Tivoli Inventory
Scalable Collection User’s Guide
Version 3.6.2
Tivoli Inventory Scalable Collection User’s Guide (November 1999)

Copyright Notice
Copyright © 1999 by Tivoli Systems, an IBM Company, including this documentation and all software. All rights reserved. May only be used pursuant to a Tivoli Systems Software License Agreement or Addendum for Tivoli Products to IBM Customer or License Agreement. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without prior written permission of Tivoli Systems. The document is not intended for production and is furnished “as is” without warranty of any kind. All warranties on this document are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

Note to U.S. Government Users—Documentation related to restricted rights—Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.

Trademarks
The following product names are trademarks of Tivoli Systems or IBM Corporation: AIX, AS/400, IBM, OS/2, OS/390, RS/6000, Tivoli, Tivoli Management Framework, Tivoli Inventory, and Tivoli Software Distribution.

Microsoft, Windows, and Windows NT are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names mentioned in this document may be trademarks or servicemarks of others.

Notices
References in this publication to Tivoli Systems or IBM products, programs, or services do not imply that they will be available in all countries in which Tivoli Systems or IBM operates. Any reference to these products, programs, or services is not intended to imply that only Tivoli Systems or IBM products, programs, or services can be used. Subject to Tivoli System’s or IBM’s valid intellectual property or other legally protectable right, any functionally equivalent product, program, or service can be used instead of the referenced product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by Tivoli Systems or IBM, are the responsibility of the user.

Tivoli Systems or IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, New York 10504-1785, U.S.A.
Contents
Preface.................................................................................................................. vii

Chapter 1—Introduction to MCollect
Scalable Collection ............................................................................................. 1-1
Components of an MCollect-Enabled Environment........................................... 1-2
Data Collection with MCollect ........................................................................... 1-4
Features of MCollect........................................................................................... 1-6

Chapter 2—Installing MCollect
MCollect Installation Overview.......................................................................... 2-3
Installing MCollect ............................................................................................. 2-4
  Selecting the Media.................................................................................... 2-5
  Installing the Tivoli Management Platform MCollect Service ................. 2-7
  Installing Inventory ................................................................................... 2-9
  Installing the Inventory Receiver Patch ................................................... 2-9
  Creating RIM Objects .............................................................................. 2-13
  Creating an Inventory Receiver Object .................................................... 2-14
  Enabling MCollect ................................................................................... 2-14

Chapter 3—Using Inventory with MCollect
How MCollect Works with Inventory ............................................................... 3-2
Configuring a Profile to Use MCollect........................................................... 3-4
Using Collectors............................................................................................... 3-5
  Stopping, Starting, and Resetting a Collector ........................................... 3-5
  Configuring a Collector............................................................................. 3-6
Using the Inventory Receiver ......................................................................... 3-8
  Configuring the Inventory Receiver........................................................ 3-8
  Moving the Inventory Receiver................................................................. 3-10
Creating and Configuring RIM Objects........................................................... 3-10
Scheduling Collections .................................................................................... 3-11
Controlling Network Data Flow .................................................................... 3-14
Preface

The Tivoli Inventory Scalable Collection User’s Guide provides information about the MCollect service, which performs asynchronous data collection from endpoints in your Tivoli enterprise.

This document includes an introduction to MCollect and information about the management tasks that you can perform using MCollect and Tivoli Inventory, Version 3.6.2.

Who Should Read This Document

The target audiences for this document are system administrators and other users of Inventory who perform inventory gathering and reporting operations about PC and UNIX systems in a large, distributed enterprise. You should have some knowledge of the following:

- UNIX and PC architectures
- Basic inventory control and system configuration management concepts
- Database and Structured Query Language (SQL) concepts
- Tivoli Management Framework architecture, including gateways, endpoints, repeater sites, and the Multiplexed Distribution (MDist) service

Prerequisite and Related Documents

Before installing MCollect, read the Tivoli Inventory Release Notes, Version 3.6.2, which include the following:

- New features included in the Inventory, Version 3.6.2, release
- System requirements, including supported platforms, relational database management systems (RDBMSs), RDBMS Interface Module (RIM) hosts, Web browsers, and configuration repository space requirements
- Installation notes, including installing Inventory, Version 3.6.2, and prerequisites and information about upgrading from Inventory, Versions 3.6 and 3.6.1
- Defects fixed by Inventory 3.6.2
Preface

- Patches included with Inventory 3.6.2
- Known defects and workarounds in Inventory 3.6.2
- Product notes about Inventory 3.6.2

Also refer to the following prerequisite and related documentation:

- *Tivoli Inventory User’s Guide, Version 3.6*
  This guide provides information about how to manage hardware and software inventory on PCs and UNIX systems.

  These documents include prerequisite information about Framework.

**What This Document Contains**

The *Tivoli Inventory Scalable Collection User’s Guide* contains the following sections:

- Chapter 1, “Introduction to MCollect”
  This chapter provides an overview of how MCollect works, including descriptions of components and features.

- Chapter 2, “Installing MCollect”
  This chapter describes the steps required to install MCollect to work with Tivoli Inventory, Version 3.6.2.

- Chapter 3, “Using Inventory with MCollect”
  This chapter describes how the MCollect service works with Inventory, and provides instructions for using MCollect with Inventory.

- Appendix A, “Commands”
  This appendix lists and documents, in alphabetical order, the Tivoli commands related to MCollect.
Conventions Used in This Document

This guide uses several typeface conventions for special terms and actions. These conventions have the following meaning:

- **Bold**: Commands, keywords, flags, and other information that you must use literally appear in *bold*.
- **Italics**: Variables and new terms appear in *italics*. Words and phrases that are emphasized also appear in *italics*.
- **Monospace**: Code examples, output, and system messages appear in a *monospace* font.

Platform-Specific Information

The following table identifies the versions for each of the listed platforms that MCollect supports at the time of publication. For more detailed and up-to-date information, see the *Tivoli Inventory Release Notes, Version 3.6.2*.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Supported Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX 4.x</td>
<td>Managed node, endpoint: IBM RS/6000 series running AIX, Versions 4.21, 4.3, 4.3.1, and 4.3.2</td>
</tr>
<tr>
<td>HP-UX</td>
<td>Managed node, endpoint: HP9000/700 and 800 series running HP-UX, Versions 10.20, 11.00, and 11 SP 1</td>
</tr>
</tbody>
</table>
Contacting Customer Support

For support inside the United States, for this or any Tivoli product, contact Tivoli Customer Support in one of the following ways:

- Send e-mail to support@tivoli.com
- Call 1-800-TIVOLI8
- Navigate our Web site at http://www.support.tivoli.com


When you contact Tivoli Customer Support, be prepared to provide identification information for your company so that support personnel can assist you more readily.
We are very interested in hearing from you about your experience with Tivoli products and documentation. We welcome your suggestions for improvements. If you have comments or suggestions about this documentation, please send e-mail to pubs@tivoli.com.
Introduction to MCollect

This chapter describes the MCollect service, lists the components of an MCollect-enabled environment, describes how MCollect and Tivoli Inventory work together to collect data, and lists the features and advantages of MCollect.

Scalable Collection

Scalable collection is the ability to control how much data is collected on your network as well as when the data is collected. With scalable collection, you can significantly reduce the time needed to scan and store vast amounts of data. The MCollect service gives you the ability to scale data collection.

The MCollect service sends scan data to its destination as the scan on each endpoint completes. Another advantage of MCollect is its ability to stage data through a hierarchy of collector nodes to distribute the processing load across the Tivoli Management Region (TMR). In essence, MCollect 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.

The MCollect service is triggered by an endpoint that has data to be collected. This endpoint informs the gateway that the data is ready by sending the collection table of contents (CTOC) which includes, among other things, the data file name and size and the source and destination of the collected data. A collector daemon running on the gateway then retrieves the data from the endpoint 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.

Since several Tivoli products potentially can use the MCollect service, the final destination of the data depends on the Tivoli application using MCollect. With Inventory, all collectors send data to the Inventory receiver, which then sends the data to the configuration repository. (The Inventory receiver is described in detail later in this chapter.)

This release of MCollect is designed for Inventory users who want to improve performance and have greater control of network resources during Inventory scans, such as users who have medium to large Tivoli deployments with 100 or more endpoints.

**Components of an MCollect-Enabled Environment**

Any environment that uses MCollect for Inventory collections must include the following components:

- **Repeater sites** organized into a repeater hierarchy—systems that use the multiplexed distribution (MDist) service. MDist 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 endpoints that are targets of the distributed data.

  The MCollect service uses a collector hierarchy that mirrors the MDist repeater hierarchy. MCollect sends data upstream through this hierarchy, in the opposite direction of MDist distributions.

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

- **Collectors**—repeater sites on which MCollect has been installed. Specifically, a collector is an MCollect daemon process on either a managed node or gateway that stores and then forwards data to other collectors or to the Inventory receiver.

  This guide refers to any managed node as a collector if MCollect has been installed on the node, and the node has been added to the MCollect collection hierarchy.
Collectors are composed of the following components:

- The **depot**, which persistently stores data that has been collected from endpoints or other collectors. The depot also sends data to other collectors that request it.

- The **queues**, which hold the CTOCs. 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**—maintains the collector hierarchy based on repeater hierarchy information obtained from the MDist repeater manager.

  - An **Inventory receiver**—the Tivoli Inventory object that receives data from collectors and sends the data to one or more RDBMS Interface Module (RIM) objects. The Inventory receiver can be considered the final collector in an Inventory system. Like collectors, the Inventory receiver has a depot and queues. However, the Inventory receiver uncompresses and decodes the data and sends it to the RIM rather than requesting collection from an upstream collector.

  - The **status collector**—collects, stores, and distributes status information for each endpoint 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 receiver. If that node fails and is restarted, status information tracked by the status collector is automatically restored.
The Inventory configuration repository—holds the data collected by Inventory in a relational database management system (RDBMS).

One or more RIM objects (in addition to the existing Inventory RIM object)—connect Inventory to the RDBMS for access to the Inventory configuration repository. You can configure multiple RIM objects to write MCollect data in parallel to the configuration repository.

Data Collection with MCollect

The following diagram illustrates the way MCollect and Inventory work together to complete scans and store information in the configuration repository.
Data collection using Inventory and MCollect occurs in three major phases:

- First, the Inventory profile scans endpoints, which causes the endpoints to send CTOCs to MCollect.
- Next, MCollect processes the CTOCs and moves data through the collector hierarchy to the Inventory receiver.
- Finally, the Inventory receiver writes the data to one or more RIM objects configured for its use.

These phases are described in the following paragraphs.
In the first phase, you must distribute an Inventory profile that has been enabled for MCollect. For more information about enabling a profile, see “Configuring a Profile to Use MCollect” on page 3-4 and “wsetipcoll” on page A-29.

You distribute the profile to the endpoints. As each scan completes on each endpoint, Inventory generates a compressed and encoded RIM data file. The endpoint sends a CTOC, which contains information about the RIM data file, to the collector on the gateway that manages the endpoint. The collector daemon queues the CTOC for processing. The scan on the endpoint then completes. After the scan completes on all endpoints, the Inventory profile distribution is finished.

In the second phase, MCollect moves data from each endpoint to a depot in the gateway collector that controls the endpoint. While the data transfers from the endpoint to the depot, the CTOC for that data remains in the input queue of the collector. When the collector has collected all the data, the CTOC moves to the output queue. The collector then notifies the next collector in the hierarchy that the data is ready to be collected. The upstream collector places the CTOC in its input queue, and then collects the data from the downstream collector. When the data is completely transferred to the upstream collector, the downstream collector discards the CTOC in its output queue and data in its depot.

This same exchange happens between each collector in the hierarchy as the collectors asynchronously transfer the data to the Inventory receiver.

In the third phase, the Inventory receiver sends the collected data to any available RIM object configured to work with it. From the RIM object, the collected data is sent to the Inventory configuration repository. The Inventory receiver sends the completion status of each scanned endpoint, as well as any error information, to the status collector. The status collector changes the status for each endpoint from pending to successful or failed.

**Features of MCollect**

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

- Asynchronous data collection and backend processing lead to better distribution of processing load across more nodes in the TMR,
Features of MCollect

as well as better use of the network. MCollect stages the data at different collectors in the network as it flows to the RIM, and allows control over the data flow between successive collectors in the hierarchy.

- MCollect returns scan data as the scan of each endpoint completes. This feature reduces RIM overload at the end of a distribution.
- After data has been collected from an endpoint, that endpoint can be disconnected without affecting the MCollect service.
- The Inventory receiver can write data in parallel to multiple RIM objects. Multiple RIM objects improve throughput by allowing data for multiple targets to be written in parallel to the RDBMS.
- MCollect reduces the load on the TMR server. The Inventory receiver stages scan data for individual endpoints while the configuration repository is updated. Without MCollect, the TMR server holds the scan data for all endpoints in memory during the update of the configuration repository. (It is recommended that you create the Inventory receiver on a machine other than the TMR server.)
- With the Tivoli scheduler, you can schedule scans and data collection traffic for times when network traffic is at a minimum. See “Scheduling Collections” on page 3-11 for more information.
- Collectors along the route store collected data in depots. If a failure occurs, the collector can resend the data when it is back online rather than scanning all the endpoints again.
- MCollect provides functionality through which you can get status information about collectors, CTOCs, and scans. See Chapter 3, “Using Inventory with MCollect,” for more information about using the status features of MCollect.
- With MCollect, you can determine the status of individual targets as they complete, rather than having to wait until all targets complete.
- MCollect allows you to retry collecting data from endpoints or other collectors following a failure.
- With MCollect, an Inventory profile distribution completes when all endpoints have been scanned rather than when scan data reaches the configuration repository. Therefore, in most cases the
profile distribution completes (and users regain access to the graphical user interface or command line interface) more quickly.
Installing MCollect

The MCollect service extends the behavior of the Tivoli managed node and associated gateway objects. MCollect installs as a patch to the Tivoli Management Framework product. It is recommended that you install MCollect on all supported managed nodes in each Tivoli Management Region (TMR).

MCollect uses a collector hierarchy to collect, stage, and send data to its destination. The collector hierarchy is based on the repeater hierarchy established by the multiplexed distribution (MDist) service.

There are two circumstances under which you might be installing MCollect:

- In the first case, you are installing Tivoli products for the first time and have not established a repeater hierarchy.
- In the second case, your repeater hierarchy is already in place.

If you have not established a repeater hierarchy, MCollect uses a default configuration in which each gateway is a collector that sends data directly to the Inventory receiver.

If you have already established a repeater hierarchy, MCollect mirrors the repeater hierarchy. When you add a new repeater to your hierarchy, you must install MCollect on the managed node that contains the new repeater if you have not already done so.

Because the MCollect and MDist services share the same network topology, you should plan your hierarchy of collectors and repeaters carefully. For example, any changes that you make to the repeater hierarchy are also made to your collection hierarchy and vice versa. You
should therefore set up your network in a way that is optimal for both the MCollect and MDist services.

**Note:** When you change your repeater hierarchy, your collector hierarchy does not change until you run the `wcollect` command using the `-r` option on all affected collectors or reexecute the TMR server and all machines acting as collectors using the `odadmin` command. For more information about the `wcollect` command, see "wcollect" on page A-6. For more information about the `odadmin` command, see the *Tivoli Management Framework Reference Manual.*

It is usually beneficial to place a machine configured to be a collector and repeater on either end of a critical link in your network, such as a wide area network (WAN) connection. With this configuration, data has to be collected from only one machine across the WAN. The following figure illustrates this configuration.
For more information about setting up a repeater hierarchy, see the *Tivoli Management Framework Planning and Installation Guide*.

**MCollect Installation Overview**

Because MCollect is a Framework service, you must install or upgrade to Version 3.6.1 of the Framework before installing MCollect or any Inventory component.
Installing MCollect

The following steps represent the overview of the procedure to install MCollect to work with Tivoli Inventory:

1. Install or upgrade to Tivoli Management Framework, Version 3.6.1. See the *Tivoli Management Framework Planning and Installation Guide*.

2. Install the MCollect patch (Tivoli Management Platform MCollect Service) on all managed nodes. See “Installing MCollect.” (You must install the MCollect service before installing the Inventory receiver patch.)

3. Install or upgrade to Tivoli Inventory, Version 3.6.2, on your managed nodes and Tivoli Inventory Gateway, Version 3.6.2, on your gateways. (You can install or upgrade Inventory before or after installing MCollect.) See *Tivoli Inventory Release Notes, Version 3.6.2*.

4. Install the patch for the Inventory receiver (Tivoli Inventory, Version 3.6.2, Receiver) on the TMR server and on the managed node you want to serve as the Inventory receiver. See “Installing the Inventory Receiver Patch” on page 2-9.

5. Create one or more additional RDBMS Interface Module (RIM) objects that Inventory will use. See “Creating RIM Objects” on page 2-13. (You can create RIM objects before or after you create the Inventory receiver.)

6. Create an Inventory receiver object. See “Creating an Inventory Receiver Object” on page 2-14.

7. Enable MCollect for Inventory profiles that you want to use the MCollect service.

Before installing MCollect, read the *Tivoli Inventory Release Notes, Version 3.6.2*, for the latest information about system requirements, including supported platforms and space requirements.

## Installing MCollect

The following sections provide the procedures to install MCollect to work with Inventory.
Selecting the Media

Complete the following steps to obtain access from the Tivoli desktop to the files that you need to install:

1. Run the Tivoli desktop on the TMR server.
2. Select **Install → Install Patch...** from the **Desktop** menu. The **Install Patch** dialog is displayed.

If **Tivoli Management Platform MCollect Service** is already in the **Select Patch to Install** scrolling list, go to “Installing the Tivoli Management Platform MCollect Service” on page 2-7. Otherwise, continue with step 3.
3. Click the Select Media... button. The File Browser dialog is displayed.

![File Browser dialog](image)

4. Enter the location of the Tivoli CD-ROM in the Path Name text box by completing one of the following tasks:
   - Type the complete path name in the Path Name text box.
   - Browse the file system by completing the following steps:
     a. In the Hosts scrolling list, select the host (or drive) on which the CD-ROM is mounted. Choosing a host updates the Directories scrolling list to show the directories (under root) of the host you selected.
     b. In the Directories scrolling list, double-click the directory that contains the installation media. Choosing a directory updates the Files list.
     c. Click the Set Path button.
5. Click the Set Media & Close button and return to the Install Patch dialog. **Tivoli Management Platform MCollect Service** is displayed in the **Select Patch to Install** scrolling list.

![Image of the Install Patch dialog]

**Installing the Tivoli Management Platform MCollect Service**

Complete the following steps to install the MCollect patch on managed nodes that will act as gateways for managing endpoints, managed nodes that will serve as collectors, and the TMR server:

1. Select **Install → Install Patch**... from the **Desktop** menu. The **Install Patch** dialog is displayed.
2. In the **Install Patch** dialog, select **Tivoli Management Platform MCollect Service** from the **Select Patch to Install** scrolling list.

3. Make sure that the TMR server and all managed nodes on which you want to install MCollect appear in the **Clients to Install On** scrolling list.

   **Note:** It is recommended that you install MCollect on all supported managed nodes. You cannot install MCollect on OS/390 or OS/2 systems.

4. In the **Clients to Install On** scrolling list, select any managed nodes on which you do *not* want to install MCollect. Click the right-arrow button to move the selected managed nodes to the **Available Clients** scrolling list.
5. To begin installing MCollect, click the **Install** button. The **Patch Install** dialog provides a list of the operations that take place and warns you of any problems that you might want to correct before installing. You can choose one of the following options:

- Review the status information and click the **Continue Install** button. The **Patch Install** dialog informs you when installation is complete.
- To install MCollect at another time, click the **Cancel** button.

**Command Line**

You can use the `wpatch` command to install MCollect from the command line. The following example installs MCollect on the managed nodes x, y, and z.

```
wpatch -c $IMAGE_PATH/cdrom -i MCOLLECT.IND x y z
```

For more information about the `wpatch` command, see the *Tivoli Management Framework Reference Manual*.

**Installing Inventory**

To use MCollect with Inventory, you must install or upgrade to Tivoli Inventory, Version 3.6.2, on your managed nodes and Tivoli Inventory Gateway, Version 3.6.2, on your gateways. See *Tivoli Inventory Release Notes, Version 3.6.2* for the installation procedure.

**Installing the Inventory Receiver Patch**

You install the Inventory receiver patch on the TMR server and on the managed node that you want to serve as the Inventory receiver. After you install the patch, you create an Inventory receiver object on the managed node.

**Note:** You must create one or more RIM objects for use by the Inventory receiver. For more information, see “Creating RIM Objects” on page 2-13.

Complete the following steps to install the patch for the Inventory receiver.
Selecting the Media

Complete the following steps to obtain access from the Tivoli desktop to the Inventory files that you need to install:

1. Run the Tivoli desktop on the TMR server.
2. Select Install -> Install Patch... from the Desktop menu. The Install Patch dialog is displayed.

   If the Inventory receiver patch (Tivoli Inventory, Version 3.6.2, Receiver) is already in the Select Patch to Install scrolling list, go to “Installing Tivoli Inventory, Version 3.6.2, Receiver” on page 2-11. Otherwise, continue with step 3.

3. Click the Select Media... button. The File Browser dialog is displayed.
4. Enter the location of the Tivoli CD-ROM in the Path Name text box by completing one of the following tasks:
   - Type the complete path name in the Path Name text box.
   - Browse the file system by completing the following steps:
     a. In the Hosts scrolling list, select the host (or drive) on which the CD-ROM is mounted. Choosing a host updates the Directories scrolling list to show the directories (under root) of the host you selected.
     b. In the Directories scrolling list, double-click on the directory that contains the installation media. Choosing a directory updates the Files list.
     c. Click the Set Path button.
5. Click the Set Media & Close button and return to the Install Patch dialog. Tivoli Inventory, Version 3.6.2, Receiver is displayed in the Select Patch to Install scrolling list.
Installing Tivoli Inventory, Version 3.6.2, Receiver

You install the Inventory receiver patch (Tivoli Inventory, Version 3.6.2, Receiver) on the TMR server and on the managed node you want to serve as the Inventory receiver. It is recommended that you create the Inventory receiver on the same managed node as one or more MCollect RIM objects to minimize network delay time between the Inventory receiver and RIM. Complete the following steps to install the Inventory receiver patch:

1. Select Install -> Install Patch… from the Desktop menu. The Install Patch dialog is displayed.
2. In the **Install Patch** dialog, select **Tivoli Inventory, Version 3.6.2, Receiver** from the **Select Patch to Install** scrolling list.

3. Make sure that the TMR server and the managed node that you want to serve as the Inventory receiver appear in the **Clients to Install On** scrolling list.

   **Note:** You cannot install the Inventory receiver on OS/390 or OS/2 systems.

4. In the **Clients to Install On** scrolling list, select any managed nodes on which you do **not** want to install the Inventory receiver patch. Click the right-arrow button to move the selected managed nodes to the **Available Clients** scrolling list.

5. To begin installing the Inventory receiver patch, click the **Install** button. The **Patch Install** dialog provides a list of the operations that take place and warns you of any problems that you might want
Installing MCollect

to correct before installing. You can choose one of the following options:

- Review the status information and click the Continue Install button. The Patch Install dialog informs you when installation is complete.
- To install the Inventory receiver patch at another time, click the Cancel button.

Command Line

You can use the `wpatch` command to install the Inventory receiver patch from the command line. The following example installs the Inventory receiver patch on the TMR server rainier and the managed node fuji.

```
wpatch -c $IMAGE_PATH/cdrom -i 362_RCV.IND @ManagedNode:rainier @ManagedNode:fuji
```

For more information about the `wpatch` command, see the Tivoli Management Framework Reference Manual.

Creating RIM Objects

To use MCollect with Inventory, you must create one or more RIM objects in addition to the RIM object used by Inventory. The RIM object used by Inventory is named `inventory`. You must name MCollect RIM objects `inventoryn` where `n` is a positive integer of up to four digits. For example, if you create two additional RIM objects for use with MCollect, you might name them `inventory1` and `inventory2`.

To create a new RIM object for MCollect with Inventory, first use the `wgetrim` command to view the settings for the Inventory RIM object that was created during installation. Enter the command as follows:

```
wgettrim inventory
```

Next, use the `wcrtrim` command to create a new RIM object on a managed node. Configure the new RIM object using the same settings as the existing RIM object, but name the new RIM object as described previously. The following example interactively creates a RIM object called `inventory1` on managed node `fuji`:

```
wcrtrim -v Oracle -h fuji -d inventory -u tivoli -H /usr/ORACLE -s fuji.world inventory1
```

RDBMS password:
Installing MCollect

For more information about the `wcrtrim` and `wgetrim` commands, see the *Tivoli Management Framework Reference Manual*.

Creating an Inventory Receiver Object

Use the `wcrtinvrcvr` command to create an Inventory receiver object on the managed node of your choice. It is recommended that you create the Inventory receiver object on the same managed node as one or more of the MCollect RIM objects to minimize network delay time between the Inventory receiver and RIM.

For more information about the `wcrtinvrcvr` command, see “wcrtinvrcvr” on page A-13.

**Note:** You can create only one Inventory receiver object per TMR.

Enabling MCollect

Use the `-m` option of the `wsetipcoll` command to enable MCollect for Inventory profiles.

The following example enables MCollect for the Inventory profile HWProf.

```
wsetipcoll -m YES @InventoryProfile:HWProf
```

For more information about the `wsetipcoll` command, see “wsetipcoll” on page A-29.
This chapter describes how Tivoli Inventory and MCollect work together and provides information about configuring and using Inventory with MCollect.

This chapter includes the following:

- An explanation of how Inventory and MCollect gather scan data and store it in the configuration repository. (See “How MCollect Works with Inventory” on page 3-2.)

- Information about configuring Inventory profiles to use MCollect. (See “Configuring a Profile to Use MCollect” on page 3-4.)

- Information about stopping, starting, resetting, and configuring collectors. (See “Using Collectors” on page 3-5.)

- Information about configuring the Inventory receiver to write data to the RDBMS Interface Module (RIM) and manage scan status information, and the procedure to move the Inventory receiver. (See “Using the Inventory Receiver” on page 3-8.)

- Information about RIM objects. (See “Creating and Configuring RIM Objects” on page 3-10.)

- An explanation of how you can schedule when collections are transferred through the collection hierarchy. (See “Scheduling Collections” on page 3-11.)

- An explanation of how you can control the way MCollect sends data through your network. (See “Controlling Network Data Flow” on page 3-14.)
How MCollect Works with Inventory

- Procedures you can use to view information about collectors, collection tables of contents (CTOCs), queues, the Inventory receiver, MCollect-enabled scans, and Inventory profiles, as well as information about viewing data in the configuration repository. (See “Obtaining Information about MCollect” on page 3-15.)
- An explanation of how you can configure Inventory to gather and return status information about scans. (See “Configuring Inventory to Return Status Information” on page 3-18.)
- Information about using MCollect across Tivoli Management Regions (TMRs). (See “Considerations for Connected TMRs” on page 3-19.)

### How MCollect Works with Inventory

The following table describes how Inventory and MCollect interact to store Inventory data in the configuration repository:

<table>
<thead>
<tr>
<th>Action</th>
<th>Product or Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>You create an Inventory profile and enable it to use MCollect.</td>
<td>Inventory</td>
</tr>
<tr>
<td>You distribute the profile to endpoints.</td>
<td>Inventory</td>
</tr>
<tr>
<td>The following events occur on the endpoint as the scan completes:</td>
<td></td>
</tr>
<tr>
<td>1. A MIF file is created.</td>
<td></td>
</tr>
<tr>
<td>2. The MIF file is parsed.</td>
<td></td>
</tr>
<tr>
<td>3. RIM data is generated.</td>
<td></td>
</tr>
<tr>
<td>4. The RIM data is compressed, encoded, and saved in a file.</td>
<td></td>
</tr>
</tbody>
</table>

Version 3.6.2
### How MCollect Works with Inventory

<table>
<thead>
<tr>
<th>Action</th>
<th>Product or Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>After the scan finishes on an endpoint, the endpoint makes an upcall to the gateway to request data collection by sending a CTOC.</td>
<td>MCollect</td>
</tr>
<tr>
<td><strong>Note:</strong> If the endpoint cannot successfully forward the CTOC to the collector, the scan data for that endpoint is sent without using the MCollect service.</td>
<td></td>
</tr>
<tr>
<td>The profile distribution completes.</td>
<td>Inventory</td>
</tr>
<tr>
<td>The gateway forwards the CTOC to the local collector.</td>
<td>MCollect</td>
</tr>
<tr>
<td>When the collector on the gateway receives the CTOC, the following events occur:</td>
<td>MCollect</td>
</tr>
<tr>
<td>1. The collector places the CTOC in the input queue.</td>
<td></td>
</tr>
<tr>
<td>2. The collector makes a downcall to the endpoint to retrieve the compressed and encoded RIM data file.</td>
<td></td>
</tr>
<tr>
<td>3. The collector stores the data file in the collector depot. The CTOC moves from the input queue to the output queue.</td>
<td></td>
</tr>
<tr>
<td>4. The collector requests data collection from the next collector in the hierarchy by sending the CTOC to the upstream collector.</td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>6. Steps 1 through 5 repeat until the data file reaches the Inventory receiver.</td>
<td></td>
</tr>
<tr>
<td>The Inventory receiver stores CTOCs in an input queue and RIM data in a depot the same as collectors do. However, rather than sending a collection request to another collector, the Inventory receiver uncompresses the RIM data and sends it to an available RIM object.</td>
<td>Inventory</td>
</tr>
</tbody>
</table>
To modify an MCollect hierarchy, you must use the `wrpt` command to set up your repeater topology as needed, and then use the `wcollect` command with the `–r` option on the collectors in the modified part of the hierarchy to reset the collector routing topology (see “Using Collectors” on page 3-5). Because MCollect shares the multiplexed distribution (MDist) repeater hierarchy (although data flows in the opposite direction), you must configure the hierarchy so that it is optimized for both services. See the *Tivoli Management Framework Reference Manual* for more information about the `wrpt` command.

### Configuring a Profile to Use MCollect

Use the `wsetipcoll` command to configure an Inventory profile to use MCollect. With the `–m` option of this command, you can turn MCollect on or off for a particular profile.

For example, you might want to configure the profile WIN_HW to use MCollect. Enter the following command:

```
wsetipcoll -m YES @InventoryProfile:WIN_HW
```

where:

- **–m YES** Indicates that the profile should use MCollect to return data collected from the scan.

- **@InventoryProfile:WIN_HW** Specifies the name of the Inventory profile that must use MCollect.

For more information about the `wsetipcoll` command, see “wsetipcoll” on page A-29.
Using Collectors

The following sections discuss the various tasks you can perform on collectors using the **wcollect** command.

**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 A-6.

**Stopping, Starting, and Resetting a Collector**

You must stop and restart a collector to perform certain tasks with MCollect. 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 runtime directory, so that you can move data from the old directory to the new directory. You may also need to stop and restart a collector during troubleshooting.

MCollect provides two options for stopping a collector using the **wcollect** **–h** option:

- **graceful**—Bringing the collector to a graceful halt stops the collector after it completes all remaining 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 **–h** option, you must use the **–s** option to start the collector again.

If you have made changes to the collector hierarchy using the **wrpt** command, you must reset the collector topology in each affected collector’s local routing cache. Use the **–r** option of the **wcollect** command to remove locally cached collection routes on a collector. After you execute this command, the collector downloads new routing information from the collection manager when the collector processes the next collection request.
Configuring a Collector

You must configure collectors after installing MCollect. You may also need to reconfigure collectors to increase disk space for the depot or after making changes to the collector hierarchy. The configuration options for collectors are described in the following sections.

Configuring the Runtime Directory and Depot

The runtime directory contains the depot and runtime (*.dat and *.log) files. By default, the MCollect runtime directory is at $DBDIR/mcollect on each collector. The depot is located within the runtime directory at $DBDIR/mcollect/depot.

You can move the runtime 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 contends for disk space with the oserv, which also writes data to the $DBDIR directory.

To move the runtime directory and depot, first specify a new location for the runtime directory using the wcollect –l option, then move the depot and all other data from the old runtime directory to the new directory. The runtime 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.

Note: After you specify a new runtime directory, you must stop the collector before moving the depot and data from the old directory to the new directory.

You can change the size of the depot using the wcollect –z option. By default, the depot size is 40 megabytes (MB), but you can make it as large as necessary. Note that you cannot make it smaller; you can only make it larger. For example, you can set the depot to 50 MB and later change that to 55 MB, but you cannot change 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 –c option. The default chunk 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 endpoint fails. For more information, see “Troubleshooting” on page B-6.

Configuring the 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 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 into and out of the depots. A number of 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 endpoints, 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 your collector threads, you enable MCollect to recover from temporary error conditions.

Use the wcollect –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 `wcollect -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.

**Using the Inventory Receiver**

The Inventory receiver can be considered the last collector in the MCollect collector hierarchy. The Inventory receiver receives data from collectors and sends the data to the configuration repository. This section describes how to configure the Inventory receiver and how to move it to a different managed node.

**Configuring the Inventory Receiver**

You create an Inventory receiver object using the `wrtinvrcvr` command. You use the `wsetinvrcvr` command to specify how the Inventory receiver writes data to the RIM. You can also use this command to configure status collector options, which specify how notifications of scan status are collected, stored, and distributed. You use the `wcollect` command to configure output threads for the Inventory receiver.

This section describes the options you can use to configure the Inventory receiver. For more information about the `wrtinvrcvr`, `wsetinvrcvr` and `wcollect` commands, see Appendix A, “Commands.”

You use the `wsetinvrcvr -s` option to specify whether the Inventory receiver stores status information. Status information is a record of whether the scan has completed on each endpoint in a profile distribution. If a system failure occurs, this information is automatically restored when the Inventory receiver is back online. By default, status information is stored.

You use the `wsetinvrcvr -d` option to specify the directory where the Inventory receiver stores status information. The default location for this directory is `$DBDIR/inventory/stat_dir` on the managed node where the Inventory receiver resides. You can specify a different directory.

Using the `wsetinvrcvr -n` option, you can specify the interval at which status information is sent. For example, if you set this value to 10, every 10 minutes the Inventory receiver sends all completed scan notifications in a single notice called a bundle. The Inventory receiver sends the bundles to the Inventory notice group, a log file, or both, depending on the
setting of the \texttt{wsetipcoll -w} option. The default value for the \texttt{wsetinvrcvr -n} option is 10 minutes.

Using the \texttt{wsetinvrcvr -q} option, you can 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 receiver bundles the status information and sends it. The default value is 10 targets.

The \texttt{wsetinvrcvr -n} and \texttt{-q} options, in combination with the \texttt{-n} option of the \texttt{wsetipcoll} command, configure status information to be sent in bundles.

\textbf{Notes:}

\begin{itemize}
  \item If you set both \texttt{-q bundle\_every\_n\_targets} and \texttt{-n bundle\_every\_n\_minutes} to 0, no bundling occurs, and you are not notified until the scans on all targets are completed.
  \item If you set both \texttt{-q bundle\_every\_n\_targets} and \texttt{-n bundle\_every\_n\_minutes} to a positive value, bundling occurs when either value (the specified number of targets or minutes) is reached.
  \item Bundling occurs only when the \texttt{-n} option for the \texttt{wsetipcoll} command is set to \texttt{BUNDLE}.
\end{itemize}

You use the \texttt{wcollect -o} option to specify the number of output threads for the Inventory receiver. You should specify one output thread for each RIM object that you create to work with the Inventory receiver. For example, if you create two RIM objects for the Inventory receiver, you set the value for the \texttt{wcollect -o} option to 2.

The \texttt{wsetinvrcvr -r} option allows you to specify the number of times the Inventory receiver tries to write data to the RIM. After the specified number of retries is reached, the Inventory receiver sends a failure notification. The default value for this option is 5 retries.

Related to the \texttt{wsetinvrcvr -r} option, the \texttt{wsetinvrcvr -t} option allows you to specify a value from which to calculate the timeout period between retries. The Inventory receiver calculates the timeout period using the algorithm \textit{timeout*retry\_count}. For example, if you set this value to 20 seconds, on the first retry the algorithm sets the timer to 20 * 1 or 20 seconds. On the second retry, the timer sets to 20 * 2 or 40 seconds. The default value for the \texttt{wsetinvrcvr -t} option is 30 seconds.
Moving the Inventory Receiver

To move the Inventory receiver, you delete it from the managed node where it currently resides, and then create a new Inventory receiver on a different managed node. Before moving the Inventory receiver, note the following recommendations:

- Do not move the Inventory receiver while any Inventory scans are active. If you move the Inventory receiver before scans complete, you will lose any data that has not reached the configuration repository.
- When you create the new Inventory receiver, you must configure it using the same values that you used for the old Inventory receiver.

Perform the following tasks to move the Inventory receiver:

1. Make sure that the MCollect patch and the Inventory receiver patch have been installed on the managed node where you will create the new Inventory receiver.
2. Delete the old Inventory receiver by running the following command:

```
wdel @InventoryReceiver:inv_receiver
```

For more information about the `wdel` command, see the *Tivoli Management Framework Reference Manual*.
3. Create the new Inventory receiver using the `wcrinvrcvr` command and appropriate options.

Creating and Configuring RIM Objects

The Inventory receiver cannot use the RIM object used by Inventory. You must create and configure one or more RIM objects for the Inventory receiver to use to write data to the configuration repository.

To create one or more RIM objects for the Inventory receiver, first use the `wgetrim` command to view the settings for the Inventory RIM object that was created during installation.

Next, use the `wcrtrim` command to create a new RIM object on a managed node in your TMR.
Configure each new RIM object using the same settings as the Inventory RIM object. You must name MCollect RIM objects inventory\(n\) where \(n\) is a positive integer of up to four digits. For example, if you create two additional RIM objects for use with MCollect, you might name them inventory1 and inventory2.

For more information about the creating and configuring RIM objects, see “Creating RIM Objects” on page 2-13.

**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 links between collectors only. You cannot specify links 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 may 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 may choose to prohibit collections from the teller terminals during these hours. See the following figure for an illustration of this network.
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 the 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:
Scheduling Collections

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

where:

offlinks_range_to_prohibit_collection
    Specifies the object dispatcher IDs of the collectors for which links to the specified collector must be turned off.

collector
    Specifies the collector to which links must be turned off.

To get the object dispatcher ID of a collector, use the odadmin command and odlist option. For more information, see the Tivoli Management Framework Reference Manual.

You can list the object dispatcher IDs, 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 IDs:

wcollect -x "4-7" collector

You must enclose the range of offlinks in double quotation marks ("").

See “wcollect” on page A-6 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 starts and stops collectors using the `wcollect -s` and `-h` options. Alternatively, you can schedule a job that shuts down a collector’s output queue or input queue by setting the `wcollect -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 Network Data Flow

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

- **Offlinks**—Offlinks are the main mechanism to regulate MCollect traffic across your network. You can use offlinks to enable and disable MCollect 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 3-11 and “wcollect” on page A-6.

- **Transmission chunk size**—You use the `wcollect -c` option to configure the size of transmission chunks. This option allows control over the size of the MCollect data packet that crosses the inter-object message (IOM) connection. The default transmission chunk size is 1 MB. If you specify a smaller chunk size, then the application data send 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 MCollect data. You configure transmission chunk size on the downstream collector (the collector that sends the data). For more information about configuring transmission chunk size, see “Configuring a Collector” on page 3-6 and “wcollect” on page A-6.

**Note:** Offlinks and transmission chunks affect data transmission between collector nodes, or between a collector and the Inventory receiver. These mechanisms do not affect transmissions between an endpoint and the gateway collector.
Obtaining Information about MCollect

Input Threads—Collectors use input threads to open an IOM session to a downstream node 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 MCollect IOM sessions, or decrease the number to reduce MCollect IOM traffic coming into the collector. For more information about configuring a collector’s input threads, see “Configuring Threads” on page 3-7 and “wcollect” on page A-6.

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

Obtaining Information about MCollect

Using various MCollect 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 a collector’s queues
- The configuration of the Inventory receiver
- Details about current MCollect-enabled Inventory scans
- Details about the MCollect and Inventory status collector options of an Inventory profile

This section describes the commands you use to get information about MCollect, and provides information about viewing Inventory data stored in the configuration repository. For specific information about using these commands and their options, see Appendix A, “Commands.”

Collector Configuration

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
Obtaining Information about MCollect

log file, location of the runtime 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:
```
wccollect collector
```
where `collector` is of the form `@ManagedNode:collector_name`, `@Gateway:collector_name`, or `@InventoryReceiver:inv_receiver`.

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` or `@Gateway:collector_name`.

CTOCs and Queues

Use the `wcstat` command to get information about a CTOC or one or more of a collector’s queues.

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. To view information about a CTOC, enter the following command:
```
wstat -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`.

A collector has input, output, error, completed, and deferred queues. You can view the following information about a collector’s queues: CTOC ID, priority of queued data, collector status, source and destination information, the method to be called on the destination object, scan
Obtaining Information about MCollect

ID, collection status, and the number of times that this collector has retried submitting requests to an upstream collector.

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

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

where [ioecd] specifies the type of queue for which you want information (input, output, error, completed, or deferred), and collector is of the form `@ManagedNode:collector_name` or `@Gateway:collector_name`.

**Inventory Receiver Configuration**

The `wgetinvrcvr` command returns configuration information about the Inventory receiver, including scan completion notification and the number of retries and the timeout period for attempts to write data to a RIM object. The `-a` option causes the `wgetinvrcvr` command to return all available information about the Inventory receiver. You can use several other options to return specific information. See “wgetinvrcvr” on page A-19 for more information.

**Inventory Scans**

The `wgetinvstat` command returns details about Inventory scans. If you run the `wgetinvstat` 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 “wgetinvstat” on page A-21 for more information. You can use this command only for scans that have been configured to use the status collector using the `-m` or `-s` option of the `wsetipcoll` command.

**Inventory Profiles**

The `wgetipcoll` command returns information about whether MCollect is used for distribution of the specified profile. It also returns information about status collector options for that profile. If you run this command with the `-a` option, it returns all available information about the Inventory profile regarding MCollect and status collector options. You
can use other options that provide specific information about MCollect and status collector options. See “wgetipcoll” on page A-23 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 the section on querying Inventory information in the *Tivoli Inventory User’s Guide*. Inventory Version 3.6.2 also provides a set of Crystal Reports that you can use to view Inventory information in the configuration repository. These reports provide customized views of Inventory scan data. For more information about the Crystal Reports, see the *Tivoli Inventory Release Notes, Version 3.6.2*.

**Configuring Inventory to Return Status Information**

You can configure Inventory to gather and return status information about scans. Status refers to the success or failure of a scan on a particular Inventory profile target.

You use the `wgetipcoll` command to configure options for notification and logging of Inventory status information:

- The `-s` option controls whether the Inventory receiver collects status information for scans when using a specified profile.
- The `-h` and `-l` options specify where status information is logged.
- The `-n` option specifies when notification is sent when a scan that uses the status collector completes on each target. You can choose to send notification immediately, periodically, or when scans on all targets are complete.
- The `-t` option sets 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.
Considerations for Connected TMRs

- The `-w` option specifies the location to which notifications are sent. You can choose to send notification to the Inventory notice group, to the log file, both, or neither.

You use the `-n` and `-q` options of the `wcrtnvrcvr` or `wsetinvrcvr` commands, in combination with the `wsetipcoll -n` option, to configure bundling for status.

You can use the `wgetinvstat` command to return status information about current Inventory scans. You can use the `wgetipcoll` command to return information about status collector options for a specified profile.

For detailed information about these commands and their options, see Appendix A, “Commands.”

Considerations for Connected TMRs

You can connect TMRs using either one- or two-way connections. However, to use MCollect across the TMRs, you must configure a two-way connection between the TMRs.

When you connect TMRs, you must update resources. Connecting TMRs implies an initial exchange and periodic update of names and object identifiers contained in each server’s name registry. See the Tivoli Management Framework Planning and Installation Guide for more information about connecting TMRs and updating resources.

If you plan to run scans across TMRs and want Inventory to return the data using MCollect, you must install MCollect in both TMRs.

MCollect uses MDist repeater hierarchies to obtain wide area network (WAN) entry point information. For example, you may have two TMRs connected by a WAN: TMR A and TMR B. TMR A contains the Inventory receiver. A profile distributes from TMR A to endpoints in TMR B. The gateway for the endpoints in TMR B collects scan results from these endpoints. Next, the WAN-entry-point collector node in TMR A collects the scan results from the gateway in TMR B and sends the data to the Inventory receiver. The following figure illustrates this operation.
Considerations for Connected TMRs

TMR A
- Inventory Receiver
- RIM Host
- Managed Node Collector
- Gateway Collector
- Endpoints

TMR B
- Server
- Gateway Collector
- Endpoints

Profile Distribution
Collection from Gateway in TMR B
Collection from Collector Node in TMR A
Commands

This appendix lists and documents, in alphabetical order, the Tivoli commands related to Tivoli Inventory and MCollect. Tivoli commands enable you to perform system operations from a UNIX or PC command line.

Command Line Syntax

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

[ ] Identifies optional arguments. Arguments not enclosed in brackets are required.

… Indicates that you can specify multiple values for the previous argument.

| Indicates mutually exclusive information. You can use either the argument to the left of the separator or the argument to the right of the separator. You cannot use both arguments in a single use of the command.

{ } Delimits a set of mutually exclusive arguments when one of the arguments is required. If the arguments are optional, they are enclosed in brackets ([ ]).

Example

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

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 command line interface (CLI) commands:

- Registered names
- Object paths

Tivoli commands support both naming schemes. Sometimes you will find it 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 register 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. Some examples of registered names used as arguments for the wls and wmv commands are as follows:

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

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.
The name registry does not allow two resources of the same type to have the same name within a single Tivoli Management Region (TMR). However, it is possible for resource names to be duplicated within two or more connected TMRs. 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:

```plaintext
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 ‘..’. Some examples of object path names used as arguments for the `wls` and `wmv` commands are as follows:

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

The syntax for specifying a resource using the object-path name style 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.

### MCollect-Related Commands

The following table describes the MCollect-related commands that you can use with Inventory. Currently, you can perform MCollect tasks only from the CLI.
These commands fall into two major groups:

- Commands for MCollect components
  You use these commands to configure and get information about collectors. These commands include `wcollect` and `wcstat`. (You can also use the `wcollect` command to configure some options for the Inventory receiver.)

- Commands for Inventory components
You use these commands to control how Inventory works with MCollect, and to gather MCollect-related information from Inventory components. These commands separate into two categories:

- Commands you use with Inventory profiles. These commands include `wgetipcoll` and `wsetipcoll`.

- Commands you use with the Inventory receiver and status collector. These commands include `wrtinvcvr`, `wgetinvcvr`, `wgetinvstat`, and `wsetinvcvr`.

**Command Syntax**

This section lists MCollect commands, with syntax and descriptions of their functions. You can also access these listings either by using the `man` command on UNIX systems or by opening the `mcolcli.hlp` file on Windows NT systems.
wcollect

wcollect

Returns information about the current configuration of a collector, starts and stops a collector, resets collection routes that are cached in a collector, and configures options for a collector.

SYNOPSIS

wcollect [options] collector

where collector is of the form @ManagedNode:collector_name,
@Gateway:collector_name, or @InventoryReceiver:inv_receiver

The options are from one of the following categories:

- Options for active collectors
  [-s | -h immediate | graceful]

- Options for collector configuration and tuning
  [-d 0 | 1 | 2 | 3]
  [-g debug_log_size]
  [-l runtime_location]
  [-z depot_size]
  [-c depot_chunk_size]
  [-i thread_idle_down_time]
  [-p thread_sleep_time]
  [-t max_input_threads]
  [-m max_input_retries]
  [-o max_output_threads]
  [-e retry_delay_time]
  [-r]
  [-x offlinks_range_to_prohibit_collection]
  [-f true | false]
The function of the `wcollect` command is threefold. First, you can obtain information about the current configuration of a collector by simply running the command and specifying the collector for which you want information.

Second, 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 to do this are referred to as active-collector arguments.

Third, you can use the `wcollect` command to configure collector attributes. Options that allow you to do this are referred to as collector-configuration arguments.

**Note:** You can execute collector-configuration arguments at any time, but you must restart the affected collectors before the changes take effect. You halt and restart collectors using the `–h` and `–s` options of the `wcollect` command.

### Authorization

**senior**

### Arguments

The arguments can be divided into arguments for active collectors and arguments for collector configuration.

#### Active Collectors

```bash
–s
```

Starts the collector after it has been halted.

```bash
–h {graceful | immediate}
```

Halts the collector. The following options are available:

- **graceful**—The collector stops after completing any remaining active collections.
- **immediate**—The collector stops without waiting for active collections to finish processing.
Collector Configuration

`-d {0 | 1 | 2 | 3}`

Specifies the level of debugging information to log in the MCollect log file. The following options are available:

- 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 MCollect log file `mcollect.log` resides in the `$DBDIR/mcollect` directory of each collector.

`-g debug_log_size`

Specifies the maximum size of the collector log file `mcollect.log` in MB. 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.

`-l runtime_location`

Specifies the location of the runtime directory for the collector. The runtime directory contains the depot and runtime (*.dat and *.log) files. This directory must reside on a stable disk with a significant amount of free space to ensure persistent storage of collections. The runtime 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 runtime directory is at `$DBDIR/mcollect` on each collector, and the depot is at `$DBDIR/mcollect/depot`.

**Note:** After you specify a new runtime directory, you must stop the collector, and then move the depot and any data from the old runtime directory to the new directory.
-z depot_size  Specifies the size of the depot directory in megabytes (MB). By default, this value is set to 40 MB. You can make the depot larger than its previous size, 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.

-e 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.

-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.

-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.

-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.

-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. 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.

–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 attempts by using –m max_input_retries.

–r
Removes locally cached collection routes on each collector. After you execute this command, when the collector processes the next collection request, the collector loads new routing information from the collection manager.

Run 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.

–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.

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 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.

By default, no object dispatcher numbers are specified.

\[-f \{true | false\}\]

Specifies whether information about completed CTOCs should be written to the log file. CTOCs in any stage of the collection can be tracked using the **wcstat** command. By default, this value is set to true.

[@Gateway:collector_name]
[@ManagedNode:collector_name]

Specifies the name of the collector on which to run this command. Use **@Gateway:collector_name** for collectors that are on gateways. Use **@ManagedNode:collector_name** for collectors that are on managed nodes that are not configured to be gateways.

[@InventoryReceiver:inv_receiver]

Specifies that this command is to be run on the Inventory receiver.

**Examples**

The following examples illustrate various operations you can complete with the **wcollect** command.

**Request Information about Current Configuration**

The following example returns the current configuration for a collector:

```
 wcollect @ManagedNode:aztlan
```

The output is as follows:

```
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
```
Stop a Collector After Collections Complete

The following command performs a halt of a collector after processing is complete on all active collections:

```
wc locals -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 causes the links from the collector on aztlan to object dispatchers 2, 5, 6, 7, 8, and 11 to be turned off. Therefore, the collector on aztlan cannot collect data from those systems. (This command does not affect data collection between endpoints 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)
**wcrtinvrcvr**

Creates an instance of the Inventory receiver object.

**SYNOPSIS**

```
wcrtinvrcvr [-d status_directory] [-n bundle_every_n_minutes] [-q bundle_every_n_targets] [-r max_RIM_retries] [-s {YES | NO}] [-t RIM_retry_delay_time] ManagedNode
```

**DESCRIPTION**

You use the `wcrtinvrcvr` command to create the Inventory receiver instance `@InventoryReceiver:inv_receiver` in the Tivoli object database on the managed node that you specify. The Inventory receiver is the Tivoli Inventory object that receives data from collectors and sends the data to the configuration repository. In addition, you can specify the interval at which scan completion notifications are sent. (You can specify this interval only for Inventory profiles in which the `-n` option for `wsetipcoll` is set to BUNDLE.) You can also set the number of retries and the timeout period for writing data to the RDBMS Interface Module (RIM) host.

**Note:** You can have only one instance of the Inventory receiver object in a TMR.

**Authorization**

`admin`, `senior`, or `super`

**Arguments**

- `-d status_directory`
  
  Specifies where the Inventory receiver 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 receiver resides.

- `-n bundle_every_n_minutes`
  
  Specifies the interval at which scan completion notifications are sent (when the `-n` option for the
**wsetipcoll** 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 **wsetinvrcvr** or **wsetinvrcvr** 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 is sent at one time. Status refers to the success or failure of a scan on a particular Inventory profile target.

The default value is 10 targets.

This option, in combination with the **–n** option of the **wsetipcoll** command, configures status information to be sent in bundles.

**Notes:**

- 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_RIM_retries**

Specifies the number of times the Inventory receiver tries to write data to the RIM host. After the maximum number of retries is reached, a failure notification is sent.

The default value is 5.
The following example creates the Inventory receiver on the managed node mckinley, sets the time interval for notification to 20 minutes (for Inventory profiles with notification set to BUNDLE—see the wsetipcoll command), and sets the number of retries for RIM failures to 3.

```
wcrtinvrcvr -n 20 -r 3 mckinley
```

SEE ALSO

wcollect, wgetinvrcvr, wgetipcoll, wsetinvrcvr, wsetipcoll
vcstat

vcstat

Returns status information about a collector, returns information about a CTOC, and returns the contents of a collector’s queues.

SYNOPSIS

vcstat collector
vcstat –v ctoc_id collector
vcstat –q [ioecd] collector

where collector is of the form @ManagedNode:collector_name or @Gateway:collector_name

DESCRIPTION

With the vcstat 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.

Authorization

senior

Arguments

collector Specifies the name of the collector on which you want to run the vcstat 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

–v ctoc_id Specifies the CTOC for which you want status information returned.
wcstat

-q [ioecd]  Specifies the type of queue to be returned. The following options return the contents of the specified queue:

- i—input queue
- o—output queue
- e—error queue
- c—completed queue
- d—deferred queue

Examples

The following examples illustrate operations 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
```

The output is as follows:

CTOC ID: wepm_ctoc_928780348  
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::InventoryReceive  
  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:

```
wstat -v ctoc3_11836_9110 @ManagedNode:calypso
```
The output is as follows:

CTOC ID: ctoc3_11836_9110
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:
  MCDEST_DIR: /tmp
Collection Status: QUEUED_OUTPUT
#Retries: 0

**SEE ALSO**

wcollect
wgetinvrcvr

Returns configuration information about the Inventory receiver.

SYNOPSIS


DESCRIPTION

The wgetinvrcvr command returns information about the Inventory receiver, including how notification occurs, the number of retries, and the timeout period for writing data to the RIM host.

Authorization

user, admin, senior, or super

Arguments

-a Returns all available information about the Inventory receiver.

-d Returns the value of status_directory as set by the wcrinvrcvr or wsetinvrcvr commands. This directory stores status information so that it can be restored in case of a system failure.

-n Returns the value of bundle_every_n_minutes as set by the wcrinvrcvr or wsetinvrcvr 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 wcrinvrcvr or wsetinvrcvr commands, which specifies the maximum number of targets in a bundle.

A bundle is a group of targets about which status is sent to the Inventory notice group at one time. Status refers to the success or failure of a scan on a particular Inventory profile target. This option as set by the wcrinvrcvr or wsetinvrcvr commands, in combination with the -n option of the wsetipcoll
wgetinvrcvr

command, configures status information to be sent in bundles.

-\texttt{-r} \quad \text{Returns the value of max\_RIM\_retries as set by the \texttt{wcrtnvrcvr} or \texttt{wsetinvrcvr} commands, which specifies the number of times the Inventory receiver must try to write data to the RIM host before returning a failure for the target to which the data is associated.}

-\texttt{-s} \quad \text{Returns information about whether the Inventory receiver stores status information in case of a system failure. If YES is returned, status information is stored. If NO is returned, status information is not stored.}

-\texttt{-t} \quad \text{Returns the value of RIM\_retry\_delay\_time as set by the \texttt{wcrtnvrcvr} or \texttt{wsetinvrcvr} commands, which specifies a value used to calculate the timeout period in seconds between retries. 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.}

\textbf{Examples}

The following example returns all available information about the Inventory receiver:

\texttt{wgetinvrcvr -a}

The output is as follows:

\begin{verbatim}
Send bundled notification every: 1 minutes
Send bundled notification every: 3 targets
Max retries for RIM errors: 5
Retry delay time for RIM errors: 30 seconds
Save status in the directory: /Tivoli/db/mckinley.db/inventory/stat_dir
Save status in case of failure: YES
\end{verbatim}

\textbf{SEE ALSO}

\texttt{wcollect, wcrtnvrcvr, wsetinvrcvr, wsetipcoll}
wgetinvstat

wgetinvstat

Returns information about current MCollect-enabled Inventory scans.

SYNOPSIS

wgetinvstat
wgetinvstat –a [–s] [–f] [–p]
wgetinvstat –i id [–i id]... [–s] [–f] [–p]

DESCRIPTION

The wgetinvstat 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.

Authorization

Inventory_view, user, admin, senior, or super

Arguments

–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.
–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 id Returns information about the scan with the specified scan ID. You can specify multiple scan IDs.
wgetinvstat

**Examples**

The following example returns detailed information about all running scans and lists failed, successful, and pending scans.

`wgetinvstat -a -f -s -p`

The output is as follows:

```
Scan Id:            144
Profile Name:       Hardware
Start time:         Thu Aug 05 17:18:57 1999
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
Profile Name:       Hardware
Start time:         Thu Aug 05 17:18:57 1999
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:
    suntm11
```

**SEE ALSO**

wcollect, wstat
wgetipcoll

Returns information about an Inventory profile regarding MCollect and Inventory status collector options.

SYNOPSIS

    @InventoryProfile:profile_name

DESCRIPTION

The wgetipcoll command returns information about whether MCollect is used for distribution of the specified profile. It also returns information about status collector options for that profile.

Authorization

    Inventory_view, user, admin, senior, or super

Arguments

    –a          Returns all information about the Inventory profile regarding MCollect and status collector options.
    –h          Returns the name of a managed node to which Inventory status information is logged.
    –l          Returns the path name of a file to which Inventory status information is logged.
    –m          Returns information about whether MCollect will be used when a scan is performed with the specified profile. If YES is returned, MCollect is used. If NO is returned, MCollect is not used.
    –n          Returns information about when notification is sent when a scan that uses MCollect completes on each target. The following values can be returned:

        ■ IMMEDIATE—indicates that notification is sent as the scan on each target completes.
wgetipcoll

- 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 receiver. Refer to the `wcrinvrcvr`, `wgetinvrcvr`, and `wsetinvrcvr` commands for more information on bundling.

-s
Returns information about whether the status collector is used when processing the scan. If YES is returned, the status collector is used. If NO is returned, the status collector is not used.

-t
Returns information about whether notification is sent for a target depending on the completion status of a scan. The following values can be returned:
- 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

-w
Returns one of the following values, which indicate 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 `wsetipcoll -h` and `-l` options
- OFF—logs no status
- ALL—logs information to both the notice group and the log file

@InventoryProfile:profile_name

Specifies the profile name about which you want information.
Examples

The following example returns information about the Inventory profile HWProf regarding MCollect and status collector options:

wgetipcoll -a @InventoryProfile:HWProf

The output is as follows:

Use MCollect: YES
When to notify: DONE
Notify for: ALL (both SUCCESS and FAIL)
Notify where: LOG_FILE
Log file host: mckinley
Log file name: /tmp/hwprof.log
Collect status: YES

SEE ALSO

wcollect, wcrinvrcvr, wgetinvrcvr, wsetinvrcvr, wsetipcoll
wsetinvcvr

wsetinvcvr
Changes the settings of the Inventory receiver.

SYNOPSIS

wsetinvcvr [-d status_directory] [-n bundle_every_n_minutes] [-q bundle_every_n_targets] [-r max_RIM_retries] [-s {YES | NO}] [-t RIM_retry_delay_time]

DESCRIPTION

The wsetinvcvr command allows you to make changes to the Inventory receiver that you created using the wcrtinvcvr command. The options are identical to the wcrtinvcvr command except that you do not specify a managed node. The wsetinvcvr command automatically makes the changes to the Inventory receiver instance @InventoryReceiver:inv_receiver in the Tivoli object database.

Authorization

admin, senior, or super

Arguments

-d status_directory
Specifies where the Inventory receiver 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 receiver resides.

-n bundle_every_n_minutes
Specifies the interval at which scan completion notifications are sent (when the -n option for the wsetipcoll 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 \texttt{wcertinvrcvr} or \texttt{wsetinvrcvr} command.

\textbf{\texttt{-q bundle_every_n_targets}}

Specifies the maximum number of targets in a bundle. A bundle is a group of targets about which status is sent at one time. Status refers to the success or failure of a scan on a particular Inventory profile target.

The default value is 10 targets.

This option, in combination with the \texttt{-n} option of the \texttt{wssetipcoll} command, configures status information to be sent in bundles.

\textbf{Notes:}

- If you set both \texttt{-q bundle_every_n_targets} and \texttt{-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 \texttt{-q bundle_every_n_targets} and \texttt{-n bundle_every_n_minutes} to a positive value, bundling occurs when either value (the specified number of targets or minutes) is reached.

\textbf{\texttt{-r max_RIM_retries}}

Specifies the number of times the Inventory receiver tries to write data to the RIM host. After the maximum number of retries is reached, a failure notification is sent.

The default value is 5.

\textbf{\texttt{-s \{YES | NO\}}} Specifies whether the Inventory receiver stores status information that can be restored in case of a system failure. If you set this option to \texttt{YES}, status information is stored. If you set this option to \texttt{NO}, status information is not stored. By default, status information is stored.
-t RIM_rety_delay_time
Specifies a value from which to calculate the timeout period in seconds between retries. 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.

Examples
The following example changes the maximum number of targets in a notification bundle to nine and the RIM host retry delay time to 20 seconds.

wsetinvrcvr -q 9 -t 20

SEE ALSO
wcollect, wcrtinvrcvr, wsetinvrcvr wsetipcoll
**wsetipcoll**

Enables and disables MCollect for an Inventory profile and configures options for notification and logging of Inventory status information.

**SYNOPSIS**

```
wsetipcoll [-h log_file_host] [-l log_file_name] [-m {YES | NO}] [-n {IMMEDIATE | BUNDLE | DONE}] [-s {YES | NO}] [-t {SUCCESS | FAIL | ALL}] [-w {NOTICE_GROUP | LOG_FILE | OFF | ALL}] @InventoryProfile:profile_name
```

**DESCRIPTION**

You can use the `wsetipcoll` command to complete the following tasks:

- Specify a managed node and directory for the Inventory status log file
- Specify whether MCollect is used when the Inventory profile is distributed
- Configure notification settings

**Authorization**

`admin`, `senior`, or `super`

**Arguments**

- **-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 host will be the managed node on which the Inventory receiver was created.

- **-l log_file_name**
  Allows you to specify the path 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 will be 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

\textbf{\texttt{\textbf{\textbackslash m \{YES | NO\}}} \textbf{\textbackslash n \{IMMEDIATE | BUNDLE | DONE\}}}

Specifies whether MCollect is used when the profile is distributed. If you set this option to \texttt{YES}, MCollect is used. If you set this option to \texttt{NO}, MCollect is not used. The default value is \texttt{NO}.

\textbf{\texttt{\textbackslash s \{YES | NO\}}} Controls whether the Inventory receiver collects status information for scans when using this profile. You can use the \texttt{wgetinvstat} command to return this status information.

You can use this option to gather information on all Inventory scans, including those that do not use MCollect.

If you set this option to \texttt{YES}, the status collector is used. If you set this option to \texttt{NO}, the status collector is
not used. If you set the –m option to YES, this option is automatically set to YES.

The default value is NO.

–t {SUCCESS | FAIL | ALL}
Sets under what circumstances a notice is sent for the specified profile. The following options are available:

- **SUCCESS**—Notification is sent for targets on which scans complete successfully.
- **FAIL**—Notification is sent for targets on which scans failed.
- **ALL**—Notification is sent for all targets.

The default value is ALL.

–w {NOTICE_GROUP | LOG_FILE | OFF | ALL}
Specifies the location to which notifications are sent. The following options are available:

- **NOTICE_GROUP**—sends status information to the Inventory notice group
- **LOG_FILE**—logs status information to the log file specified with the –h and –l options
- **OFF**—logs no status
- **ALL**—logs information to both the notice group and the log file

The default value is NOTICE_GROUP.

**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.

@InventoryProfile:profile_name

Specifies the Inventory profile to which these settings apply.
Examples

The following example specifies properties for the Inventory profile HWProf so that notification for scans is written to the file /tmp/hwprof.log on managed node mckinley.

```
wssetipcoll -h mckinley -l /tmp/hwprof.log -w LOG_FILE
@InventoryProfile:HWProf
```

The following example specifies properties for the Inventory profile SWProf so that only notification for targets that fail is sent. In addition, the notification for scans will be bundled (sent periodically) according to the settings on the Inventory receiver. (Refer to the `wsetinvrcvr` and `wgetinvrcvr` commands.)

```
wssetipcoll -t FAIL -n BUNDLE @InventoryProfile:SWProf
```

SEE ALSO

`wcollect, wcrtinvrcvr, wgetinvrcvr, wgetinvstat, wgetipcoll, wsetinvrcvr`
Troubleshooting MCollect

This appendix describes files and procedures that can help you research and resolve problems that you may encounter while using the MCollect service. In general, you should check the Tivoli 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.

Log and Data Files

The following sections describe MCollect-related log and data files. These files reside in one of the following directories, unless a different directory is specified:

- On the Inventory receiver—$DBDIR/inventory/mc/file_name
- On all other collectors—$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

MCollect logs information about a collector’s queue data in the following queue data files:
Log and 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 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 3-11 and “wcollect” on page A-6.

**Note:** The .dat files for the queues exist so that if a collector shuts down, MCollect can recover and continue processing the collected data where it left off.

You can view these files using the `wcollect -q` command.

These files are four 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 four 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 A-6.

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 mcollect.log activity, be alert for the strings WARNING, ERROR, and exception, which indicate error conditions.

Inventory Receiver Log File

The Inventory receiver 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 receiver message. For more information on configuring and monitoring this log file, see “Collector Log File” on page B-2.

Collection Manager Log File

The collection manager is installed on the same system as your Tivoli Management Region (TMR) server. The collection manager log file is $TMPDIR/mcollect_collmgr.log on the TMR server, where $TMPDIR is a temporary directory. This temporary directory is usually one of the following paths:

- For UNIX: /tmp, /usr/tmp, or /var/tmp
- For Windows NT: 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
Log and Data Files

Inventory Status Log File

By default, Inventory sends status information about scans to the Inventory notice group only. Using the `wsetipcoll -w` 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: `/tmp`, `/usr/tmp`, or `/var/tmp`
- For Windows NT: `c:\temp`

Note: On UNIX systems, `$TMPDIR` is the temporary directory returned in the `tmpnam` system call. On Windows NT systems, `$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:

```
wcollect collector
```

where `collector` is of the form `@ManagedNode:collector_name` or `@Gateway:collector_name`.

You use the `wcollect -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.

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 MCollect troubleshooting, you can check this log file for information about exceptions that MCollect has created or connectivity problems. The $LCF_DATDIR directory is created when you install Framework. For more information about the lcfd.log file, see the Tivoli Management Framework Reference Manual and the Tivoli Management Framework Planning and Installation Guide.

- **/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.
Depot Contents

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

This section covers resolutions for situations you might encounter when using MCollect. For more information about the commands described in this section, see Appendix A, “Commands,” and the *Tivoli Management Framework Reference Manual*.

- **The scan completes but the data is not in the database.**

  Without MCollect, Inventory profiles complete when data is populated in the configuration repository. However, a profile enabled with MCollect completes when all targets have been physically scanned, but data has not necessarily been populated in the configuration repository.

  The `wgetinvstat` 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 an endpoint has reached the configuration repository, run the command as follows:

  `wgetinvstat -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.
Troubleshooting

- **The scan seems to take too long to finish.**
  
  First, use the `wgetinvstat` command to view the progress of the endpoints being scanned. If scans for most of the endpoints 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:

  ```bash
  wrimtest -l rim_object_label
  ```

  where `rim_object_label` is the name of RIM object used by the Inventory receiver.

  **Note:** The RIM object used by Inventory is different from that used by the Inventory receiver. Also, your Inventory receiver might be connected to more than one RIM object. To properly diagnose the problem, you must perform this test on each RIM object used by the Inventory receiver.

  To exit the `wrimtest` utility, enter the `x` command option.

  For more information about the `wadminep` and `wrimtest` commands, see the *Tivoli Management Framework Reference Manual*.

  Also, check the names of the RIM objects to make sure they are named correctly. The RIM object used by Inventory must be named `inventory`. Additional RIM objects used by the Inventory receiver must be named `inventoryn` where `n` is a positive integer, such as `inventory1` and `inventory2`.

  You should also make sure all RIM objects share the same configuration. Run the following command for each RIM object to check its configuration:

  ```bash
  wgetrim rim_object_label
  ```

  where `rim_object_label` is the name of RIM object.

- **The profile is configured to use MCollect, but it is not using it.**
  
  To troubleshoot this problem, perform the following actions:

  - Run the `wgetipcoll` command to verify that the profile is enabled for MCollect.
Troubleshooting

- Run the `wgetinvcvr` command to verify that the Inventory receiver exists.
- Run the `wgettrim` command to verify that one or more RIM objects have been created for use with MCollect.
- Verify that the platform type of the endpoint is supported by MCollect.
- Verify that the platform type of the gateway is supported by MCollect. If it is not, Inventory will not use MCollect to collect from that gateway.
- Verify that MCollect 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 MCollect.

Notes:
- MCollect can currently be installed on Solaris, HP-UX, AIX, and Windows NT managed node platforms.
- MCollect returns data for scans of endpoints only. Inventory can scan managed nodes or PC managed nodes, but it returns the data without using the MCollect service.

■ 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 receiver, the Inventory receiver 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 receiver.
Troubleshooting

- **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 3-11 and “wcollect” on page A-6.
  
  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 receiver. If connectivity is lost between any two collectors in this route, the Inventory receiver 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 endpoint to the Inventory receiver. 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.
Glossary

A

absolute path
A path that begins with the root directory. The absolute path may also be known as the “full pathname.” Contrast with relative path.

B

bandwidth
A measure of the capacity of a communication transport medium (such as a TV cable) to convey data.

C

cache
A buffer storage that contains frequently accessed instructions and data; it is used to reduce access time.
An optional part of the directory database in network nodes where frequently used directory information may be stored to speed directory searches.
To place, hide, or store in a cache.

CLI
See command line interface (CLI).

collector
In a Tivoli environment, either (a) a repeater site on which MCollect is installed or (b) an MCollect daemon on a managed node or gateway that stores and then forwards data to other collectors or to the Inventory receiver.
command line interface (CLI)
A type of computer interface in which the input command is a string of text characters.

configuration
The devices and programs that make up a system, subsystem, or network.

configuration repository
In a Tivoli environment, a RIM repository that contains information that is collected or generated by Tivoli Inventory and Tivoli Software Distribution. For example, in the configuration repository, Tivoli Inventory stores information regarding hardware, software, system configuration, and physical inventory; and Tivoli Software Distribution stores information regarding file package operations.

daemon
A program that runs unattended to perform a standard service. Some daemons are triggered automatically to perform their task; others operate periodically.

database
A collection of data with a given structure for accepting, storing, and providing, on demand, data for multiple users.
A collection of interrelated data organized according to a database schema to serve one or more applications.
A collection of data fundamental to a system.
A collection of data fundamental to an enterprise.

desktop
A graphical user interface (GUI) that enables a user to interact with and perform operations on a computer system.
directory
In a hierarchical file system, a container for files or other directories. See *path*.

downcall
In a Tivoli environment, a method invocation from the TMR 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, a Tivoli client that is the ultimate recipient for any type of Tivoli operation.
In a Tivoli environment, a Tivoli service that runs on multiple operating systems and performs Tivoli operations on those systems, thereby enabling the Tivoli Management Framework to manage the systems as Tivoli clients.

exception
An abnormal condition such as an I/O error encountered in processing a data set or a file.

G
gateway
In a Tivoli environment, software running on a managed node that provides all communication services between a group of endpoints and the rest of the Tivoli environment. This gateway includes the
multiplexed distribution (MDist) function, enabling it to act as the fanout point for distributions to many endpoints.

**Graphical User Interface (GUI)**

A type of computer interface consisting of a visual metaphor of a real-world scene, often of a desktop. Within that scene are icons, representing actual objects, that the user can access and manipulate with a pointing device. Contrast with **Command Line Interface (CLI)**.

**Host**

A computer that is connected to a network (such as the Internet or an SNA network) and provides an access point to that network. Also, depending on the environment, the host may provide centralized control of the network. The host can be a client, a server, or both a client and a server simultaneously. In a Tivoli environment, a computer that serves as a managed node for a profile distribution.

**Inventory Receiver**

In a Tivoli MCollect topology, the Tivoli Inventory object that receives data from a Tivoli Inventory scan and uses one or more connections to send the data to the configuration repository.

**Managed Node**

In a Tivoli environment, any managed resource on which the Tivoli Management Framework is installed.

**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.

**man page**

In UNIX systems, one page of online documentation. “Man page” is an abbreviation for “Manual page.” Each UNIX command, utility, and library function has an associated man page that can be viewed by entering this command: `man command name`.

**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.

**MCollect**

Multiplexed collection. In a Tivoli environment, a service that enables efficient collection of large amounts of data across complex networks.

**MDist**

Multiplexed distribution. In a Tivoli environment, a service that enables efficient distribution of large amounts of data across complex networks.

**method**

In object-oriented design or programming, the software that implements the behavior specified by an operation.

**MIF**

See Management Information Format (MIF).

**multiplexed distribution (MDist)**

See MDist.
name registry

In a Tivoli environment, a name service consisting of a two-dimensional table that maps resource names to resource identifiers and corresponding information within a Tivoli Management Region.

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.

notice group

In a Tivoli environment, an application- or operation-specific container that stores and displays notices pertaining 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.

notification

An unscheduled, spontaneously generated report of an event that has occurred.

In systems management, information emitted by a managed object relating to an event that has occurred within the managed object, such as a threshold violation or a change in configuration status.

object dispatcher

See object request broker (ORB).

object path

In a Tivoli environment, an absolute or relative path to a Tivoli object, similar to paths in file systems.
**object request broker (ORB)**

In object-oriented programming, software that serves as an intermediary by transparently enabling objects to exchange requests and responses.

**oserv**

The name of the object request broker used by the Tivoli environment. Oserv runs on the TMR server and each TMR client.

**packet**

In data communication, a sequence of binary digits, including data and control signals, that is transmitted and switched as a composite whole. The data, control signals, and, possibly, error control information are arranged in a specific format.

**patch**

A code change that is sent to the owners of a software product license after the release of a product. The licensees can then apply this code change to correct a reported problem.

**path**

A list of one or more directory names and an object name (such as the name of a file) that are separated by an operating system-specific character, such as the slash (/) in UNIX operating systems, the backslash (\) in Windows operating systems, and the semicolon (;) in OS/2 operating systems. The directory names detail the path to follow, in left-to-right order, to locate the object within the file system. This concept of path is also known as the “pathname.”

A list of directory names, usually separated by a colon (:), that are to be searched (in left-to-right order) to locate an object. This concept of path is also known as the “search path.”

**PC managed node**

In a Tivoli environment, an object that represents a client PC. The Tivoli Management Framework can communicate with the client PC only if the
PC agent is installed on the PC. Client PCs are most often referred to as PC managed nodes.

**persist**
To be maintained across session boundaries, usually in nonvolatile storage such as a database system or a directory.

**persistent**
Pertaining to data that is maintained across session boundaries, usually in nonvolatile storage such as a database system or a directory.

**platform**
An ambiguous term that may refer to the hardware, the operating system, or a combination of the hardware and the operating system on which software programs run.

**policy**
In a Tivoli environment, a set of rules that are applied to managed resources. A specific rule in a policy is referred to as a “policy method.”

**policy region**
In a Tivoli environment, a group of managed resources that share one or more common policies. Tivoli administrators use policy regions to model the management and organizational structure of a network computing environment. The administrators can group similar resources, define access to and control the resources, and associate rules for governing the resources. The policy region contains resource types and the list of resources to be managed. A policy region is represented on the Tivoli desktop by an icon that resembles a capitol building (dome icon). When a Tivoli Management Region (TMR) is created, a policy region with the same name is also created. In this case, the TMR has only one policy region. However, in most cases, a Tivoli administrator creates other policy regions and subregions to represent the organization of the TMR. A TMR addresses the physical connectivity of resources whereas a policy region addresses the logical organization of resources.
**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; the template includes information about the resources that can be managed by that Tivoli application.

A profile is created in the context of a profile manager; the profile manager links a profile to the Tivoli resource (for example, a managed node) that uses the information contained in the profile. A profile does not have any direct subscribers.

**query**

In a Tivoli environment, a combination of statements that are used to search the configuration repository for systems that meet certain criteria.

**R**

**RDBMS**

See relational database management system (RDBMS).

**relational database management system (RDBMS)**

A collection of hardware and software that organizes and provides access to a relational database.

**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.
relative path
A path that begins with the working directory. Contrast with absolute path.

repeater
A node of a local area network; a device that regenerates signals in order to extend the range of transmission between data stations or to interconnect two branches. See repeater site.

repeater site
In a Tivoli Management Region, a managed node that is configured with the MDist feature. A repeater site receives a single copy of data and distributes it to the next tier of clients.

RIM
See RDBMS Interface Module (RIM).

RIM host
In a Tivoli environment, the managed node from which a Tivoli administrator runs the database client software and from which the relational database management system (RDBMS) server is contacted.

S
scalable
Pertaining to the capability of a system to adapt readily to a greater or lesser intensity of use, volume, or demand. For example, a scalable system can efficiently adapt to work with larger or smaller networks performing tasks of varying complexity.

scheduler
A computer program designed to perform functions such as scheduling, initiation, and termination of jobs.
target

See endpoint.

task

In a Tivoli environment, the definition of an action that must be routinely performed on various managed nodes 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 Inventory

A Tivoli product 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.

Tivoli Management Framework

The base software that is required to run the applications in the Tivoli product suite. This software infrastructure enables the integration of systems management applications from Tivoli Systems Inc. and the Tivoli Partners. The Tivoli Management Framework includes the following:

- Object request broker (oserv)
- Distributed object database
- Basic administration functions
- Basic application services
- Basic desktop services such as the graphical user interface

In a Tivoli environment, the Tivoli Management Framework is installed on every client and server; however, the TMR server is the only server that holds the full object database.

Tivoli Management Region (TMR)

In a Tivoli environment, a Tivoli server and the set of clients that it serves. An organization can have more than one TMR. A TMR
addresses the physical connectivity of resources whereas a policy region addresses the logical organization of resources.

**TMR server**

A Tivoli server for a specific Tivoli Management Region (TMR).

**topology**

In communications, the physical or logical arrangement of nodes in a network, especially the relationships among nodes and the links between them.

**trace**

A record of the execution of a computer program. It exhibits the sequences in which the instructions were executed.

**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**.
Index

Symbols
SDBDIR directory 3-6, B-1
SDBDIR/inventory/mc directory B-1
SDBDIR/inventory/stat_dir directory 3-8, A-13
SDBDIR/mcollect directory 3-6, A-8, B-1, B-4
SDBDIR/mcollect/depot directory 3-6, B-6
SLCF_DATDIR directory B-5

A
absolute path A-3
active-collector argument A-7, A-8
argument, active-collector A-7, A-8
argument, collector-configuration A-7

B
bold typeface, use of ix
bundle
configuring destination 3-9, A-31
configuring interval at which it is sent 3-8, A-13, A-26
configuring maximum number of targets 3-9, A-14, A-27
viewing information A-19

C
collection
completed 1-3
defered 1-3, 3-11, 3-14, A-10
error conditions 1-3, B-3
failed B-9
scheduling 3-11, 3-14, A-10
using MCollect 1-4
collection manager 1-3, 3-5, A-10, B-3
collector
configuring retries 3-7, A-9
configuring the log file 3-7, A-8, A-11, B-2
configuring threads 3-7, 3-15, A-9, A-10
increasing the size of the depot 3-6, A-9, B-8
moving the depot 3-6, A-8
moving the runtime directory 3-6, A-8
resetting A-7
resetting collection routes 2-2, 3-5, A-10
scheduling collections 3-11, A-10
starting 3-5, A-7
stopping 3-5, A-7
troubleshooting B-9
using 3-5
viewing configuration information 3-15, A-7
viewing status information 3-16, A-16
collector hierarchy 1-2, 1-3, 2-1, 3-4, 3-5, 3-6
collector-configuration argument A-7
commands B-7
MCollect A-3
odadmin 2-2, 3-13
syntax A-1
tail B-2, B-3, B-4, B-5
wadminep B-7
wcollect 2-2, 3-4, 3-5, 3-8, 3-12, 3-14, 3-15, A-6, B-2, B-4, B-8, B-9
wcrtrim 2-13, 3-10
wcstat 3-16, A-11, A-16, B-2
wdel 3-10
wcollect 3-17, A-19, A-32, B-8
wcollect 3-17, 3-19, A-21, A-30, B-6, B-7
wcollect 3-17, 3-19, A-23, B-7
wcollect 2-13, 3-10, B-8
wlsinst B-8
wpatch 2-9, 2-13
wrimtrace B-5
wrpt 3-4, 3-5, A-10, B-8, B-9
configuration repository
definition 1-4
viewing Inventory data 3-18
configuring RIM objects 3-10
conventions used ix
creating an Inventory receiver object 2-14, A-13
creating RIM objects 2-13, 3-10
CTOC
role in data collection 1-5, 1-6, 3-3
viewing information about 3-16, A-16
CTOC_log.dat B-4
customer support
See Tivoli Customer Support

data units 3-6, 3-14, A-9
depot
checking contents B-6
default location 3-6, A-8
definition 1-3
increasing the size 3-6, A-9
moving 3-6, A-8
troubleshooting B-8
dialog
File Browser 2-6, 2-10
Install Patch 2-5, 2-7, 2-8, 2-10, 2-11, 2-12
Patch Install 2-9, 2-12, 2-13
directory
$DBDIR 3-6, B-1
$DBDIR/inventory/mc B-1
$DBDIR/inventory/stat_dir 3-8, A-13
$DBDIR/mcollect 3-6, A-8, B-4
$DBDIR/mcollect/depot 3-6, B-6
$LCF_DATDIR B-5
<table>
<thead>
<tr>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabling MCollect 2-14</td>
</tr>
<tr>
<td>error conditions 1-3, B-3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Browser dialog 2-6, 2-10</td>
</tr>
<tr>
<td>Framework</td>
</tr>
<tr>
<td>installing 2-4</td>
</tr>
<tr>
<td>scheduling jobs in 3-12</td>
</tr>
<tr>
<td>supported version 2-3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>hierarchy</td>
</tr>
<tr>
<td>collector 1-2, 1-3, 2-1, 3-4, 3-5, 3-6</td>
</tr>
<tr>
<td>repeater 1-2, 2-1, 3-4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install Patch dialog 2-5, 2-7, 2-8, 2-10, 2-11, 2-12</td>
</tr>
<tr>
<td>installing Framework 2-4</td>
</tr>
<tr>
<td>Inventory 2-9</td>
</tr>
<tr>
<td>Inventory Gateway 2-7, 2-11</td>
</tr>
<tr>
<td>using the command line to install MCollect 2-9</td>
</tr>
<tr>
<td>using the command line to install the Inventory receiver patch 2-9</td>
</tr>
<tr>
<td>inv_scan_nn.log B-4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inventory</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuring to return status information 3-18</td>
</tr>
<tr>
<td>installing or upgrading 2-9</td>
</tr>
<tr>
<td>Inventory Gateway 2-7, 2-11</td>
</tr>
<tr>
<td>Inventory receiver</td>
</tr>
<tr>
<td>configuring 3-8, A-13, A-26</td>
</tr>
<tr>
<td>configuring output threads 3-9</td>
</tr>
<tr>
<td>configuring retries 3-9, A-14, A-27</td>
</tr>
<tr>
<td>configuring status information 3-8, A-13, A-26, A-29</td>
</tr>
<tr>
<td>configuring the log file A-29</td>
</tr>
<tr>
<td>configuring timeout between retries 3-9, A-15, A-28</td>
</tr>
<tr>
<td>creating 2-14, A-13</td>
</tr>
<tr>
<td>definition 1-3</td>
</tr>
<tr>
<td>installing the patch 2-9</td>
</tr>
<tr>
<td>moving 3-10</td>
</tr>
<tr>
<td>using 3-8</td>
</tr>
<tr>
<td>viewing configuration information 3-17, A-19</td>
</tr>
<tr>
<td>italic typeface, use of ix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>lcfd.log B-5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCollect</td>
</tr>
<tr>
<td>components needed 1-2</td>
</tr>
<tr>
<td>enabling 2-14</td>
</tr>
<tr>
<td>features 1-6</td>
</tr>
<tr>
<td>how it works with Inventory 3-2</td>
</tr>
<tr>
<td>installation overview 2-3</td>
</tr>
<tr>
<td>installing 2-4</td>
</tr>
<tr>
<td>installing from the command line 2-9</td>
</tr>
<tr>
<td>installing from the GUI 2-4</td>
</tr>
<tr>
<td>Term</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>installing the patch</td>
</tr>
<tr>
<td>obtaining information about</td>
</tr>
<tr>
<td>mcollect.log</td>
</tr>
<tr>
<td>MDist</td>
</tr>
<tr>
<td>MIF file</td>
</tr>
<tr>
<td>monospace typeface, use of</td>
</tr>
<tr>
<td>multiplexed distribution</td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>name registry</td>
</tr>
<tr>
<td>nobody account</td>
</tr>
<tr>
<td>notification</td>
</tr>
<tr>
<td>See status information</td>
</tr>
<tr>
<td>notification, scan</td>
</tr>
<tr>
<td>O</td>
</tr>
<tr>
<td>object dispatcher ID</td>
</tr>
<tr>
<td>object paths</td>
</tr>
<tr>
<td>object references</td>
</tr>
<tr>
<td>odadmin</td>
</tr>
<tr>
<td>offlink</td>
</tr>
<tr>
<td>operating systems, supported</td>
</tr>
<tr>
<td>oserv</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>Patch Install dialog</td>
</tr>
<tr>
<td>path, absolute</td>
</tr>
<tr>
<td>path, relative</td>
</tr>
<tr>
<td>platforms, supported</td>
</tr>
<tr>
<td>prerequisite documents</td>
</tr>
<tr>
<td>product support</td>
</tr>
<tr>
<td>profile</td>
</tr>
<tr>
<td>configuring notification</td>
</tr>
<tr>
<td>configuring to use MCollect</td>
</tr>
<tr>
<td>troubleshooting</td>
</tr>
<tr>
<td>viewing information about</td>
</tr>
<tr>
<td>Q</td>
</tr>
<tr>
<td>queue</td>
</tr>
<tr>
<td>completed</td>
</tr>
<tr>
<td>deferred</td>
</tr>
<tr>
<td>description</td>
</tr>
<tr>
<td>error</td>
</tr>
<tr>
<td>input</td>
</tr>
<tr>
<td>log files</td>
</tr>
<tr>
<td>output</td>
</tr>
<tr>
<td>viewing information about</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>registered name, definition</td>
</tr>
<tr>
<td>related documents</td>
</tr>
<tr>
<td>relative path</td>
</tr>
<tr>
<td>release notes</td>
</tr>
<tr>
<td>repeater hierarchy</td>
</tr>
<tr>
<td>repeater site</td>
</tr>
<tr>
<td>resetting a collector</td>
</tr>
<tr>
<td>resetting collection routes</td>
</tr>
<tr>
<td>retry</td>
</tr>
<tr>
<td>RIM object</td>
</tr>
<tr>
<td>configuring</td>
</tr>
<tr>
<td>creating</td>
</tr>
<tr>
<td>definition</td>
</tr>
<tr>
<td>rim_db_log</td>
</tr>
</tbody>
</table>

**Index–4**

Version 3.6.2
runtime directory
  default location 3-6, A-8
  moving 3-6, A-8

S
scalable collection 1-1
scan notification 3-9
  See status information
  scans
troubleshooting B-6, B-7
  viewing information about 3-17, A-21
scheduler daemon 1-3
scheduling collections 3-11, A-10
size of data units 3-6, 3-14, A-9
starting a collector 3-5, A-7
status collector
  commands used with  A-5
  configuring options 3-8, A-13, A-26, A-29
  definition 1-3
  how it works 1-6
  viewing options 3-15, 3-17, 3-19, A-23
status information
  configuring 3-8, 3-18, A-13, A-26, A-29
  configuring the log file A-29
  definition 3-8
  scan notification 3-9
  viewing collector status 3-16, A-16
  viewing scan status A-19
stopping a collector 3-5, A-7
supported platforms ix
text conventions ix
threads
  configuring A-9, A-10
  threads, configuring 3-7, 3-15, A-9
Tivoli Customer Support, contact
  information x
Tivoli desktop 2-5, 2-10
Tivoli Management Framework
  See Framework
Tivoli scheduler 3-12, 3-14
tmersrvd account 3-6, A-8
TMR
  considerations for connected TMRs 3-19
  number of Inventory receivers per 2-14, A-13
  resources of the same name and type within a TMR A-3
transmission chunk 3-6, 3-14, A-9
typefaces, use of ix

W
wadminep B-7
WAN 2-2, 3-19
wcollect 2-2, 3-4, 3-5, 3-8, 3-12, 3-14, 3-15, A-6, B-2, B-4, B-8, B-9
wcrtrim 2-13, 3-10
cstat 3-16, A-11, A-16, B-2
cdel 3-10
wgetinvrcvr 3-17, A-19, A-32, B-8
wgetinvstat 3-17, 3-19, A-21, A-30, B-6, B-7
wgetpcoll 3-17, 3-19, A-23, B-7
wgettrim 2-13, 3-10, B-8
wlsinst B-8
wpatch  2-9,  2-13
wrimtest  B-7
wrimtrace  B-5
wrpt    3-4,  3-5,  A-10,  B-8,  B-9