establish with the configuration repository. It is recommended that the total number of RDBMS connections set for all RIM objects used by the inventory data handler match the number of output threads set for the inventory data handler. For example, if the inventory data handler has 10 output threads and uses two RIM objects to connect to the RDBMS, you could configure each RIM object to have five RDBMS connections.

If you created a RIM object for use by the inventory data handler but did not set the application label or maximum number of RDBMS connections, use the **wsetrim** command and the **-a** and **-m** options.

For more information about these procedures, see “Creating and configuring RIM objects for the inventory data handler” on page 17. For more information about the **wcrtrim** and **wsetrim** commands, see the *Tivoli Management Framework: Reference Manual*.

Scans

“Scalable Collection Service” on page 288 provides information about troubleshooting SCS while it processes scan data. This section provides additional information about troubleshooting scans.

Cancelling scans

You can cancel active scans using the **wcancelscan** command. This command can cancel a scan at the following points:

- During the profile distribution, before the profile reaches the endpoint
- As the data travels through the collector hierarchy to inventory data handler through SCS
- At the inventory data handler
- For scans of pervasive devices, at the Web Gateway component before the job is processed

After you run the **wcancelscan** command, you might see an error message in the Inventory notice group similar to the following example:

The Inventory status collector returned a status for scan ID: 17 which is not valid

This error message occurs because of the way the inventory data handler handles cancelled scans. The inventory data handler does not keep a list of IDs for cancelled scans. Instead, if a scan is cancelled, the inventory data handler discards the ID of the cancelled scan. Then, when data for a cancelled scan reaches the

inventory data handler, it does not recognize the scan ID. The inventory data handler discards the data for the cancelled scan and sends the error message described previously.

A cancelled scan can reach the inventory data handler under the following circumstances:

- If you run **wcancelscan** after scan data has passed through the collector hierarchy and is on the way to the inventory data handler.
- If you run **wcancelscan** after a scan begins to run on an endpoint but before the scan data is collected from the endpoint.
- When you cancel scans of pervasive devices. The Web Gateway component packages the scan data for pervasive devices in a .DAT file. If you run multiple scans, the Web Gateway component might package data for multiple scans into one .DAT file. Collectors do not read the scan IDs in these .DAT files and therefore cannot determine if a cancelled scan is contained in the .DAT file. However, the inventory data handler can read the .DAT files. If the inventory data handler reads the ID for a cancelled scan in the .DAT file, it does not recognize the ID so it discards the scan data.

If you cannot cancel a scan on the Web Gateway component using the **wcancelscan** command, use the **wwebgw** command to view information about the jobs on the Web Gateway component. You can then use the **wwebgw** command to cancel the job from the Web Gateway component. For more information about the **wwebgw** command, see the *Reference Manual for Software Distribution*.

Profile distribution failures

You might see an error message similar to the following after a profile distribution:

```
Failed to access the file $DBDIR/inventory/INV00003/config.dmp
with the mode 4.
```

To recover from this error, first use the **wdepot** command to view information about the depot. Then perform one of the following tasks:

- Delete the `$rpt_dir/states` or `$rpt_dir/depot` directory.
- Use the **wmdist** command and `-s` option to change `rpt_dir` to a path unique to the object dispatcher, such as a subdirectory below `$DBDIR`.

Software scan with the option "Scan Registry for Product Information"

The scan completes, but the `NATIV_SWARE_QUERY` is empty after a software scan with the option "Scan Registry for Product Information" enabled.

To resolve this error, check that the `inv_XX_admin.sql` file has been run after the installation (where `XX` is the Database ID).

Appendix G. Accessibility

Accessibility features help users who have physical disabilities, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in this product enable users to do the following:

- Use assistive technologies, such as a screen-reader, to hear what is displayed on the screen. Consult the product documentation of the assistive technology for details on using those technologies with this product.
- Operate features using only the keyboard.
- Magnify what is displayed on the screen.

In addition, the product documentation includes the following features that enable the use of a screen-reader, thereby aiding accessibility:

- All documentation is available in both HTML and convertible PDF formats.
- All images in the documentation are provided with alternative text so that users with vision impairments can understand the contents of the images.

Navigating the interface using the keyboard

Standard shortcut and accelerator keys are used by the product and are documented by the operating system. Refer to the documentation provided by your operating system for more information.

The signature, signature package, and filter tables in the Inventory GUI provide a number of keyboard shortcuts for sorting and filtering actions, as shown in Table 15.

Table 15. Keyboard Shortcuts for Inventory

Select all displayed rows	Ctrl+A
Deselect all displayed rows	Ctrl+Shift+A
Edit the filter for the current column	Ctrl+E
Clear the sort	Ctrl+J
Clear all filters	Ctrl+K
Show context menu	Ctrl+M
Clear all sorts	Ctrl+Q
Show the filter row	Ctrl+R
Hide the filter row	Ctrl+Shift+R
Sort the current column in ascending order	Ctrl+S
Sort the current column in descending order	Ctrl+Shift+S
Unapply the filter for the current column	Ctrl+U
Apply the filter for the current column	Ctrl+Shift+U
Edit the sort	Ctrl+W

Magnifying what is displayed on the screen

You can enlarge information on the product windows using facilities provided by the operating systems on which the product is run. For example, in a Microsoft Windows environment, you can lower the resolution of the screen to enlarge the font sizes of the text on the screen. Refer to the documentation provided by your operating system for more information.

Appendix H. Support information

This section describes the following options for obtaining support for IBM products:

- “Searching knowledge bases”
- “Obtaining fixes”
- “Contacting IBM Software Support” on page 306

Searching knowledge bases

If you have a problem with your IBM software, you want it resolved quickly. Begin by searching the available knowledge bases to determine whether the resolution to your problem is already documented.

Search the information center on your local system or network

IBM provides extensive documentation that can be installed on your local computer or on an intranet server. You can use the search function of this information center to query conceptual information, instructions for completing tasks, reference information, and support documents.

Search the Internet

If you cannot find an answer to your question in the information center, search the Internet for the latest, most complete information that might help you resolve your problem. To search multiple Internet resources for your product, expand the product folder in the navigation frame to the left and select **Web search**. From this topic, you can search a variety of resources including:

- IBM technotes
- IBM downloads
- IBM Redbooks
- IBM developerWorks
- Forums and newsgroups
- Google

Obtaining fixes

A product fix might be available to resolve your problem. You can determine what fixes are available for your IBM software product by checking the product support Web site:

1. Go to the IBM Software Support Web site (<http://www.ibm.com/software/support>).
2. Under **Products A - Z**, select your product name. This opens a product-specific support site.
3. Under **Self help**, follow the link to **All Updates**, where you will find a list of fixes, fix packs, and other service updates for your product. For tips on refining your search, click **Search tips**.
4. Click the name of a fix to read the description and optionally download the fix.

To receive weekly e-mail notifications about fixes and other news about IBM products, follow these steps:

1. From the support page for any IBM product, click **My support** in the upper-right corner of the page.
2. If you have already registered, skip to the next step. If you have not registered, click **register** in the upper-right corner of the support page to establish your user ID and password.
3. Sign in to **My support**.
4. On the My support page, click **Edit profiles** in the left navigation pane, and scroll to **Select Mail Preferences**. Select a product family and check the appropriate boxes for the type of information you want.
5. Click **Submit**.
6. For e-mail notification for other products, repeat Steps 4 and 5.

For more information about types of fixes, see the *Software Support Handbook* (<http://techsupport.services.ibm.com/guides/handbook.html>).

Contacting IBM Software Support

IBM Software Support provides assistance with product defects.

Before contacting IBM Software Support, your company must have an active IBM software maintenance contract, and you must be authorized to submit problems to IBM. The type of software maintenance contract that you need depends on the type of product you have:

- For IBM distributed software products (including, but not limited to, Tivoli, Lotus, and Rational products, as well as DB2 and WebSphere products that run on Windows or UNIX operating systems), enroll in Passport Advantage in one of the following ways:
 - **Online:** Go to the Passport Advantage Web page (http://www.lotus.com/services/passport.nsf/WebDocs/Passport_Advantage_Home) and click **How to Enroll**
 - **By phone:** For the phone number to call in your country, go to the IBM Software Support Web site (<http://techsupport.services.ibm.com/guides/contacts.html>) and click the name of your geographic region.
- For IBM eServer software products (including, but not limited to, DB2 and WebSphere products that run in zSeries, pSeries, and iSeries environments), you can purchase a software maintenance agreement by working directly with an IBM sales representative or an IBM Business Partner. For more information about support for eServer software products, go to the IBM Technical Support Advantage Web page (<http://www.ibm.com/servers/eserver/techsupport.html>).

If you are not sure what type of software maintenance contract you need, call 1-800-IBMSERV (1-800-426-7378) in the United States or, from other countries, go to the contacts page of the IBM Software Support Handbook on the Web (<http://techsupport.services.ibm.com/guides/contacts.html>) and click the name of your geographic region for phone numbers of people who provide support for your location.

Follow the steps in this topic to contact IBM Software Support:

1. Determine the business impact of your problem.
2. Describe your problem and gather background information.
3. Submit your problem to IBM Software Support.

Determine the business impact of your problem

When you report a problem to IBM, you are asked to supply a severity level. Therefore, you need to understand and assess the business impact of the problem you are reporting. Use the following criteria:

Severity 1	Critical business impact: You are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.
Severity 2	Significant business impact: The program is usable but is severely limited.
Severity 3	Some business impact: The program is usable with less significant features (not critical to operations) unavailable.
Severity 4	Minimal business impact: The problem causes little impact on operations, or a reasonable circumvention to the problem has been implemented.

Describe your problem and gather background information

When explaining a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Software Support specialists can help you solve the problem efficiently. To save time, know the answers to these questions:

- What software versions were you running when the problem occurred?
- Do you have logs, traces, and messages that are related to the problem symptoms? IBM Software Support is likely to ask for this information.
- Can the problem be re-created? If so, what steps led to the failure?
- Have any changes been made to the system? (For example, hardware, operating system, networking software, and so on.)
- Are you currently using a workaround for this problem? If so, please be prepared to explain it when you report the problem.

Submit your problem to IBM Software Support

You can submit your problem in one of two ways:

- **Online:** Go to the "Submit and track problems" page on the IBM Software Support site (<http://www.ibm.com/software/support/probsub.html>). Enter your information into the appropriate problem submission tool.
- **By phone:** For the phone number to call in your country, go to the contacts page of the IBM Software Support Handbook on the Web (techsupport.services.ibm.com/guides/contacts.html) and click the name of your geographic region.

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support provides a workaround for you to implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the IBM product support Web pages daily, so that other users who experience the same problem can benefit from the same resolutions.

For more information about problem resolution, see Searching knowledge bases and Obtaining fixes.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to

IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information in softcopy form, the photographs and color illustrations might not display.

Trademarks

AIX	DB2	IBM
OS/2	OS/400	S/390
Tivoli	Tivoli Enterprise	Tivoli Enterprise Console
TME	Wake on LAN	

AIX, DB2, IBM, OS/2, OS/400, S/390, Tivoli, Tivoli Enterprise, Tivoli Enterprise Console, TME, and Wake on LAN are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Lotus, Freelance Graphics, Word Pro, and 1-2-3 are trademarks of International Business Machines Corporation and Lotus Development Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

A

admin role. *See* authorization role.

administrator. *See* Tivoli administrator.

authorization role. In a Tivoli environment, a role assigned to Tivoli administrators to enable them to perform their assigned systems management tasks. A role may be granted over the entire Tivoli Management Region or over a specific set of resources, such as those contained in a policy region. Examples of authorization roles include: super, senior, admin, and user.

B

bulletin board. The mechanism by which the Tivoli Management Framework and Tivoli applications communicate with Tivoli administrators. The bulletin board collects notices in notice groups. Administrators can access the bulletin board from the Tivoli desktop. The bulletin board is an audit trail for important operations that the administrators perform. *See also* notice and notice group.

C

collector. In a Tivoli environment, either (a) a repeater site on which Scalable Collection Service (SCS) is installed or (b) an SCS daemon on a managed node or gateway that stores and then forwards data to other collectors or to the inventory receive or inventory data handler.

configuration repository. In a Tivoli environment, a RIM repository that contains information that is collected or generated and stored by Inventory and Software Distribution.

D

default policy. In the Tivoli environment, a set of resource property values that are assigned to a resource when the resource is created.

downcall. In a Tivoli environment, a method invocation from the Tivoli server or the gateway “down” to an endpoint. Contrast with *upcall*.

downstream. In a network, pertaining to the direction to which data flows.

In a hierarchical network structure, pertaining to the location of a network entity that is lower in the hierarchy. For example, a client is downstream from a server.

Contrast with *upstream*.

E

endpoint. In a Tivoli environment, the agent that is the ultimate recipient for any type of Tivoli operation.

event console. In the Tivoli Enterprise Console product, a graphical user interface (GUI) that enables system administrators to view and respond to dispatched events from the event server.

event server. A server program that processes events.

G

gateway. Software that provides services between the endpoints and the rest of the Tivoli environment.

H

host. In a Tivoli environment, a computer that serves as a managed node for a profile distribution.

I

IDL. *See* Interface Definition Language.

Inventory. A component of IBM Tivoli Configuration Manager that enables system administrators to gather hardware and software information for a network computing environment. It scans the managed resources and stores inventory information in the configuration repository.

Interface Definition Language (IDL). In CORBA, a declarative language that is used to describe object interfaces, without regard to object implementation.

inventory data handler. In a Scalable Collection Service topology, the Inventory object that receives data from an inventory scan and uses one or more connections to send the data to the configuration repository.

J

job. In a Tivoli environment, a resource consisting of a task and its preconfigured execution parameters.

Among other things, the execution parameters specify the set of hosts on which the job is to execute.

M

managed node. In a Tivoli environment, a computer system on which the Tivoli Management Framework is installed. Contrast with *endpoint*.

managed resource. In a Tivoli environment, any hardware or software entity (machine, service, system, or facility) that is represented by a database object and an icon on the Tivoli desktop. Managed resources must be a supported resource type in a policy region and are subject to a set of rules. Managed resources include, but are not limited to, managed nodes, task libraries, monitors, profiles, and bulletin boards.

Management Information Format (MIF). The Desktop Management Interface (DMI) specification that defines the syntax for describing management information about the hardware and software components that can be installed on a computer system.

MDist 2. A multiplexed distribution service provided by Tivoli Management Framework that enables efficient transfer of data to multiple targets. Administrators can monitor and control a distribution throughout its life cycle. Another multiplexed distribution service, MDist, lacks these management features.

MIF. See Management Information Format.

N

name registry. See Tivoli name registry.

notice. In a Tivoli environment, a message generated by a systems management operation that contains information about an event or the status of an application. Notices are stored in notice groups. See also bulletin board.

notice group. In a Tivoli environment, an application- or operation-specific container that stores and displays notices that pertain to specific Tivoli functions. The Tivoli bulletin board is comprised of notice groups. A Tivoli administrator can subscribe to one or more notice groups. The administrator's bulletin board contains only the notices that reside in a notice group to which the administrator is subscribed. See also bulletin board and notice.

O

object dispatcher. The name of the object request broker used by the Tivoli environment. The object dispatcher runs on the Tivoli Management Region server and each Tivoli Management Region client.

object reference. In a Tivoli environment, the object identifier (OID) that is given to an object during its creation.

OID. Object identifier.

P

PDA. See personal data assistant.

personal data assistant (PDA). A handheld device that is used for personal organization tasks (such as calendaring, note-taking, and recording telephone and fax numbers), and networking functions such as e-mail and synchronization.

policy. In the Tivoli environment, a set of rules that are applied to managed resources.

policy region. A group of managed resources that share one or more common policies and which model the management or organizational structure of a network computing environment. Administrators use policy regions to group similar resources, to define access to the resources, to control the resources, and to associate rules for governing the resources.

policy subregion. In a Tivoli environment, a policy region created or residing in another policy region. When a policy subregion is created, it initially uses the resource and policy properties of the parent policy region. The Tivoli administrator can later change or customize these properties to reflect the specific needs and differences of the subregion.

profile. In a Tivoli environment, a container for application-specific information about a particular type of resource. A Tivoli application specifies the template for its profiles, which includes information about the resources that can be managed by that Tivoli application.

profile manager. In a Tivoli environment, a container for profiles that links the profiles to a set of resources, called subscribers. Tivoli administrators use profile managers to organize and distribute profiles. A profile manager can operate in the dataless mode or database mode.

prototype profile. In a Tivoli environment, a model profile from which a Tivoli administrator can create other like profiles, often by cloning the existing profile.

Q

query. In a Tivoli environment, a combination of statements that are used to search the configuration repository for systems that meet certain criteria. See also query library.

query library. In a Tivoli environment, a facility that provides a way to create and manage Tivoli queries. See also query.

R

RDBMS interface module (RIM). In the Tivoli Management Framework, the module in the distributed object database that contains information about the installation of the relational database management system (RDBMS).

registered name. In a Tivoli environment, the name by which a particular resource is registered with the name registry when it is created.

repeater site. In a Tivoli management region, a managed node that is configured with the MDist or MDist 2 feature. A repeater site receives a single copy of data and distributes it to the next tier of clients.

resource. A hardware, software, or data entity that is managed by Tivoli management software.

resource type. In a Tivoli environment, one of the properties of a managed resource. Resource types are defined in the default policy for a policy region.

RIM. See RDBMS interface module.

RIM host. In the Tivoli environment, the managed node on which one or more RIM objects is installed. See also RIM object.

RIM object. An object that provides the attributes and methods that enable applications to access an RDBMS. See also RIM host.

role. A job function that identifies the tasks that a user can perform and the resources to which a user has access. A user can be assigned one or more roles.

S

Scalable Collection Service (SCS). In a Tivoli environment, a distributed service that enables efficient, asynchronous collection of large amounts of data across complex networks. Formerly called MCollect.

scan. To perform an operation that gathers information from a computer system.

schema. The set of statements, expressed in a data definition language, that completely describe the structure of a database.

SCS. See Scalable Collection Service.

senior role. See authorization role.

signature. The set of unique information that identifies a software application, such as the name, version, and file size of an application.

signature package. A logical grouping of two or more signatures.

Software Distribution. A component of IBM Tivoli Configuration Manager that automates software distribution to clients and servers in a network-computing environment. An organization can use this component to install and update applications and software in a coordinated, consistent manner across a network. Software Distribution creates software packages (in Version 4) and file packages (in Version 3) and distributes them to predefined subscribers.

subscriber. In a Tivoli environment, a resource that is subscribed to a profile manager.

subscription. In a Tivoli environment, the process of identifying the subscribers to which profiles will be distributed.

subscription list. In a Tivoli environment, a list that identifies the subscribers to a profile manager. A profile manager can be included in a subscription list in order to subscribe several resources simultaneously rather than adding each resource individually. In Tivoli Plus modules, a profile manager functions as a subscription list.

super role. See authorization role.

T

target. In the Tivoli environment, a Tivoli client on which a job or other activity is performed.

task. In a Tivoli environment, the definition of an action that must be routinely performed on various managed resources throughout the network. A task defines the executables to be run when the task is executed, the authorization role required to execute the task, and the user or group name under which the task will execute.

Tivoli administrator. In a Tivoli environment, a system administrator who has been authorized to perform systems management tasks and manage policy regions in one or more networks.

Tivoli client . A client of a Tivoli server. See Tivoli management region client and Tivoli management region server.

Tivoli desktop. In the Tivoli environment, the desktop that system administrators use to manage their network computing environment.

Tivoli Enterprise Console. A Tivoli product that collects, processes, and automatically initiates corrective actions for system, application, network, and database

events; it is the central control point for events from all sources. The Tivoli Enterprise Console product provides a centralized, global view of the network computing environment; it uses distributed event monitors to collect information, a central event server to process information, and distributed event consoles to present information to system administrators.

Tivoli environment. The Tivoli applications, based upon the Tivoli Management Framework, that are installed at a specific customer location and that address network computing management issues across many platforms.

Tivoli management region (Tivoli region). In a Tivoli environment, a Tivoli server and the set of clients that it serves. An organization can have more than one region. A Tivoli management region addresses the physical connectivity of resources whereas a policy region addresses the logical organization of resources.

Tivoli management region client. In a Tivoli environment, any computer -- except the Tivoli management region server -- on which the Tivoli Management Framework is installed. The object dispatcher daemon runs on the Tivoli management region client, and the region client maintains a local object database. See Tivoli client and Tivoli management region server.

Tivoli management region server (Tivoli server). The server for a specific Tivoli management region that holds or references the complete set of Tivoli software, including the full object database.

Tivoli name registry. In a Tivoli environment, the table that maps names of managed resources to resource identifiers (and the corresponding information) within a Tivoli management region.

Tivoli region. See Tivoli management region.

Tivoli server. See Tivoli management region server.

U

upcall. In a Tivoli environment, a method invocation from an endpoint “up” to the gateway. Contrast with *downcall*.

upstream. In a network, pertaining to the direction from which data flows.

In a hierarchical network structure, pertaining to the location of a network entity that is higher in the hierarchy. For example, a server is upstream from a client.

Contrast with *downstream*.

user role. See authorization role.

V

validation policy. In a Tivoli environment, the policy that ensures that all resources in a policy region comply with the region's established policy. Validation policy prevents Tivoli administrators from creating or modifying resources that do not conform to the policy of the policy region in which the resources were created. Contrast with default policy.

Index

Special characters

- .exe files, scanning for 39, 199, 214
- \$DBDIR directory 288
- \$DBDIR/inventory/data_handler
 - directory 112, 288
- \$DBDIR/inventory/stat_dir
 - directory 117
- \$DBDIR/mcollect directory
 - configuring run-time directory 14, 112
 - CTOC log files 290
 - log and data files 288
 - log file location 112
- \$DBDIR/mcollect/depot directory 292
- \$LCFROOT/inv/ISOLATED
 - directory 159
- \$LCFROOT/inv/ISOLATED/common
 - directory 159
- \$LCFROOT/inv/ISOLATED/common/
 - generic/codeset directory 159
- \$LCFROOT/inv/ISOLATED/depot
 - directory 159
- \$TMPDIR directory 290

A

- absolute object path 104
- accessibility 303
- admin role 101
- application label for RIM objects 18, 301
- assigning policy 224
- authorization roles
 - admin role 101
 - backup role 101
 - Install_client role 101
 - Install_product role 101
 - Inventory_edit role 101
 - Inventory_end_user role 101
 - Inventory_query role 101
 - Inventory_scan role 101
 - Inventory_view role 101
- policy
 - assigning 224
 - creating 222
 - replacing 223
- profiles
 - cloning 47
 - creating 28
 - deleting 48
 - distributing 52
- queries
 - creating custom 88
 - defining subscribers with 94
 - list of roles 102
- Query_edit role 101
- Query_execute role 101
- Query_view role 101
- required for Inventory 101
- restore role 101
- RIM_update role 101

- authorization roles (*continued*)
 - RIM_view role 101
 - running queries 92
 - senior role 101
 - super role 101
 - user role 101
 - viewing inventory information 96
- AutoTrace 281

B

- backup role 101
- basic information
 - filtering using the CLI 199, 214
 - filtering using the GUI 38
 - modifying the filter list from the CLI 157
 - scanning 38, 199, 214
- BasicInventoryConfig policy object 222
- books
 - See publications
- bundle
 - configuring destination 24, 188
 - configuring interval at which it is sent 25, 117, 184
 - configuring maximum number of targets 25, 118, 184
 - enabling bundling 131, 184
 - viewing information 131

C

- checkpoint mode 111
- checkpointGL_dqfile.dat file 288
- checkpointGL_eqfile.dat file 288
- checkpointGL_igfile.dat file 288
- checkpointGL_oqfile.dat file 288
- checksums, generating 39, 199, 214
- CIT
 - troubleshooting 295
- cloning profiles 47
- codeset.zip file 159
- collection
 - deferred 20, 21, 113
 - error conditions 289
 - failed 293
 - scheduling 20, 21, 113
 - using SCS 4
- collection manager
 - description 2
 - log file 289
- collector
 - configuring 14, 111
 - configuring retries 15, 112
 - configuring the log file 111, 112, 289
 - configuring the run-time directory 14, 112
 - definition 2
 - increasing the size of the depot 14, 113, 293

- collector (*continued*)
 - input threads 15, 22, 113
 - moving the depot 14
 - output threads 15, 112
 - resetting collection routes 16, 113
 - restarting to enable changes 114
 - scheduling collections 20, 21, 113
 - starting 14, 111
 - stopping 14, 111
 - thread idle down time 112
 - thread sleep time 113
 - troubleshooting 293
 - using 13
 - viewing configuration information 23, 110
 - viewing status information 23, 120
- collector hierarchy 2, 16
- collector output queue
 - monitor 114
- columns, naming 67
- commands
 - dmiscan 283
 - idcall 19
 - odadmin 113, 287, 295
 - overview 103
 - syntax 103
 - tail 289, 290, 291
 - wadminep 282, 292
 - wcancelscan 108
 - wcodeset 56
 - wcollect 13, 110
 - wcrtadmin 74
 - wcrtinvcb 116
 - wcrtinvdh 117
 - wcrtpol 222
 - wcrtprf 48
 - wcrtquery 70
 - wcrtrim 18
 - wcstat 120, 288
 - wdel 49
 - wdistinv 122
 - wepscan 55, 56, 127, 285
 - wgetinvdh 131, 293
 - wgetinvglobal 133
 - wgetinvpfiles 136
 - wgetinvpchw 138
 - wgetinvpcsw 140
 - wgetinvpvdconfig 143
 - wgetinvpvdhw 144
 - wgetinvpvdsw 145
 - wgetinvswd 146
 - wgetinvunixfiles 147
 - wgetinvunixhw 149
 - wgetinvunixsw 151
 - wgetpolm 223
 - wgetrim 293
 - wgetscanstat 154
 - winvdeps 156, 277
 - winvfilter 157
 - winviso 56, 159
 - winvmgr 161

- commands (*continued*)
 - winvmigrate 166
 - winvpkgage 167
 - winvrnode 170
 - winvsig 172
 - winvupdatesid 175
 - wloadiso 56, 178
 - wlookup 19
 - wlsinst 293
 - wlspolm 223
 - wmvinvcb 180
 - wmvinvdh 181
 - wputpolm 223
 - wqueryinv 182
 - wrimset 300
 - wrimtest 292, 300
 - wrimtrace 291
 - wrpt 16, 113, 293
 - wruninvquery 96
 - wrunquery 94, 96
 - wsetadmin 74
 - wsetinvdh 184
 - wsetinvglobal 187
 - wsetinvpcfiles 192
 - wsetinvpchw 196
 - wsetinvpcsw 199
 - wsetinvpvdconfig 203
 - wsetinvpvdhw 204
 - wsetinvpvdsw 205
 - wsetinvswd 206
 - wsetinvunixfiles 207
 - wsetinvunixhw 211
 - wsetinvunixsw 214
 - wsetpr 224
 - wsetrim 19
 - wtransfer 217
 - wwaitscan 219
 - wwebgw 302
- common inventory technology
 - enabling traces 296
- components of Inventory 1
- computer system ID
 - finding 170
 - resolving duplicates 175
 - updating 175
- configuration files
 - distributing 33, 187
 - policy 229
 - writing contents to a log file 128
- configuration repository
 - adding tables 69
 - adding views 85
 - configuring scans to send data
 - from the CLI 199, 214
 - from the GUI 35, 36, 43
 - customizing 67
 - definition 1
 - deleting data for one system 170
 - deleting data from custom tables 69
 - editing 67
 - modifying filter list 38, 157
 - modifying signature packages 64, 167
 - modifying signatures 61, 172
 - populating 58
 - resolving duplicate endpoint records 175

- configuration repository (*continued*)
 - specifying whether scan data is
 - updated or replaced 33, 134, 189
 - troubleshooting 281
 - viewing inventory data 24
- conventions
 - typeface xi
- conventions used xi
- copying profiles 47
- creating
 - custom views 85
 - history tables for custom tables 68
 - inventory callback object 116
 - inventory data handler 117
 - profiles 28
 - queries 88
 - RIM objects 17
 - signature packages 64, 167
 - signatures 61, 172
 - User Data Form 75
 - useradd.mif file 75
- CTOC
 - description 2
 - role in data collection 6
 - viewing information about 23, 120
- CTOC_log.dat file 290
- custom information, viewing 85
- custom MIF files
 - collecting using the CLI 192, 207
 - collecting using the GUI 44
 - creating 66
 - example 66
 - guidelines 66
 - naming 66, 68
 - policy 239
 - using 66
- custom scripts 44, 192, 207
- customer support
 - see Software Support 306
- customizing
 - configuration repository 67
 - device scans 203
 - hardware scans from the CLI
 - devices 204
 - PC systems 196
 - UNIX and OS/400 systems 211
 - hardware scans from the GUI 41
 - profiles 32
 - queries 88
 - software scans from the CLI
 - devices 205
 - PC systems 199
 - UNIX and OS/400 systems 214
 - software scans from the GUI 34

D

- data options, setting 33, 189
- data units 15, 22, 111
- DEBUG3 log file 286
- defaults for new profiles 221
- deleting
 - profiles 48
 - scan data for one system 170
 - signature packages 64, 167
 - signatures 61, 172

- depot
 - checking contents 292
 - definition 2
 - increasing the size 14, 113
 - moving 14
 - troubleshooting 293
- directories
 - SDBDIR 288
 - SDBDIR/inventory/
 - data_handler 112, 281, 288
 - SDBDIR/inventory/stat_dir 117
 - SDBDIR/mcollect 14, 112, 288, 290
 - SDBDIR/mcollect/depot 292
 - SLCFROOT/inv/ISOLATED 159
 - SLCFROOT/inv/ISOLATED/
 - common 159
 - SLCFROOT/inv/ISOLATED/
 - common/generic/codeset 159
 - SLCFROOT/inv/ISOLATED/
 - depot 159
 - STMPDIR 290
 - rules for specifying in scans 192, 207
 - scanning 39, 192, 207
- directory names, notation xii
- disability 303
- distributing profiles
 - from the CLI 122
 - from the GUI 52
 - overview 51
- distribution
 - setting deadline 54, 122, 161
 - setting options 33, 187
 - setting timeout value 54, 122, 125, 161, 164
 - troubleshooting failures 302
- DMI
 - customizing scans 42
 - dmi_RDBMS_type _schema.sql
 - script 283
 - error messages 282
 - naming tables and columns 283
 - scanner 42
 - troubleshooting 281
- dmi_RDBMS_type _schema.sql 283
- dmiscan command 283

E

- editing
 - configuration repository 67
 - signature packages 64, 167
 - signatures 61, 172
 - User Data Form 75
 - useradd.mif file 75
- education
 - see Tivoli technical training xi
- endpoint-initiated scans
 - CLI information 127
 - description 55
 - prerequisite tasks 285
 - setting
 - environment 127
 - troubleshooting 285
 - writing scan data to a log file 128, 178
- endpoints
 - definition 3

endpoints (*continued*)
 deleting scan data 170
 querying the configuration repository
 for information 182
 resolving duplicate records 175
environment variables, notation xii
error conditions
 for collector activity 289
 when adding a DMI layer 282
EXE files, scanning for 39, 199, 214

F

features of Inventory 7
file extensions, policy 242, 258
file names, rules for specifying 193, 208
files
 configuring file scans 39, 192, 207
 Qchain.exe 40
 scanning header information 38, 199
 wsusscan.cab 40
 wsusscn2.cab 40
filter
 modifying files in the custom filter
 list 157
 scans for .exe files only 39, 199, 214
 scans for basic information 38, 199,
 214
fixes, obtaining 305
foreign keys 68

G

gateway output queue
 monitor 114
gateway, definition 3
global properties
 configuring 33, 187
 viewing 133
GUI
 enabling UNIX connections 287
 log files 286
 overview 32
 system requirements 286
 troubleshooting 286

H

hardware scans
 of devices 46, 204
 of OS/400 systems 42, 211
 of PC systems 41, 196
 of UNIX systems 42, 211
header information, scanning 38, 199
hierarchy
 collector 2, 16
 repeater 2, 16
history tables
 creating for custom tables 68
 deleting data from 70, 170

I

idllcall command 19
information centers, searching to find
 software problem resolution 305
Install_client role 101
Install_product role 101
interconnected Tivoli regions 98
Internet, searching to find software
 problem resolution 305
INV_ISO.DAT 178
INV_SA.DAT 129
INV_SA.LOG 128
inv_scan_nn.log 290
invdh_1 17
Inventory
 commands 105
 components 1
 enabling integration with Software
 Distribution 206
 features 7
 how it works 3
 how it works with SCS 4
 notice group 51
 security 9
 tasks performed 1
 viewing settings of integration with
 Software Distribution 146
inventory callback object
 creating 116
 definition 3
 moving 17, 180
inventory data handler
 configuring
 notice for unsolicited scans 25,
 185
 retries 17, 118, 185
 status information 24, 117, 184
 timeout between retries 17, 118,
 185
 where status information is
 stored 24, 184
 whether status information is
 stored 24, 185
 creating 117
 definition 2
 moving 17, 181
 output threads 16, 117, 185
 using 16
 viewing configuration
 information 24, 131
Inventory_edit role 101
Inventory_end_user role 101
Inventory_query role 101
Inventory_scan role 101
Inventory_view role 101
InventoryConfig resources, setting
 defaults 221
InvGuiDebug.log file 287
isolated scans
 copying scan files to endpoint 159
 description 56
 procedure 56
 uploading scan data 178
 using wepscan 127

K

keyboard 303
knowledge bases, searching to find
 software problem resolution 305

L

lcfcd.log file 291
log files
 collection manager 289
 collector 15
 collector queue data 288
 configuration file 128
 CTOC completed status 290
 CTOC_log.dat 290
 CTOCs 112
 DEBUG3 286
 endpoint-initiated scans 178
 endpoints 123, 128, 163, 283
 inventory data handler 289
 Inventory GUI 286
 inventory status 289
 InvGuiDebug.log 287
 lcfcd.log 291
 level of debugging for collector 111
 maximum size of collector log
 file 112
 mcollect.log 111, 289
 naming status log file 24, 187
 pervasive devices 124, 163
 RIM objects 291
 rim_db_log 291
 sa_config.log 128, 285
 sa_results.log 128, 285
 scan data 128
 schema creation errors 281
 sending to the inventory data
 handler 129
 sh.out 286
 Tivoli Management Framework
 endpoint messages 181
 WLOADISO.LOG 178

M

managed resources, setting defaults 221
management by subscription 3
manuals
 See publications
mcollect.log 111, 289
MDist 2 2, 52
MDist 2 distribution options
 changing, creating, and deleting 161
MIF files
 custom
 collecting using the CLI 192, 207
 collecting using the GUI 44
 creating 66
 example 66
 guidelines 66
 naming 66, 68
 policy 239
 using 66
 definition 3
 generated by Inventory 66

- mobile endpoints
 - definition 8
 - forcing profile distribution 123, 162
 - hiding distributions 123, 162, 188
 - mandatory distributions 124, 164
 - reminder messages 122, 162
 - support 8, 125, 164

N

- name registry 104
- naming
 - custom MIF files 66, 68
 - DB2 tables and columns 67
 - Informix tables and columns 67
 - profiles 28
 - tables and columns 67
- nobody account 112
- notation
 - environment variables xii
 - path names xii
 - typeface xii
- notice group 51
- notice, configuring for unsolicited scans 25, 185

O

- object dispatcher number
 - queries in interconnected regions 99
 - specifying offlinks 21, 113
- object ID
 - definition 99
 - finding 19, 170
- object number 99
- object paths 104
- object references 104
- odadmin command 113, 287, 295
- offlinks
 - controlling data flow 21
 - definition 20
 - specifying 113
- online publications
 - accessing x
- ordering publications x
- OS/400 systems
 - hardware scans 42, 211
 - software scans 36, 214
- output queue
 - monitor 114
 - threshold 114

P

- patch scan 35
- path names, notation xii
- paths, object 104
- PC systems
 - collecting custom MIF files 44, 192
 - hardware scan options
 - CLI 138, 196
 - GUI 41
 - running custom scripts during scans 44, 192
 - setting files and directories to be scanned 44, 192

- PC systems (*continued*)
 - software scan options
 - CLI 140, 192, 199
 - GUI 34
 - pervasive callback object, definition 3
 - pervasive device management, description 8
 - pervasive devices
 - data options 46
 - deleting scan data 170
 - device configuration scans 46, 203
 - hardware scans 46, 204
 - software scans 46, 205
 - viewing configuration data scan options 143
 - viewing hardware scan options 144
 - viewing software scan options 145
 - policy
 - assigning to a policy region 224
 - creating 222
 - default 221
 - ic_def_global_action 229
 - ic_def_global_ep_timeout 230
 - ic_def_global_logfile_host 231
 - ic_def_global_logfile_path 232
 - ic_def_global_notice_interval 233
 - ic_def_global_notice_location 234
 - ic_def_global_notice_type 235
 - ic_def_global_security 236
 - ic_def_global_update_replace 237
 - ic_def_pc_custom_before_script 238
 - ic_def_pc_custom_mifs 239
 - ic_def_pc_custom_script 240
 - ic_def_pc_exclude_dirs 241
 - ic_def_pc_extensions 242
 - ic_def_pc_hw_gran 243
 - ic_def_pc_hw_outfile 244
 - ic_def_pc_hw_scan 245
 - ic_def_pc_include_dirs 246
 - ic_def_pc_sw_crc 247
 - ic_def_pc_sw_flags 248
 - ic_def_pc_sw_outfile 249
 - ic_def_pc_sw_scan 250
 - ic_def_pvd_config_scan 251
 - ic_def_pvd_hw_scan 252
 - ic_def_pvd_sw_scan 253
 - ic_def_unix_custom_before_script 254
 - ic_def_unix_custom_mifs 255
 - ic_def_unix_custom_script 256
 - ic_def_unix_exclude_dirs 257
 - ic_def_unix_extensions 258
 - ic_def_unix_hw_gran 259
 - ic_def_unix_hw_outfile 260
 - ic_def_unix_hw_scan 261
 - ic_def_unix_include_dirs 262
 - ic_def_unix_sw_crc 263
 - ic_def_unix_sw_flags 264
 - ic_def_unix_sw_outfile 265
 - ic_def_unix_sw_scan 266
 - overview 221
 - replacing 223
 - setting a default method 224
 - populating the configuration repository 58
 - primary keys 66, 68
 - priority, setting for scans 52, 124, 164

- problem determination
 - describing problem for IBM Software Support 307
 - determining business impact for IBM Software Support 307
 - submitting problem to IBM Software Support 307
- profile managers
 - database 28
 - dataless 28
 - definition 3
 - types of subscribers allowed 28
 - using 28
- profiles
 - cancelling profile distributions 58, 108, 301
 - cloning 47
 - collecting custom MIF files 44, 192, 207
 - copying 47
 - creating 28
 - customizing 7
 - overview 32
 - using the CLI 46
 - using the GUI 33
 - definition 3
 - deleting 48
 - distributing 52, 122
 - mobile endpoints
 - forcing distributions 123, 162
 - hiding distributions 123, 162, 188
 - mandatory distributions 124, 164
 - reminder messages 122, 162
 - support 125, 164
 - naming 28
 - organizing 28
 - overview of distribution 51
 - recovering 49
 - renaming 49
 - sending data to the configuration repository
 - hardware 43, 196, 211
 - software 35, 36, 199, 214
 - setting
 - custom scripts to run 44, 192, 207
 - data options 33, 189
 - defaults for new 221
 - directories to scan 39, 192, 207
 - distribution deadline 54, 122, 161
 - distribution options 33, 187
 - distribution timeout value 122, 161
 - files to scan 39, 192, 207
 - global properties 33, 187
 - notification interval 54, 124, 164
 - priority 52, 124, 164
 - status options 24, 187
 - timeout for profile
 - distribution 54, 125, 164
 - timeout for scan attempt 187
 - timeout for target response 55, 123, 162
 - targets allowed 28
 - troubleshooting 293
 - troubleshooting distribution failures 302
 - using 27

- profiles (*continued*)
 - viewing
 - custom MIF files to be collected 136, 147
 - device configuration scan options 143
 - device hardware scan options 144
 - device software scan options 145
 - files and directories to be scanned 136, 147
 - global properties 24, 133
 - hardware scan options 138, 149
 - information about 24, 133
 - scripts to be run during scans 136, 147
 - software scan options 140, 151
 - Wake on LAN support 54, 125, 165, 189
- publications
 - accessing online x
 - ordering x
 - prerequisite ix
 - related ix
- publications, included with IBM Tivoli Configuration Manager viii

Q

- Qchain.exe
 - description 40
- queries
 - authorization roles 102
 - creating 88
 - creating libraries 89
 - customizing 90
 - defining subscribers 95
 - defining targets 94
 - example 92
 - in interconnected Tivoli regions 98
 - INVENTORY_HWARE 182
 - overview 87
 - running 92
 - running from the CLI 182
 - running from the icon menu 97
 - SIG_PACKAGE_QUERY 65, 167
 - troubleshooting 300
- query libraries, creating 89
- Query_edit role 101
- Query_execute role 101
- Query_view role 101
- QUERY_VIEWS table 85
- queue
 - completed 2
 - deferred 2
 - description 2
 - error 2
 - input 2
 - log files 288
 - output 2
 - viewing information about 23, 120

R

- region number 99
- registered name, definition 104
- relative object path 104

- renaming profiles 49
- repeater hierarchy 2, 16
- repeater site 2
- resetting a collector 114
- resetting collection routes 16, 113
- resource groups
 - definition 3
- resource type, InventoryConfig
 - InventoryConfig
 - adding to a policy region 29
- restore role 101
- retries
 - configuring for the collector 112
 - configuring for the inventory data handler 17, 118, 185
 - definition 15
- RIM hosts
 - definition 2
- RIM objects
 - configuring 17
 - creating 17
 - definition 2
 - invdh_1 17
 - moving 19
 - specifying application label 18, 301
 - specifying maximum database connections 18, 301
 - troubleshooting 300
- rim_db_log file 291
- RIM_update role 101
- RIM_view role 101
- run-time directory
 - configuring 14, 112
 - default location 14, 112

S

- sa_config.log file 128, 285
- sa_results.log file 128, 285
- scalable collection 4
- scanner
 - DMI 42, 283
 - Tivoli hardware 41
- scanning
 - from the CLI 122
 - from the GUI 52
- scans
 - cancelling 58, 108, 301
 - collecting custom MIF files 44, 192, 207
 - determining when they complete 219
 - endpoint-initiated 55, 127
 - for .exe files only 39, 199, 214
 - for basic information 38, 199, 214
 - generating checksums 39, 199
 - header 38, 199
 - initial scans 51
 - isolated
 - copying scan files to endpoint 159
 - description 56
 - procedure 56
 - uploading scan data 178
 - using wepscan 127
 - of files 35, 36, 199, 214
 - of mobile endpoints
 - forcing distributions 123, 162

- scans (*continued*)
 - of mobile endpoints (*continued*)
 - hiding distributions 123, 162, 188
 - mandatory distributions 124, 164
 - reminder messages 122, 162
 - support for 125, 164
 - of patches 35
 - of systems outside Tivoli region 56
 - of Windows 35
 - of Windows registry 35, 199
 - OS/400 hardware 42, 211
 - OS/400 software 36, 214
 - PC hardware 41, 196
 - PC software 34, 199
 - pervasive device
 - configuration 46, 203
 - hardware 46, 204
 - software 46, 205
 - replace with current results 34, 134, 189
 - sending data to the configuration repository
 - hardware 43, 196, 211
 - software 35, 36, 199, 214
 - setting
 - data options 33, 189
 - distribution deadline 54, 122, 161
 - distribution options 33, 187
 - distribution timeout value 122, 161
 - notification interval 54, 124, 164
 - priority 52, 124, 164
 - timeout for profile
 - distribution 54, 125, 164
 - timeout for scan attempt 187
 - timeout for target response 55, 123, 162
 - signature 38, 199, 214
 - specifying custom scripts 44, 192, 207
 - specifying directories 39, 192, 207
 - specifying files 39, 192, 207
 - targets allowed 28
 - troubleshooting 292, 301
 - UNIX hardware 42, 211
 - UNIX software 36, 214
 - update with differences 33, 134, 189
 - uploading isolated scan data 178
 - viewing status 24, 154
 - Wake on LAN support 54, 125, 165, 189
- scheduler daemon for collectors 2
- scheduling collections 20, 113
- scripts
 - custom 44, 192, 207
 - dmi_RDBMS_type_schema.sql 283
- SCS
 - controlling data flow 21
 - features 9
 - how it works 4
 - obtaining information about 22
 - troubleshooting 288
- senior role 101
- sh.out log file 286
- shortcut keys 303
- SIG_PACKAGE_QUERY 65, 167
- signature packages 64, 167

- signatures
 - creating 61, 172
 - definition 59
 - deleting 61, 172
 - downloading updated 61
 - editing 61, 172
 - installing 60
 - logically grouping 64, 167
 - scanning with 38, 199, 214
 - size of data units 22, 111
 - SMBIOS scanner, description 8
 - Software Distribution
 - enabling integration with Inventory 206
 - viewing integration settings 146
 - software scans
 - of devices 46, 205
 - of OS/400 systems 36, 214
 - of PC systems 34, 199
 - of UNIX systems 36, 214
 - Software Support
 - contacting 306
 - describing problem for IBM Software Support 307
 - determining business impact for IBM Software Support 307
 - submitting problem to IBM Software Support 307
 - starting a collector 14, 111
 - status collector
 - configuring options 24, 117, 184
 - definition 2
 - troubleshooting 294
 - viewing options 25, 133
 - status information
 - configuring 24, 117, 184
 - definition 24
 - specifying circumstances under which it is sent 25, 189
 - specifying host 24, 187
 - specifying log file 24, 187
 - specifying when it is sent 25, 188
 - specifying where it is sent 188
 - viewing collector status 23, 120
 - viewing scan status 131
 - stopping a collector 14, 111
 - subscribers
 - defining with queries 94
 - definition 3
 - types allowed 28
 - super role 101
 - syntax, for commands 103
- T**
- tables
 - adding to the configuration repository 69
 - deleting data from custom 69
 - designing custom 67
 - for DB2 and Informix RDBMSs 67
 - naming 67
 - tail command 289, 290, 291
 - tasks performed by Inventory 1
 - text conventions xi
- U**
- UNIX systems
 - collecting custom MIF files 44, 207
 - hardware scan options
 - CLI 149, 211
 - GUI 42
 - running custom scripts during scans 44, 207
 - setting files and directories to be scanned 44, 207
 - software scan options
 - CLI 151, 207, 214
 - GUI 36
 - threads
 - for collectors
 - idle down time 112
 - input 15, 22, 113
 - output 15, 112
 - sleep time 113
 - for the inventory data handler 16, 117, 185
 - timeout
 - for profile distribution 54, 125, 164
 - for scan attempt 187
 - for target response 55, 123, 162
 - Tivoli Enterprise Console, sending status information 24, 133, 188
 - Tivoli hardware scanner 41
 - Tivoli Management Framework, description 1
 - Tivoli region
 - inventory data handlers per 117
 - resources of the same name and type within 104
 - Tivoli scheduler 20, 21
 - Tivoli software information center x
 - Tivoli technical training xi
 - tmrsrvd account 112
 - training, Tivoli technical xi
 - transmission chunk 15, 22, 111
 - troubleshooting
 - AutoTrace 281
 - CIT 295
 - collectors 293
 - configuration repository 281
 - depots 293
 - DMI 281
 - endpoint-initiated scans 285
 - failed collections 293
 - GUI 286
 - logging collector debug information 111
 - logging endpoint debug information 283
 - logging in to GUI 287
 - profile distribution failures 302
 - profiles 293
 - queries 300
 - RIM objects 300
 - scans 292, 301
 - SCS 288
 - status collector 294
 - typeface conventions xi
 - typefaces, use of xi
- V**
- variables, notation for xii
 - viewing inventory information
 - about one client 96
 - from the CLI 182
 - from the icon menu 97
 - views 85
- W**
- wadminep 282, 292
 - Wake on LAN
 - description 8
 - enabling 54, 125, 165, 189
 - wcancelscan 108
 - wcodeset 56
 - wcollect 13, 110
 - wcrtadmin 74
 - wcrtinvcb 116
 - wcrtinvdh 117
 - wcrtpol 222
 - wcrtprf 31, 48
 - wcrtquery 70
 - wcrtrim 18
 - wcstat 120, 288
 - wdel 49
 - wdistinv 122
 - Web Gateway component 2
 - wepscan 55, 56, 127, 285
 - wgetinvdh 131, 293
 - wgetinvglobal 133
 - wgetinvpcfiles 136
 - wgetinvpchw 138
 - wgetinvpcsw 140
 - wgetinvpvdconfig 143
 - wgetinvpvdhw 144
 - wgetinvpvdsw 145
- User Data Form**
- completing 83
 - creating 75
 - editing 75
- User Data Template**
- adding MIF attributes 76
 - adding MIF groups 76
 - editing 75
- user information**
- gathering 71
 - planning to collect 72
 - retrieving and saving 84
- user role 101**
- useradd.mif file**
- creating 71, 75
 - editing 75
 - viewing groups and attributes 81
- UserLink**
- collecting custom information with 70
 - editing the User Data Template 75
 - enabling access 72
 - overview 71
 - planning to use 72
 - retrieving and saving user information 84
- userlink.htm file 83**

- wgetinvswd 146
- wgetinvunixfiles 147
- wgetinvunixhw 149
- wgetinvunixsw 151
- wgetpolm 223
- wgetrim 293
- wgetscanstat 154
- winvdeps 156
- winvdeps command 277
- winvfilter 157
- winviso 56, 159
- winvmgr 161
- winvmigrate 166
- winvpackage 167
- winvrnode 170
- winvsig 172
- winupdatecsid 175
- wloadiso 56, 178
- WLOADISO.LOG 178
- wlookup 19
- wlsinst 293
- wlspolm 223
- WMI scanner 8
- wmvinvcb 180
- wmvinvdh 181
- wputpolm 223
- wqueryinv 182
- wrimset 300
- wrimtest 292, 300
- wrimtrace 291
- wrpt 16, 113, 293
- wruninvquery 96
- wrunquery 94, 96
- wsetadmin 74
- wsetinvdh 184
- wsetinvglobal 187
- wsetinvpcfiles 192
- wsetinvpchw 196
- wsetinvpcsw 199
- wsetinvpvdconfig 203
- wsetinvpvdhw 204
- wsetinvpvdsw 205
- wsetinvswd 206
- wsetinvunixfiles 207
- wsetinvunixhw 211
- wsetinvunixsw 214
- wsetpr 224
- wsetrim 19
- wsusscan.cab
 - description 40
- wsusscn2.cab
 - description 40
- wtransfer 217
- wwaitscan 219
- wwebgw 302



Program Number: 5724-C06

Printed in USA

SC23-4713-06

