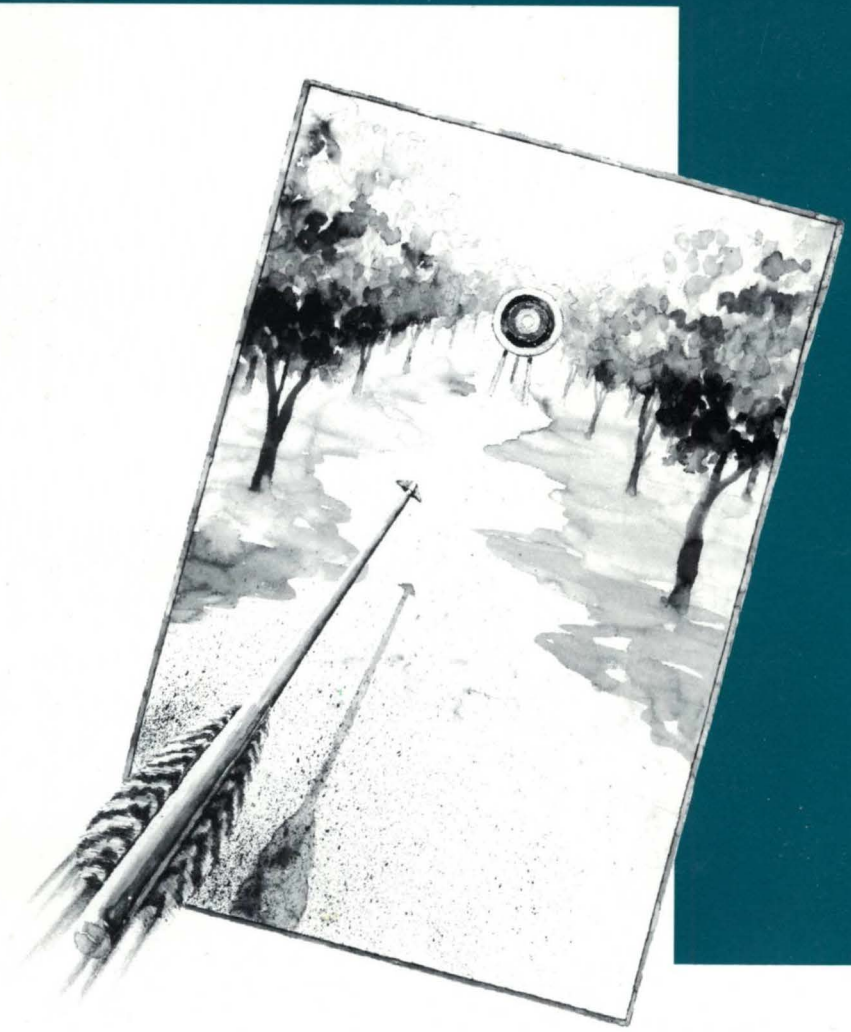
HEWLETT PACKARD



MeasureWare Agent: User's Manual

MeasureWare Agent: User's Manual

HP Part No. B4967-90001
Printed in USA 1195

E1195

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

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

Lotus®1-2-3® is a United States registered trademark of Lotus Development Corporation.

Hewlett-Packard Company
Support Technology Center
8000 Foothills Boulevard
Roseville, CA 95747-5726, U.S.A.

Printing History

New editions are complete revisions of the manual. The dates on the title page change only when a new edition is published.

A software code may be printed before the date; this indicates the version level of the software product at the time the manual was issued.

Edition 1 11/95 B4967-90001 E1195

About This Manual

This manual is designed to help you understand MeasureWare Agent and to support you while you use it.

Chapter 1 provides an overview of MeasureWare Agent and its features.

Chapter 2 tells you how to manage your data collection processes. It includes descriptions of the **scopeux** collector program and the `parm` file, as well as suggestions and tips for managing performance data. Read this chapter before you have logged more than one month's worth of data.

Chapter 3 describes the **utility** program and its commands and functions.

Chapter 4 describes the **extract** program and its commands and functions. It also describes how to export data for reporting and other uses.

Chapter 5 describes performance alarms. It explains the syntax for creating alarm definitions and provides examples of how alarms can be used to monitor performance.

Appendix A describes **utility** scan reports.

The Glossary provides an alphabetic list of terms used in this manual.

Other Documentation

Additional documentation for MeasureWare Agent consists of:

- *MeasureWare Agent: Installation and Configuration Guide*
- *MeasureWare Agent: Data Source Integration Guide*
- *MeasureWare Agent: Transaction Tracker Handbook*
- Online help for the **utility** program
- Online help for the **extract** progran
- Online **man** pages.

The following documents are in printable files that are shipped with MeasureWare Agent. For printing instructions, see Chapter 4 in your *MeasureWare Agent: Installation and Configuration Guide*.

- *MeasureWare Agent Metrics Definitions*
- *MeasureWare Agent Metrics Dictionary*
- *MeasureWare Network Response Facility User's Guide*
- *MeasureWare Network Response Facility Best Practices White Paper*

Conventions

boldface	Words in boldface represent the names of programs and commands.
`computer font`	Words in computer font represent file names, syntax, or text as you should enter it on your workstation or terminal, and text that appears on the screen.
Italics	Italics are used to emphasize words, phrases, or characters in the text or indicate variables in syntax strings.
Return *or* Enter	Depending on your keyboard, one or the other represents the key used to execute a command.

Contents

Figures

Tables

1

Overview

This chapter is an introductory overview of MeasureWare Agent its components, and related products. It discusses:

- what MeasureWare Agent does
- data sources
- the **scopeux collector**
- the **parm** file
- repository servers
- **utility** and **extract** programs
- related performance products

What MeasureWare Agent Does

MeasureWare Agent collects, summarizes, time stamps, and detects alarm conditions on current and historical resource data across the system. It replaces and enhances the functionality previously provided by Performance Collection Software. MeasureWare Agent provides end-to-end transaction response time measurements, supports database measurement information, and provides data and alarm information to PerfView for analysis.

Data collected outside MeasureWare Agent can be integrated using data source integration (DSI) capabilities. For example, network, database, and

application data can be brought in through DSI and is treated the same as data collected by MeasureWare Agent. All DSI data is logged, time stamped, and can be alarmed on. (For details, see the *MeasureWare Agent: Data Source Integration Guide*.)

All of the data collected or received by MeasureWare Agent can be analyzed using spreadsheet programs, Hewlett-Packard analysis tools such as PerfView, or third-party analysis products.

The comprehensive data logged by MeasureWare Agent allows you to:

- Characterize the workloads in the environment.
- Analyze resource usage and load balance.
- Perform trend analyses to isolate and identify bottlenecks.
- Perform service-level management based on transaction response time.
- Perform capacity planning.
- Respond to alarm conditions.
- Solve system management problems before they arise.

MeasureWare Agent gathers comprehensive and continuous information on system activity without imposing significant overhead on the system.

Its design offers considerable opportunity for customization. You can accept default configurations or set parameters to collect data about specific conditions.

The Agent Products

MeasureWare Agent software consists of two agent products:

- MeasureWare Server Agent
- MeasureWare Desktop Agent

MeasureWare Desktop Agent is the same as MeasureWare Server Agent *except* it does not support application and process data collection.

Note

The name MeasureWare Agent is used as a generic term throughout this manual when discussing topics that apply to both agent products. However, when discussing topics that apply only to a specific agent product, the name of that agent product is used. For example, the `application` and `process` log files are used only with MeasureWare Server Agent.

Components

The following diagram shows the relationships among components of the MeasureWare Agent system.

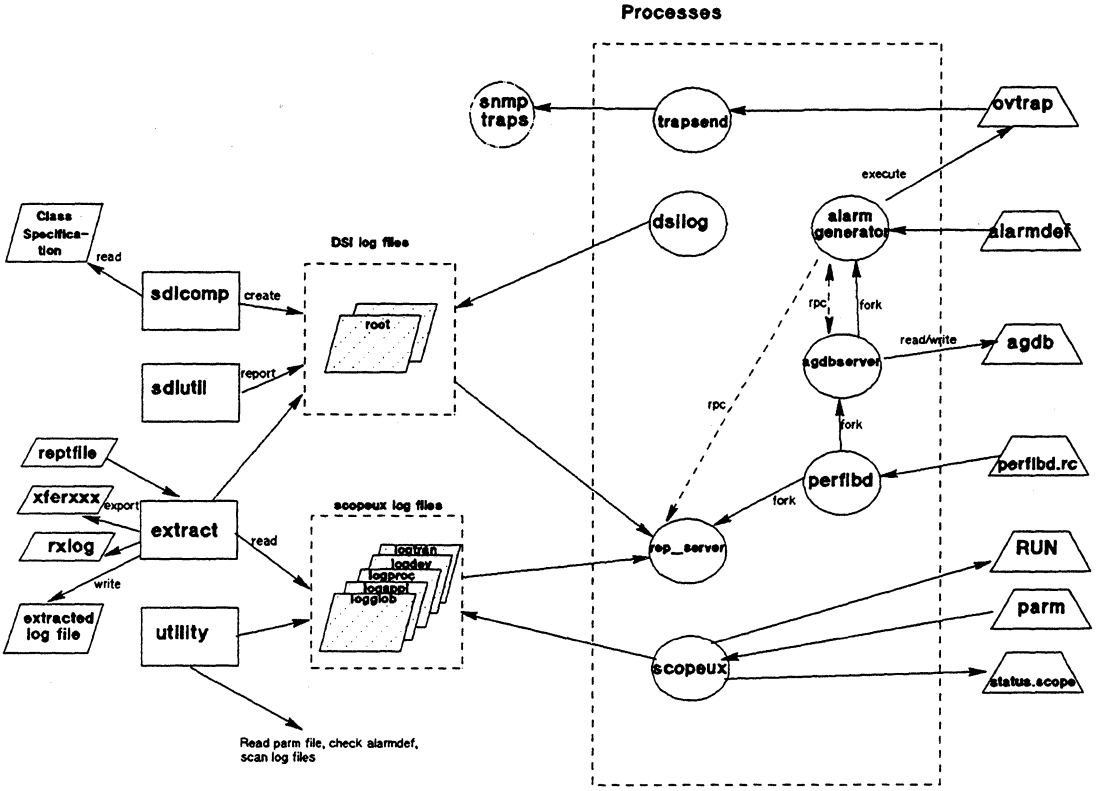


Figure 1-1. MeasureWare Agent Components

Scopeux Collector

The **scopeux** collector program collects performance data from the operating system kernel in which MeasureWare Agent resides, summarizes it, and logs it into five raw log files, depending on the types of information desired.

Log Files

The following raw log files are created by the **scopeux** collector.

- The `logglob` file holds measurements of system-wide, or global, information.
- The `logappl` file includes summary measurements of the processes in each user-defined application.
- The `logproc` file contains measurements of selected interesting processes.
- The `logdev` file contains measurements of individual device performance.
- The `logtran` file contains measurements of Transaction Tracker transactions data.

A sixth log file, `logindx`, is created when information is logged into the other log files. It contains additional information needed to access data in the other log files. It will not grow appreciably in size over time.

The Parm File

The type of data collected is determined by parameters set in the MeasureWare Agent programs and in the `parm` file, a standard ASCII file used to customize the collection environment. The `parm` file contains instructions that tell **scopeux** to log specific performance measurements.

DSI Log Files

DSI log files contain self-describing data that is collected outside of MeasureWare Agent. These log files are created by MeasureWare Agent's DSI programs. DSI processes and the creation of DSI log files are described in detail in the *MeasureWare Agent: Data Source Integration Guide*.

Utility and Extract Programs

Two MeasureWare Agent programs, **utility** and **extract**, provide the means for managing both **scopeux** log files and DSI log files.

The **utility** program lets you open and generate reports on raw and extracted **scopeux** log files, resize raw **scopeux** log files, and check `parm` file syntax. It also lets you check the syntax in your alarm definitions file and analyze alarm conditions in historical **scopeux** and DSI log file data. (Extracted log files contain subsets of the information included in raw log files.)

The **extract** program lets you extract data from raw or previously extracted **scopeux** log files, summarize it, and write it to extracted log files. The extracted log files contain selected performance data for specific analysis needs. The **extract** program also lets you export **scopeux** and DSI data for use by analysis products such as PerfView.

Data Sources

MeasureWare Agent uses a set of repository servers that provide data to the alarm generator and the PerfView analysis product. There is one repository server for each data source consisting of a **scopeux** or DSI log file set.

You configure data sources in a file called `perflbd.rc` that resides in your *datafiles* or *configuration* directory. This allows you to selectively make data available for alarm processing and/or analysis. When you first start MeasureWare Agent, the `perflbd` daemon reads the `perflbd.rc` file and starts a repository server for each data source that has been configured. The default

`perfldb.rc` file contains one entry for a data source named SCOPE that starts a repository server for a **scopeux** log file set.

Note

Throughout this manual, the names *configuration* and *datafiles* are used to indicate platform-specific directories in MeasureWare Agent. See the **man** page for **scopeux** to see which of these directories is applicable to your MeasureWare Agent system.

For more information on `perflbd.rc` and instructions for configuring data sources, see the "Starting Up MeasureWare Agent" chapter in the *MeasureWare Agent: Installation and Configuration Guide*.

Transaction Tracker

MeasureWare Agent includes Transaction Tracker technology that allows information technology (IT) managers to measure end-to-end response time of business application transactions. For details, see the *MeasureWare Agent: Transaction Tracker Handbook*.

Network Response Facility

MeasureWare Agent, working with HP NetMetrix Network Response Agent software, adds critical LAN/WAN response time measurement data to its set of resource and performance measurement data. The response time measurements, from a variety of target nodes, are provided by the HP NetMetrix Network Response Agent to be logged by MeasureWare Agent. The response times can be alarmed on along with other MeasureWare Agent metrics, and be made available to PerfView for analysis, alarm monitoring, comparing, and forecasting.

Related Performance Products

MeasureWare Agent is one of four compatible performance products from Hewlett-Packard. Each of these related products fulfills a particular need within the range of performance management. This lets you purchase as much functionality as you need and add products over time without risking incompatibilities or overlapping product capabilities.

Related HP performance products include the following:

- **PerfView**

  PerfView provides integrated performance management for multi-vendor distributed networks. It gives you a single interface and a common method for centrally monitoring, analyzing, comparing, and graphically forecasting resource measurement data supplied by MeasureWare Agent. PerfView also displays and analyzes alarm information sent by MeasureWare Agent.

- **GlancePlus**

  GlancePlus is an online diagnostic tool that displays current performance data directly to a user terminal or workstation. It is designed to assist you in identifying and troubleshooting system performance problems as they occur.

- **MeasureWare Agent Database Modules**

  MeasureWare Database Modules are an optional data source product that provides metrics on Oracle, Sybase, and Informix databases to MeasureWare Agent. Once these metrics are received by MeasureWare Agent, they can be logged, time stamped, exported, and alarmed on. For more information, see your Hewlett-Packard sales representative.

2

Managing Scopeux Data Collection

This chapter tells you how to manage the following **scopeux** data collection activities that are involved in using MeasureWare Agent.

- using the **scopeux** collector
- defining and modifying the `parm` file and its parameters
- stopping and restarting data collection
- effective strategies for data collection management
  - rolling back log files periodically
  - rolling back log files automatically
  - archiving data
  - system analysis tasks

The Scopeux Collector

The MeasureWare Agent data collection environment is controlled by the **scopeux** collector program.

scopeux collects and summarizes performance measurements of system-resource utilization and logs the data into the following log files, depending on the types of information desired.

- The `logglob` file contains measurements of system-wide, or global, resource utilization information. Global data is summarized and recorded every five minutes.
- The `logappl` file contains aggregate measurements of processes in each user-defined application. Application data is summarized every five minutes and each application that had any activity during the five minute interval is recorded.
- The `logproc` file contains measurements of selected "interesting" processes. Process data is summarized every minute. However, only interesting processes are recorded. The concept of interesting processes is a filter that helps minimize the volume of data logged for "non-interesting" processes.

  A process becomes interesting when it is first created, when it ends, and when it exceeds user-defined thresholds for CPU use, disk use, response time, or transaction rate.
- The `logdev` file contains measurements of individual device (disk, logical volume, or netif) performance. Device data is summarized every five minutes and each device that had any activity is recorded.
- The `logtran` file contains measurements of Transaction Tracker data. This data is summarized every five minutes and each transaction that had any activity is recorded. (For more information, see the *Transaction Tracker Handbook*.)

- A sixth log file, `logindx`, is created when information is logged into the other log files. It contains additional information needed to access data in the other log files. It does not grow appreciably over time.

Note

`logappl` and `logproc` are supported in MeasureWare Server Agent only.

Other Files Used by Scopeux

In addition to the log files, two other files are created when **scopeux** is started. They are the `RUN` file and the `status.scope` file.

The `RUN` file is created to indicate that the **scopeux** process is running. Removing this file causes the **scopeux** program to terminate.

The `status.scope` file serves as a status/error log for the **scopeux** process. New information is appended to this file each time the **scopeux** collector is started. You can use the **tail** command to see the most recent information (for example, `tail -30 status.scope`).

Note

You should view the `status.scope` file on a regular basis to ensure that the collection process is running correctly.

The `status.scope` file has a maximum size of 100 Kbytes. When that size is reached, the file is removed and then recreated.

Note

Throughout this manual, the names *datafiles* and *configuration* are used to indicate platform-specific directories in MeasureWare Agent. See the **man** page for **scopeux** to see which of these directories is applicable to your MeasureWare Agent system.

Parm File Parameters

scopeux is controlled by specific parameters in the **parm** file that resides in the *datafiles* or *configuration* directory. You can modify these parameters to tell **scopeux** to log measurements that match the requirements of your particular system. The **parm** file parameters used by **scopeux** are listed in Table 2-1 on the next page.

Table 2-1.
Parm File Parameters Used by Scopeux

Parameter	Values or Options
id=	*system ID*
log=	**global** **application** (MeasureWare Server Agent only) **process** (MeasureWare Server Agent only) **device=disk, logical volume, vol,** or **lvm** **transaction**
threshold=	**cpu=** *percent* **disk=** *rate* **first=** (HP-UX only) **resp=** (HP-UX only) **trans=** (HP-UX only) **nonew=** **nokilled=** **shortlived=** (HP-UX only)
wait=	**cpu=** (HP-UX only) **disk=** (HP-UX only) **mem=** (HP-UX only) **sem=** (HP-UX only) **lan=** (HP-UX only)
application	*application name* (MeasureWare Server Agent only)
file	*file name* [, ...]
user	*user login name* [, ...]
group	*groupname* [, ...]
or	
priority	*low value-high value* (range is -511–255, depending on your system)
size=	**global=***nn* (values are in megabytes) **application=***nn* (MeasureWare Server Agent only) **process=***nn* (MeasureWare Server Agent only) **device=** *nn* **transaction=** *nn*
mainttime	*hh:mm* **(24 hour time)**

Note

The items in Table 2-1 that are applicable only to HP-UX are described in detail in the *MeasureWare Agent: Installation and Configuration Guide (for HP-UX).*

The `nonew`, `nokilled` and `shortlived` threshold options act as `on/off` flags.

If **scopeux** finds `nonew` or `nokilled` in the thresholds in the `parm` file, it does not log new or killed processes that are otherwise uninteresting. If `nonew` or `nokilled` are not found, **scopeux** logs new and killed processes.

Default Values

When **scopeux** starts, it looks for the `parm` file in the *configuration* directory. If it does not find the `parm` file or if a specific parameter is not included in the `parm` file, **scopeux** uses its default options. Table 2-2 on the next page lists the default values for each parameter.

Table 2-2. Scopeux Default Values

Parameter	Default Value
id	*nodename* from *uname -n*
log	global, process (process is MeasureWare Server Agent only)
threshold	cpu=10.0 disk=10.0 IO/sec first=5.0 sec/tran (HP-UX only) resp=30.0 sec/tran" (HP-UX only) trans=100 trans/interval (HP-UX only) nonew=false nokilled=false shortlived=true (HP-UX only)
wait	cpu wait=100.00%(HP-UX only) disk wait=100.00%(HP-UX only) mem wait=100.00% (HP-UX only) sem wait=100.00% (HP-UX only) lan wait=100.00% (HP-UX only)
application file user pri group	Applicable only when applications are specified (MeasureWare Server Agent only).
size	global=10 application=10 (MeasureWare Server Agent only) process=20 (MeasureWare Server Agent only) device=10 transaction=10
mainttime	23:30

Sample Parm File

A sample **parm** file is provided with MeasureWare Agent. It is placed in the *newconfig* directory and conditionally copied into the *datafiles* or *configuration* directory during installation. **scopeux** reads the **parm** file when it starts up.

If you haven't run the product before, you can use the sample **parm** file to become familiar with the type of data collected. Once you are familiar with the MeasureWare Agent environment, you should tailor the **parm** file to your performance data collection needs.

The sample **parm** file is set up to collect an average amount of log file data. The maximum amount is dependent on your system. For more information regarding log file space, please see "Disk Space Required for Running MeasureWare Agent" in the "Installation" chapter of your *MeasureWare Agent: Installation and Configuration Guide*.

If you already have experience with Performance Collection Software or MeasureWare Agent and are familiar with the **parm** file parameters, you might want to modify this file before starting the **scopeux** collector. See "The Parm File" section later in this chapter for a discussion of **parm** file parameters and how to modify them.

Starting the Scopeux Collector

You use the **mwa** script to start **scopeux**. For more information, see the "Starting Up MeasureWare Agent" chapter in your *MeasureWare Agent: Installation and Configuration Guide*.

Note

After **scopeux** has been working for a day or so, you may want to see a sample of collected data. To do so, run the demo script **mwademo**.

The Parm File

The `parm` file is an ASCII text file shared by both MeasureWare Agent and GlancePlus. MeasureWare Agent uses all of the `parm` file parameters; GlancePlus uses only the `parm` file parameters that pertain to applications.

The `parm` file contains the instructions that tell **scopeux** to log specific performance measurements. You can modify the `parm` file to tell **scopeux** to log measurements that match the requirements of your particular system.

Modifying the Parm File

You can use almost any text editor to create or modify the `parm` file. The result must be a standard ASCII file.

When you create or modify the `parm` file, the following rules and conventions apply:

- You need only specify a parameter if you want to override a default.
- The order in which the parameters are entered into the `parm` file is not important except as follows:
  - If a parameter is entered more than once, the last one entered is used.
  - The `file`, `group`, `user`, `or`, and `priority` statements must follow the `application` statement that they define.
- You can use uppercase letters, lowercase letters, or a combination of both for all commands and parameter statements.
- You can use blanks or commas to separate key words in each statement.
- You can comment the `parm` file. Any line starting with a comment code (`/*`) or pound sign (`#`) is ignored.

Note

After modifying the `parm` file, you *must* issue the **`mwa restart scope`** command in order for the changes to take effect. This action causes **scopeux** to stop, restart, and reread the `parm` file.

Parameters Used Exclusively by MeasureWare Agent

The following `parm` file parameters are used by MeasureWare Agent exclusively:

- `ID`
- `log`
- `threshold`
- `size`
- `mainttime`

ID

The system ID value is a string of characters that identifies your system. If you have multiple systems, use different ID strings on each one. This identifier is carried with the log files to identify the system on which the data was collected.

You can specify a maximum of 40 characters.

The default ID is the **nodename** as returned by `uname -n`.

Log

Specifying `global` in the `log` parameter causes global records to be written to the `logglob` log file.

Caution

You must log global data in all cases. You must have global data records to view and analyze performance data on your system.

Specifying `application` in the `log` parameter causes application records to be written to the `logappl` log file.

Specifying `process` in the `log` parameter causes "interesting" processes to be written to the `logproc` log file.

Note

The `log` parameter in the `parm` file supports `application` and `process` log files *only* in MeasureWare Server Agent.

Specifying `device=disk` in the `log` parameter causes individual disk records to be written to the `logdev` file.

Specifying `device=vol` (or `device=lvm`) in the `log` parameter causes individual logical volume records to be written to the `logdev` file.

Specifying `tran` in the `log` parameter causes Transaction Tracker records to be written to the `logtran` log file. To collect the data, MeasureWare Agent must have a process running that is configured with the Transaction Tracker API. (For more information, see the *MeasureWare Agent: Transaction Tracker Handbook*.)

All of the log files are created automatically if logging to them is specified and they do not already exist. If a particular type of logging is disabled, the corresponding log file is not removed.

If you specify `log` without options, the default global and process data are logged.

Threshold

You can change "interesting" process thresholds by changing their values.

You can enter the options for thresholds on the same parameter line (separated by commas) or on separate

lines. You must include the word 'threshold' on each line.

CPU Option. This option sets the percentage of CPU utilization that a process must exceed to become "interesting" and be logged. It is used only if process logging is enabled.

The value "percent" is a real number indicating overall CPU use. For example, `cpu=7.5` indicates that a process is logged if it exceeds 7.5 percent of CPU utilization in a 1-minute sample.

The default is `cpu=10.0`.

Disk Option. This option sets the rate of major page faults or block I/Os that a process must exceed to become "interesting" and be logged. The value "rate" is the disk I/O rate in transfers per second and is a real number. For example, `disk=8.0` indicates that a process will be logged if it exceeds either eight major page faults or eight block I/Os per second in a 1-minute sample.

The default is `disk=10.0`.

Nonew Option. This option turns off logging of new processes if they have not exceeded any threshold.

The default is `nonew=false`.

Nokilled Option. This option turns off logging of exited processes if they did not exceed any threshold.

The default is `nokilled=false`.

Note

For platform-specific threshold options that may be applicable to your system, see your *MeasureWare Agent: Installation and Configuration Guide*.

Size

The `size` parameter is used to set the maximum size (in megabytes) of any raw log file. By default, `logglob` (the

global file), `logappl` (the application file), and `logdev` (the device file), and `logtran`, (the transaction file) have sizes of 10 megabytes each, while `logproc` (the process file) has a size of 20 megabytes. You cannot set the size to be less than one megabyte.

The maximum size allowed is `INT_MAX/100000` as defined in the `limits include` file on your operating system. This maximum number can vary among platforms.

Note

`logappl` and `logproc` are supported only in MeasureWare Server Agent.

The **scopeux** collector reads these specifications when it is initiated from `mwa` or `mwa start scope`. If any of these log files achieve their maximum size during collection, they will be rolled back automatically. During the roll back, the oldest 25 percent of the data is removed from the log file.

If the `size` specification in the `parm` file is changed, **scopeux** detects it during startup and takes appropriate action. If the maximum log file size is decreased to the point where existing data does not fit, an automatic resize takes place. If the existing data fits within the new maximum size specified, no action is taken.

Mainttime

Log files are rolled back by **scopeux** only at a specific time each day. The time is normally 23:30 p.m. but it can be changed using the `mainttime` parameter.

Every five minutes **scopeux** checks available disk space for the file system in which the log files reside. If less than 1 megabyte of space is available to non-`root` users, it performs any log file roll backs immediately. If there is still less than one megabyte of free space, **scopeux** terminates with an appropriate message in the `status.scope` file.

Application Definition Parameters

The following parameters are used by both MeasureWare Agent and GlancePlus: `application`, `file`, `user`, `group`, `or`, and `priority`. These parameters all pertain to defining applications.

Note

These parameters in the `parm` file are supported only in MeasureWare Server Agent.

You can group logically related processes together into an application to log the combined affect of those processes on computing resources such as disk, memory, and CPU.

Any process on the system belongs to one application only. Processes are based on name, not program path. Therefore, two processes with the same name but different paths (file system location), are considered to be the same process.

An application can simply be a list of files combined with a list of `users`, `files`, and `groups` with a `priority` range, or all of these in conjunction with each other. If `user`, `file`, and `priority` parameters are all specified for the same application, a process must meet the specifications of all three—`user`, `file`, and `priority`—to belong to that application.

You can have up to 300 file, user, and group specifications for all applications combined.

Application

The application name defines an application or class that groups together multiple processes and reports on their combined activities. It is a string of up to 19 characters used to identify the application. Application names cannot contain embedded blanks. If you have an existing application name containing an embedded blank, you must replace the blank with an underscore

(_). For example, `application 1` must be changed to `application_1`.

`application=`*`application_name`*

The `application` parameter must precede any combination of `file`, `user`, `group`, `or`, or `priority` parameters that refer to it, with all such parameters applying against the last `application` workload definition.

Each parameter can be up to 170 characters long including the carriage return, with no continuation characters permitted. If your list of files, users, or groups is longer than 170 characters, continue the list on the next line after another `file`, `user`, or `group` statement.

If `user`, `file`, `group`, and `priority` parameters are specified for the same application, a process must match one of the file name type parameters. The user login must match one of the `user` and `group` parameters and fall within the specified priority range in order to belong to a particular application. A process cannot belong to a particular application if it fails to match any of the four parameters.

You can define up to 31 applications. MeasureWare Agent predefines an application called `other`. The `other` application collects all processes not defined by `application` statements in the `parm` file.

For example:

```
application=Prog_Dev
file=vi,cc,ccom,pc,pascomp,dbx,xdb

application=xyz
file=xyz*,startxyz
```

You can have a maximum of 300 `file`, 300 `user`, and 300 `group` specifications for all applications combined. The previous example includes nine `file` specifications.

(**xyz*** counts as only one specification even though it can match more than one program file.)

If a program file is included in more than one application, it is logged in the first application that contains it.

By default, no user applications are defined.

Note

Any process on the system belongs to only one application. No process is counted into two or more applications. Processes are collected based on program name, not program path. Therefore, two processes with the same program name, but different paths (file system locations) are considered to be the same process by the **scopeux** collector.

File

The `file` parameter specifies which program files belong to an application. All interactive or background executions of these programs are included. It applies to the last `application` statement issued. An error is generated if no `application` statement is found.

The *file name* can be any of the following:

- A single UNIX program file such as `vi`.
- A group of UNIX program files (indicated with a wildcard) such as **xyz***.

  In this case, any program name that starts with the letters **xyz** is included. A file specification with wildcards counts as only one specification toward the maximum of 300 each for all `file`, `user`, and `group` specifications.

The name in the `file` parameter is limited to 15 characters in length.

You can enter multiple file names on the same parameter line (separated by commas) or in separate `file`

statements. File names cannot be qualified by a path name.

For example:

```
application=payroll
file=account1,basepay,endreport
application=Prog_Dev
file=vi,cc,ccom,pc
file=pascomp,xdb
```

If you do not specify a `file` parameter, all programs that satisfy the other parameters qualify.

User

The `user` parameter specifies which user login names belong to an application.

For example:

```
application=Prog_Dev_Group2
file=vi,xb,abb,ld,lint
user=ted,rebecca,test*
```

User specifications that include wildcards count as only one specification toward the maximum of 300 each for all `file`, `user`, and `group` specifications.

If you do not specify a `user` parameter, all programs that satisfy the other parameters qualify.

The name in the `user` parameter is limited to 15 characters in length.

Group

The `group` parameter specifies which user group names belong to an application.

For example:

```
application=Prog_Dev_Group2
file=vi,xb,abb,ld,lint
```

```
user=ted,rebecca,test*
group=lab,test
```

If you do not specify a group, all programs that satisfy the other parameters qualify.

The name in the group parameter is limited to 15 characters in length.

Or

Use the or parameter to allow more than one application definition to apply to the same application. Within a single application definition, a process must match at least one of each category of parameters. Parameters separated by the or parameter are treated as independent definitions. If a process matches the conditions for any definition, it will belong to the application. For example,

```
application=Prog_Dev_Group2
user=fred
or
user=ted
file=vi,store,dmp
```

This defines the application (Prog_Dev_Group2) that consists of any programs run by the user fred plus other programs (vi,store,dmp) *if* they are executed by the user ted.

Priority

You can restrict processes in an application to those belonging to a specified range by specifying values in the priority parameter. For example:

```
application=swapping
priority=128-131
```

This gathers all processes running at PSWP priority into one application.

Processes can range in priority from -511 to 255, depending on which platform MeasureWare Agent is running. See your *MeasureWare Agent: Installation and Configuration Guide* for the exact priority range for your system.

The priority can be changed over a process' life. The scheduler adjusts the priority of time-share processes. You can also change priorities programmatically or while executing. See your *MeasureWare Agent Installation and Configuration Guide* for information about the commands you can use to change priorities on your system.

The process priority is sampled at the end of each one-minute sample interval. If the process has changed priority, it can change applications. All activity for a process during the one-minute interval is assumed to have occurred at the new priority and is attributed to the application that matches the process at the end of each one-minute sample interval.

The `parm` file is processed in the order entered and the first match of program name and/or user `login` as well as `group` and `priority` (if used), defines the application to which a particular process belongs.

Application Definition Examples

The following list shows application definition examples.

```
application=Prog_Dev_Group1
file=vi,dbx,abb,ld,lint
user=bill,debbie

application=Prog_Dev_Group2
file=vi,dbx,abb,ld,lint
user=ted,rebecca,test*
group=labmqa

application=Other_Editors
file=ed,sed,awk

application=Compilers
file=cc,ccom,xlc,c++fe

application=Users
user=nelson,ted,rebecca,gene
```

The following is an example of how several of the programs would be logged using the preceding **parm** file.

Program	User login	Priority	Application
vi	bill	60	Prog_Dev_Group1
awk	dan	60	Other_Editors
vi	test5	60	Prog_Dev_Group2
sh	gene	60	Users
cc	gary	60	Compilers
dbx	dan	60	Other

Stopping and Restarting Data Collection

The **scopeux** collector is designed to run continuously. The only time you should stop it is when any of the following occurs:

- You are updating the MeasureWare Agent software to a new release.
- You are changing the **parm** file and want the changes to take effect. Changes made to the **parm** file take effect only when **scopeux** is started.
- You are using **utility** to change the size of a MeasureWare Agent log file.

The collection process also stops if a system failure occurs or if **scopeux** aborts.

Stopping Data Collection

To manually stop **scopeux** when you want to stop data collection, use `mwa stop scope`. Using this script ensures that no data is lost when **scopeux** stops.

Restarting Data Collection

You have different options for restarting data collection after the system has been down, if **scopeux** has been terminated, or if **scopeux** is running and you just want to restart it:

- To start **scopeux** after the system has been down, use the `mwa` script, which starts up MeasureWare Agent and all of its processes.
- To restart **scopeux** after it has been terminated but MeasureWare Agent is still running, use `mwa start scope`.
- To restart **scopeux** while it is running, use `mwa restart scope`, which stops the currently running **scopeux** and then starts up a new **scopeux**. `mwa restart scope` is the equivalent of performing a `mwa stop scope` followed by a `mwa start scope`.

- You would also use `mwa restart scope` to restart **scopeux** after making changes to the `parm` file.

When you restart **scopeux**, MeasureWare Agent continues to use the same log files (`logglob`, `logappl`, `logproc`, `logdev`, `logtran`, and `logindx`) used before stopping the program. New records are appended to the end of the existing files.

If you want to collect data to a new set of files, it is critical that you rename or archive and remove *all* log files - `logglob`, `logappl`, `logproc`, `logdev`, `logtran`, and `logindx` - before you restart **scopeux**, because data is synchronized among the files.

Automating Scopeux Start Up and Shut Down

The start up and shut down processes can be automated to ensure that **scopeux** is always running when the system is operating and that any shutdown of the system includes a shutdown of **scopeux** without any loss of data.

Start Up

The **scopeux** collection process was designed to provide you with continuous collection of performance information. In order to do that, **scopeux** must be running whenever the system is up.

For this reason, we recommend that you include the following as part of your system start up script that starts **scopeux** each time your system boots.

```
# Start-up the MeasureWare Agent collection process directory path/mwa start
```

Shut Down

You can also automate a graceful shut down of the **scopeux** collection process before you reboot or halt your system. A data collection shut down command, `mwa stop` is provided with MeasureWare Agent. By invoking `mwa stop` during your shut down procedure, you ensure that no data is lost as a result of system reboot or halt.

For this reason we recommend that you include the following as part of your shut down procedure script:

```
# Shut down the MeasureWare Agent collection process directory path/mwa stop
```

Effective Strategies for Data Collection Management

Efficient analysis of performance depends on how easy it is to access the performance data you collect. Discussed here are effective strategies for activities such as managing log files, data archiving, and system analysis to make the data collection process easier, more effective, and more useful.

Rolling Back Log Files Periodically

You can resize files by rolling them back. Roll back means the deletion of one or more days' worth of data from a log file, deleting oldest data first. Roll backs are done when a raw log file exceeds its maximum size parameter.

Note

`logappl` and `logproc` log files are supported in MeasureWare Server Agent only.

You can roll log files back automatically or manually with the **utility** program's **resize** command. If you use **utility** to roll back the log files manually, you need 75 percent of the current log file space in the *datafiles* or *configuration* directory and 100 percent in `/tmp`. This is because **utility** creates a temporary copy in `/tmp`.

Note

You may want to prevent **scopeux** from rolling the log files automatically because **scopeux** stops collecting data for as long as it takes to perform the roll back. To do this, use the `size` parameter in the `parm` file to increase your raw log file size. (See the `size` parameter description in this chapter for information about setting log file sizes).

Rolling Back Log Files Automatically

This section gives you a recommended procedure for rolling back the log files, automatically, once a week.

First, create a file called `autoutil` that contains the commands to **utility** shown here. Redirect `stdin` to this file when you run **utility**.

```
detail off
scan
resize global      empty=10 maybe
resize application empty=10 maybe

resize process     empty=10 maybe
resize disk        empty=10 maybe
resize transaction empty=10 maybe
quit
```

Save this file as `autoutil` and set up `cron` to do the following steps once a week:

1. Stop **scopeux** using `mwa stop scope`.
2. Back up log files.
3. Use the command:

   ```
   utility < autoutil >> util.rept 2>> util.rept
   ```

   to redirect `stdin` to `autoutil`.
4. Restart **scopeux** using `mwa start scope`.

This procedure ensures that you have enough space to collect the next week's data, provided that the amount of data collected per day does not increase unexpectedly (such as an increase in system workload or a change in **scopeux** parameters).

Data Archiving

To archive collection data, the following tasks are recommended:

- Size raw log files to hold the desired amount of data. **scopeux** automatically rolls data out of them whenever they reach their maximum sizes.
- Use the following **extract** commands each month to extract detail data monthly. You can then store the data on tape. See Chapter 4 for descriptions of the **extract** program and its commands.

```
>extract
global both
application both
process detail
disk both
transaction both
lvolume both
netif both
monthly
quit
```

or

```
>extract -gGaApdDnNzZ -C transaction both -xm
```

If monthly extracted files are too large, do a weekly extraction instead.

This detailed extraction preserves all of your collected performance data. If ever you need to investigate a situation in depth, these files can be restored to disk and analyzed.

- Use the following **extract** commands to summarize data weekly, appending it to the same summary file.

```
>extract
global summary
application summary
process off
disk summary
netif summary
lvolume summary
transaction summary
weekly
quit
```

or

```
>extract -GADNZ -C transaction summary -xw
```

It is a good idea to extract and keep data for a long time in the event that you want to show trends or do forecasting.

Hint

You can use the **extract** program to combine data from multiple extracted files or to make a subset of the data for easier transport and analysis.

- For example, you can combine data from several yearly extracted files in order to do multiple-year trending analysis.

  or

- You can restore a monthly detailed extracted file, then extract only one week's global and application summary data for analysis with PerfView.

It is imperative that you extract *all* data (summaries and detail) to the monthly log files. You cannot get data from a log file on a subsequent extraction if that data was not included in the original extraction. If

you extract only summaries from a log file, you cannot recreate detail data from it later.

System Analysis Tasks

We recommend the following for system analysis:

- Extract summary data only. You could transfer this extracted data to the analysis workstation or leave it on the host for analysis.
- Identify any problem areas or dates on the extracted log file.
- Extract detail data for problem dates. You could also look at the problem dates on the raw log file.
- Once you have extracted detail data on the problem dates, you can analyze the details locally.

3

Using the Utility Program

The MeasureWare Agent **utility** program serves as a tool for managing and reporting information on log files, the `parm` file, and the `alarmdef` file. You can use the **utility** program interactively or in batch mode to perform the following tasks.

- Scan raw or extracted log files and produce a report showing:
  - dates and times covered
  - times when the **scopeux** collector was not running
  - changes in **scopeux** parameter settings
  - changes in system configuration
  - log file disk space
  - effects of application and process settings in parameter file
- Resize raw log files (requires `superuser` capability)
- Check the `parm` file for syntax warnings or errors.
- Check an alarm definition file for syntax warnings or errors.
- Process log file data against alarm definitions to detect alarms in historical data.

This chapter contains information about the **utility** program's commands, parameters, and defaults. You can also use **utility** online `help` commands to check command syntax.

Running the Utility Program

There are three ways to run the **utility** program:

- Command line

  You can control the program by using parameters in the command line that invoke the program.

- Interactive mode

  You can supply parameters and commands interactively by executing the program with `stdin` set to an interactive terminal or workstation.

  If you are an experienced user, you can quickly select only those commands required for a given task. If you are a new user, you may want to use guided mode to receive more assistance in using the commands. In guided mode, you are asked to select from a list of options in order to perform a task. While in guided mode, the interactive commands that accomplish each task are listed as they are executed, so you can see how they are used. You can quit or re-enter guided mode at any time.

- Batch mode

  You can run the program and redirect `stdin` to a file that contains commands and parameters.

The syntax for the command line interface is similiar to typical UNIX command line interfaces on other programs and is described in detail in this chapter. The syntax for interactive and batch modes is the same and is described next.

Errors and missing data are handled differently for interactive mode than for command line and batch mode. You can supply additional data or correct mistakes in interactive mode, but not in command line and batch mode.

For interactive and batch mode the command syntax is the same: a command followed by one or more parameters. Parameters can be entered in any order; if a parameter has a value associated with it, the value must be entered immediately after the corresponding parameter.

There are two types of parameters—*required* parameters (for which there are no defaults) and *optional* parameters (for which defaults are provided). How **utility** handles these parameters depends on the mode in which it is running.

Parameters in Interactive Mode

If an *optional* parameter is missing, the program displays the default parameter and lets you either confirm it or override it.

If a *required* parameter is missing, the program prompts you to enter the parameter.

Parameters in Batch Mode

If an *optional* parameter is missing, the program uses the default values.

If a *required* parameter is missing, the program terminates.

Example

The example shows the differences between how the **utility** program's **resize** command works in batch mode and in interactive mode.

The **resize** command lets you set parameters for the following functions:

- Type of log file to be resized.
- Size of the new file.
- Amount of empty space to be left in the file.
- An action specifying whether or not the resize is to be performed.

The following example of the **resize** command resizes the global log file so that it contains a maximum of 120 days of data with empty space equal to 45 days:

```
resize global days=120 empty=45 yes
```

The results are the same whether you enter this command interactively or from a batch job.

The first parameter—`global`—indicates the type of log file data to be resized. If you do not supply this parameter, the consequent action for interactive and batch users would be the following:

- Batch users.

  The batch job would terminate because the `logfile` parameter has no default.

- Interactive users.

  You would be prompted to choose which type of log file to resize to complete the command.

The last parameter—`yes`—indicates that resizing will be performed unconditionally.

If you do not supply the `yes` parameter, the consequent action for interactive and batch users would be the following:

- Batch users.

  Resizing would continue since `yes` is the default action.

- Interactive users.

  You would be prompted to supply the action before resizing takes place.

Utility Command Line Interface

In addition to the interactive and batch mode command syntax, parameters and commands can be passed to the **utility** program through the command line interface. The command line interface fits into the typical UNIX environment by allowing the **utility** program to be easily invoked by shell scripts and allowing its input and output to be redirected into UNIX pipes.

Input and Output File Redirection

utility allows the following files to be redirected using normal UNIX shell command syntax.

`stdout`	Standard output file. All reports and extensive listings.
`stderr`	Standard errors file. All error messages and interactive prompts.
`stdin`	Standard input file. All interactive input.

For example, execute the following command to take commands from a file called `utilin` and direct reports to a file called `utilout` while directing all error messages to a file called `utilerr` (Korn shell example).

```
utility < utilin > utilout 2> utilerr
```

Command line arguments are listed in the following table.

Command Line Arguments

Command	Parameters		Description
`-b`	`date`	`time`	Sets starting date and time. (See **start** command description.)
`-e`	`date`	`time`	Sets ending date and time. (See **stop** command description.)
`-l`	`logfile`		Specifies input logfile. (See **logfile** command description.)
`-f`	`listfile`		Specifies output listing file. (See **list** command description.)
`-D`			Enables detail for **analyze**, **scan**, and **parm** file checking. (See **detail** command description.)
`-d`			Disables detail for **analyze**, **scan**, and **parm** file checking. (See **detail** command description.)
`-T`			Generates terse output report formats. (See **terse** command description.)
`-v`			Generates verbose output report formats.
`-xp`	`parmfile`		Syntax checks a **parm** file. (See **parmfile** command description.)
`-xc`	`alarmdef`		Syntax checks and sets the `alarmdef` file name to use with `-xa` (or **analyze** command). (See **checkdef** command description.)

Command Line Arguments (continued)

Command	Parameters		Description
-xa			Analyzes the log files using the **alarmdef** file. (See **analyze** command description.)
-xs	**logfile**		Scans the log files and produces a report. (See **scan** command description.)
-xr			Resizes a log file. (See **resize** command description.)
	global	**SIZE=nnn**	
	application	**DAYS=nnn**	(MeasureWare Server Agent only)
	process		(MeasureWare Server Agent only)
	device		
	transaction		
	EMPTY=nnn	**YES**	
	SPACE=nnn	**NO**	
		MAYBE	
-? or ?			Prints command line syntax. (See **help** command description.)

The following situation applies while evaluating parameters and executing commands entered on the command line:

Errors and missing data are handled exactly as in the corresponding batch mode command. That is, missing data is defaulted if possible and all errors cause the program to terminate immediately.

Echoing of commands and command results is disabled. **utility** does not read from its `stdin` file. It terminates following the actions in the command line.

```
utility -xp -d -xs
```

Which translates into:

`-xp`	Syntax checks the default `parm` file.
`-d`	Disables details in the scan report.
`-xs`	Performs the scan operation. No log file was specified so the default log file is scanned.

Utility Commands

This section describes the **utility** commands. It includes a syntax summary and a command reference section listing the commands in alphabetical order.

Note

Commands and parameters for **utility** can be entered with any combination of uppercase and lowercase letters. Only the first three letters of the command's name are required. For example, the command **logfile** can be entered as **LogFile** or it can be abbreviated to **log**.

Examples of various tasks using the **utility** program can be found in the program's online `help` facility.

Utility Command Syntax Summary

The following table contains a summary of **utility** command syntax and parameters.

Table 3-1.
utility Commands: Syntax and Parameters

Command	Parameter
analyze	
checkdef	*alarmdef* file
detail	`on` `off`
exit **e**	
guide	
help	*topic*
list	*filename* or *
logfile	*logfile*
menu **?**	
parmfile	*parmfile*
quit **q**	

Table 3-1.
utility Commands: Syntax and Parameters
(continued)

Command	Parameter

resize	`global`
	`application` (MeasureWare Server Agent only)
	`process` (MeasureWare Server Agent only)
	`device`
	`transaction`
	
	`days=` *maxdays*
	`size=` *maxMB*
	
	`empty=` *days*
	`space=` *MB*
	
	`yes`
	`no`
	`maybe`

scan	*logfile* Operation is also affected by the **list**, **start**, **stop**, and **detail** commands.
sh **!**	*shell command*
show	`all`

Table 3-1.
utility Commands: Syntax and Parameters
(continued)

Command	Parameter
start	*date* [*time*] `today` [*-days*][*time*] `last` [*-days*][*time*] `first` [*+days*][*time*]
stop	*date* [*time*] `today` [*-days*][*time*] `last` [*-days*][*time*] `first` [*+days*][*time*]
terse	`on` `off`

ANALYZE

Use the **analyze** command to analyze the data in a log file against alarm definitions in an `alarmdef` file and report resulting alarm status and activity. Before issuing the **analyze** command, you should run the **checkdef** command to check the alarm definitions syntax. **checkdef** also sets and saves the `alarmdef` file name to be used with **analyze**. If you do not issue **checkdef** before **analyze**, you are prompted for an `alarmdef` file name.

If you are using command line mode, (`-xa`), the default `alarmdef` file is used.

For detailed information about the `alarmdef` file and how to set up your alarm definitions, see Chapter 5, "Performance Alarms."

Syntax

```
analyze
```

How to Use It

When you issue the **analyze** command, by default it analyzes the log files in the default SCOPE data source against the alarm definitions in an `alarmdef` file.

If you want to analyze a specific log file, you can override the default SCOPE data source. To do so, you must place a USE statement in your alarm definition that specifies the name of the data source containing that log file. See "USE Statement" in Chapter 5, "Performance Alarms", for an explanation of how to do this. Once the data source is defined in the USE statement, you can use the **analyze** command to report alarm status and activity in that data source's log file.

The **analyze** command allows you to evaluate whether or not your alarm definitions are a good match against the historical data collected on your system. It also lets you

decide if your alarm definitions will generate too many or too few alarms on your analysis workstation.

Also, you can perform data analysis with definitions (IF statements) set in the `alarmdef` file because you can get information output by PRINT statements when conditions are met. (See Chapter 5, "Performance Alarms", for explanations of how to use the IF and PRINT statements in an alarm definition.)

You can optionally run the **start, stop**, and **detail** commands with **analyze** to customize the analyze process. You specify these commands in the following order:

```
checkdef
start
stop
detail
analyze
```

Use the **start** and **stop** commands if you want to analyze log file data that was collected during a specific period of time. (See the descriptions of the **start** and **stop** commands later in this chapter.)

While the **analyze** command is executing, it lists alarm events such as alarm start, end, and repeat status plus any text in associated print statements. Also, any text in PRINT statements is listed as conditions (in IF statements) become true. EXEC statements are not executed but are listed so you can see what would have been executed. SYMPTOMS are not evaluated. An alarm summary report shows a count of the number of alarms and the amount of time each alarm was active (on). The count includes alarm starts and repeats, but not alarm ends.

If you want to see the alarm summary report only, issue the `detail off` command.

Note

The **analyze** command defaults to `detail` on and automatically lists alarm events and the alarm summary. (This works in batch and interactive mode, but *not* in command line mode.)

Example

In the following example, the **checkdef**command checks the alarm definitions syntax in the `alarmdef1` file and then saves the name. Next, the **analyze** command analyzes the log file in the default SCOPE data source against the alarm definitions. The **start today** command specifies that only data logged today is to be analyzed. The **detail off** command causes **analyze** to list only the alarm summary.

```
utility>
checkdef alarmdef1
detail off
start today
analyze
```

CHECKDEF

Use the **checkdef** command to check the syntax of the alarm definitions in an `alarmdef` file and report any warnings or errors that are found. This command also sets and saves the `alarmdef` file name for use with the **analyze** command.

See Chapter 5, "Performance Alarms," for descriptions of the alarm definitions syntax and how to specify alarm definitions.

Syntax

```
checkdef [ alarmdef]
```

Parameters

alarmdef Specify the name of any `alarmdef` file. This can be a user-specified file or the default `alarmdef` file. If the `alarmdef` file is not in your current working directory, you must provide its path.

How To Use It

When you have determined that the alarm definitions are correct, you can process them against the data in a log file using the `analyze` command.

In batch mode, if no `alarmdef` file is specified, the default is the `alarmdef` file in the *datafiles* or *configuration* directory.

Note

Throughout this manual, the names *configuration* and *datafiles* are used to indicate platform-specific directories in MeasureWare Agent. See the **man** page for **scopeux** to see which of these directories is applicable to your MeasureWare Agent system.

In interactive mode, if no `alarmdef` file is specified, you are prompted to specify one.

Example

In the following example, **checkdef** checks the alarm definitions syntax in the `alarmdef1` file and then saves the name. Next, **analyze** processes the log file in the default SCOPE data source against the alarm definitions. The **detail off** command causes **analyze** to list only an alarm summary report.

```
utility>
checkdef alarmdef1
detail off
analyze
```

DETAIL

Use the **detail** command to control the level of detail printed in the `analyze`, `parm`, and `scan` reports.

The default is `detail on` in interactive and batch modes and `detail off` in command line mode.

Syntax

```
detail [on]
       [off]
```

Parameters

`on`	Prints the effective contents of the `parm` file as well as `parm` file errors. Prints complete `analyze` and `scan` reports.
`off`	In the `parm` file report, application definitions are *not* printed. In the `scan` report, **scopeux** collection times, initial `parm` file global information, and application definitions are *not* printed. In the `analyze` report, alarm events and alarm actions are *not* printed.

How to Use It

For explanations of how to use the **detail** command with the **analyze** and **parmfile** commands, see the **analyze, parmfile,** and **scan** sections in this chapter.

Examples

See the descriptions of the **analyze, parmfile**, and **scan** commands in this chapter for examples.

EXIT

Use the **exit** command to terminate the **utility** program. The **exit** command and the **quit** command are equivalent.

Syntax

```
exit
e
```

GUIDE

Use **guide** to enter guided commands mode. The guided command interface guides you through the various **utility** commands and prompts you to perform the most common functions that are available.

Syntax

```
guide
```

How to Use It

This command does not provide all possible combinations of parameter settings. It selects settings that should produce useful results for the majority of users. You can obtain full control over **utility's** functions through the regular command interface.

- To enter guided commands mode from the regular command interface, type `guide`.
- To accept the default value for a parameter, press Return.
- To terminate guided commands mode and return to the regular command interface, type `q` at any `utility guide>` prompt.

HELP

Use the **help** command to access the **utility** program's online help facility interactively.

Syntax

```
help [topic]
```

How to Use It

You enter parameters to obtain information on **utility** commands and tasks, or on help itself. You can navigate to different topics by entering a key word. If more than one page of information is available, the display pauses and waits for you to press Return before continuing. Type Q or quit to exit the help system and return to the **utility** program.

You can also request help on a specific topic. For example,

```
help tasks
```
or
```
help resize parms
```

When you use this form of the **help** command, you receive the help text for the specified topic and remain in the **utility** command entry context. Since you do not enter the help subsystem interactively, you do not have to type quit before entering the next **utility** command.

LIST

Use the **list** command to specify the list file for all **utility** reports. The report listed depends on which other commands are issued along with the **list** command. For example, using the **list** command with the **logfile, detail on**, and **scan** commands produces the list file for a detailed summary report of a log file.

Syntax

```
list [listfile]
```

How to Use It

There are two ways to specify the list file for reports:

- Redirect `stdout` when invoking the **utility** program by typing:

```
utility > utilrept
```

or

- Use the **list** command when **utility** is running by typing:

```
list utilrept
```

In either case, user interactions and errors are printed to `stderr` and reports go to the file specified.

The *listfile* parameter on the **list** command must represent a valid filename to which you have write access. Existing files have the new output appended to the end of existing contents. If the file does not exist, it will be created. You can output to a UNIX pipe for further processing.

To determine the current list file, type the **list** command without parameters:

```
list
```

If the list file is not `stdout`, most commands are echoed to the list file as they are entered.

Example

The following example shows how the **list** command is used to produce a summary report on the extracted log file `rxlog`. `list utilrept` directs the scan report listing to a disk file. `detail off` specifies less than full detail in the report. `scan` reads `rxlog` and produces the report. `list *` sets the list device back to the default `stdout`. `!lp utilrept` sends the disk file to the system printer.

```
utility>
logfile rxlog
list utilrept
detail off
scan
list *
!lp utilrept
```

LOGFILE

Use the **logfile** command to open a log file. For many **utility** program functions, a log file must be opened. You do this explicitly by issuing the **logfile** command or implicitly by issuing some other command. If you are in batch or command line mode and do not provide a log file name, the default `logglob` file in the *datafiles* directory is used. If you are in interactive mode and do not provide a log file name, you are prompted to provide one.

Syntax

`logfile` [*logfile*]

How to Use It

You can specify the name of either a raw or extracted log file. If you specify an *extracted* log file name, all information is obtained from this single file. If you specify a *raw* log file name, you must specify the name of the global log file. If the log file is not in your working directory, you must provide its path.

The other raw log files must be in the same directory as the global log file. Log files have the following names:

`logglob`	global log file.
`logappl`	application log file.
`logproc`	process log file.
`logdev`	device log file.
`logtran`	transaction log file
`logindx`	index log file

Note

`logappl` and `logproc` are supported only in MeasureWare Server Agent.

Once a log file is opened successfully, a report is printed or displayed showing the general content of the log file (or log files), as in the following example:

```
Global      file: logglob version D
Application file: logappl
Process     file: logproc
Device      file: logdev
Index       file: logindx
System ID: Homer
System Type  7013/20 S/N 0005 O/S 4.2.1
Data Collector: UX A.03.01
File created:   06/14/95
Data covers:    27 days to 7/10/95
Shift is:      All Day
Data records available are:
     Global Application Process Disk Volume
Maximum file sizes:
     Global=2.0 Application=5.0 Process=5.0 Device=10.0 Transaction=0.0 MB

The first GLOBAL      record is on 06/14/95 at 12:00 AM
The first APPLICATION record is on 06/25/95 at 12:00 AM
The first PROCESS     record is on 07/06/95 at 12:01 AM
The first DEVICE      record is on 05/01/95 at 11:50 AM
The Transaction   data file is empty
The default starting date & time = 05/01/95 11:50 AM   (FIRST +   0)
The default stopping date & time = 07/10/95 11:59 PM   (LAST -    0)
```

You can verify the log file you opened with the **show** command, as described later.

You can open another log file at any time by entering another **logfile** command. Any currently opened log file is closed before the new log file is opened.

The **resize** and **scan** commands require a log file to be open. If no log file is currently open, an implicit **logfile** command is executed.

Caution

Do not *rename* raw log files! Access to these files assumes the standard log file names are in effect.

If you must have more than one set of raw log files on the same system, create a separate directory for each set of files. Although the log file names cannot be changed, different directories may be used. If you want to resize the log files in any way, you must have read/write access to all the log files.

MENU

Use the **menu** command to print a short list of the available **utility** commands.

Syntax

menu

Example

```
Command    Parameters          Function
HELP       [topic]             Get information on commands and options
GUIDE                          Enter guided commands mode for novice users
LOGFILE    [logname]           Specify a log file to be processed
LIST       [filename|*]        Specify the listing file

START      [startdate time]    Set starting date & time for SCAN or ANALYZE
STOP       [stopdate  time]    Set ending   date & time for SCAN or ANALYZE
DETAIL     [ON|OFF]            Set report detail for SCAN, PARMFILE, or ANALYZE
SHOW       [ALL]               Show the current program settings

PARMFILE   [parmfile]          Check parsing of a parameter file

SCAN       [logname]           Read the log file and produce a summary report
RESIZE     [GLOB|APPL|PROC|DEV|TRAN] [DAYS=] [EMPTY=] Resize raw log files

CHECKDEF   [alarmdef]          Check parsing and set the alarmdef file
ANALYZE                        Analyze the log file using the alarmdef file

TERSE      [ON|OFF]            Set level of prompting and output details
! or Sh    [command]           Execute a system command
MENU or ?                      List the command menu (This listing)
EXIT or Q                      Terminate the program
```

PARMFILE

Use the **parmfile** command to view the MeasureWare Agent **parm** file settings that are used for data collection.

Syntax

```
parmfile [parmfile]
```

How to Use It

You can use the **parmfile** command to do any of the following:

- Examine the **parm** file for syntax warnings and review the resulting settings. All parameters are checked for correct syntax and errors are reported regardless of whether you are using MeasureWare Server Agent or MeasureWare Desktop Agent. This allows MeasureWare Desktop Agent to validate **parm** files for use with MeasureWare Server Agent. However, after the syntax check is completed, only the applicable settings are reported.
- Find out how much room is left for defining applications (MeasureWare Server Agent only).
- If **detail on** is specified, print the effective contents of the **parm** file plus any default settings that were not overridden, and print application definitions.

Note

If the **parm** file is not in your current working directory, you must provide its directory path when you specify it in the **parmfile** command.

In batch mode, if no **parm** file name is specified, the default is the **parm** file in the *datafiles* or *configuration* directory.

In interactive mode, if no **parm** file name is supplied, you are prompted to supply one.

Example

The following example checks the syntax of the current **parm** file, reports any warnings or errors, and then prints the logging parameter settings.

```
utility>
detail on
parmfile
```

QUIT

Use the **quit** command to terminate the **utility** program. The **quit** command and the **exit** command are equivalent.

Syntax

```
quit
q
```

RESIZE

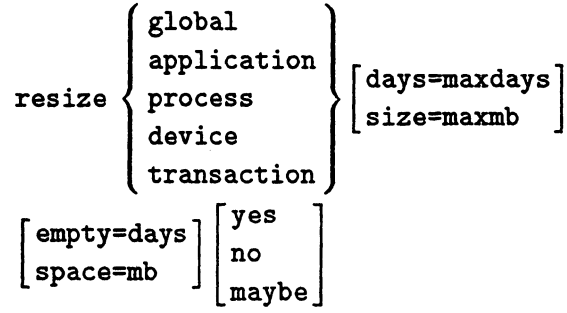
Use the **resize** command to manage the space in raw log files. This is the *only* program you should use to resize the raw log files in order to preserve coordination between the files and their internal control structures. If you use other tools you might remove or destroy the validity of these control structures.

You must have **superuser** capability to use this command.

Caution

The **utility** program *cannot* be used to resize extracted files. If you want to resize an extracted file, use the **extract** program to create a new extracted log file.

Syntax



Parameters

`log file type` Specifies the type of raw data you want to resize: `global`, `application`, `process`, `device`, or `transaction`, which correspond to the raw log files `logglob`, `logappl`, `logproc`, `logdev`, and `logtran`. If you do not specify a data type and are running **utility** in batch mode, the batch job terminates. If you are running **utility** interactively, you are prompted to supply the data type

based on those log files that currently exist.

Note

`logappl` and `logproc` are supported only in MeasureWare Server Agent.

`days` and `size` — Specify the maximum size of the log file. The actual size depends on the amount of data in the file.

`empty` and `space` — Specify the minimum amount of room required in the file after the resizing operation is complete. This value is used to determine if any of the data currently in the log file must be removed in the resizing process.

You might reasonably expect that a log file would not fill up until the specified number of days after a resizing operation. You may want to use this feature of the **resize** command to minimize the number of times a log file must be resized by **scopeux** because resizing can occur any time the file is filled. Using **utility** to force a certain amount of empty space in a log file causes the log file to be resized when you want it to be.

The `days` and `empty` values are entered in units of days; the `size` and `space` values are entered in units of megabytes. Days are converted to megabytes by using an average megabytes-per-day value for the log file. This conversion factor varies depending on the type of data being logged and the particular characteristics of your system.

More accurate average-megabytes-per-day conversion factors can be obtained if you issue the **scan** command on the existing log file before you issue the **resize** command. A **scan** measures the accumulation rates for your system. If no **scan** is done or if the measured conversion factor seems unreasonable, the **resize**

command uses a default conversion factor for each type of data.

Yes Specifies that resizing should be unconditionally performed. This is the default action if **utility** is not running interactively. If no action is specified when **utility** is running interactively, you are prompted to supply the action.

No Specifies that resizing should not be performed. This parameter can be specified as an action if you want to see the resizing report but do not want to perform the resizing at that time.

Maybe Specifies that **utility** should decide whether or not to resize the file. This parameter forces **utility** to make this decision based on the current amount of empty space in the log file (before any resizing) and the amount of space specified in the **resize** command. If the current log file contains at least as much empty space as specified, resizing does not occur. If the current log file contains less than the specified empty space, resizing occurs.

If the resizing can be made without removing any data from the log file (for example, increasing the maximum log file size, or reducing the maximum log file size without having to remove any existing data), resizing occurs.

The **maybe** parameter is intended primarily for use by periodic batch executions. See the "Examples" subsection below for an explanation of

how to use the **resize** command in this manner.

Default resizing parameters are shown in the following table.

Table 3-2. Default Resizing Parameters

Parameter	If Executed Interactively	If Executed in Batch
`log file type`	You are prompted for each available log file type.	No default. This is a required parameter.
`days` `size`	The current file size.	The current file size.

`empty`
 	
`space`
	The current amount of empty space or enough empty space to retain all data currently in the file, whichever is smaller.	The current amount of empty space or enough empty space to retain all data currently in the file, whichever is smaller.

`yes`
 	
`no`
`maybe`
	You are prompted following the reported disk space results.	Yes. Resizing will occur.

How to Use It

You must shut down **scopeux** before resizing log files.

A raw log file must be opened before resizing can be performed. Open the raw log file with the **logfile** command before issuing the **resize** command. The files cannot be opened by any other process.

The **resize** command creates the new file `/tmp/scopelog` before deleting the original file. Make sure there is

sufficient disk space in the /tmp directory to hold the original log file before doing the resizing procedure.

After resizing, a log file consists of data plus empty space. The data retained is calculated as the maximum file size minus the required empty space. Any data removed during the resizing operation is lost. To save log file data for longer periods, use **extract** to copy this data to an extracted file *before* doing the **resize** operation.

Resize Command Reports

One standard report is produced when you resize a raw log file. It shows the three interrelated disk space categories of maximum file size, data records, and empty space, before and after resizing. For example:

```
resize global days=120;empty=10
empty space raised to match file size and data records

final resizing parameters:
file: logglob                          megabytes / day: 0.101199
              -----currently-----    --after resizing---
maximum size:   65 days (  6.6 mb)   120 days ( 12.1 mb)  83% increase
data records:   61 days (  6.2 mb)    61 days (  6.2 mb) no data removed
empty space:     4 days (  0.5 mb)    59 days (  6.0 mb) 1225% increase
```

The megabytes per day value is used to convert between days and megabytes. It is either the value obtained during the scan function or a default for the type of data being resized.

The far right-hand column is a summary of the net change in each category of log file space. Maximum size and empty space can increase, decrease, or remain unchanged. Data records have either no data removed or a specified amount of data removed during resizing.

If the resize is done interactively and one or more parameters are defaults, you can get a preliminary resizing report. This report summarizes the current log

file contents and any parameters that were provided. The report is provided to aid in answering questions on the unspecified parameters. For example:

```
resize global days=20

file resizing parameters (based on average daily
space estimates and user resizing parameters)
file: logglob                          megabytes / day: 0.101199
                 -----currently-----    --after resizing---
maximum size:    65 days (  6.6 mb)     20 days (  2.0 mb)
data records:    61 days (  6.2 mb)     ??
empty space:      4 days (  0.5 mb)     ??
```

In this example, you are prompted to supply the amount of empty space for the file before the final resizing report is given. If no action parameter is given for interactive resizing, you are prompted for whether or not to resize the log file immediately following the final resizing report.

Examples

The following commands show how to resize a raw process log file. The scan is performed before the resize to increase the accuracy of the number-of-days calculations.

```
utility>
logfile logglob
detail off
scan
resize process days=60 empty=30 yes
```

`days=60` specifies holding a maximum of 60 days of data. `empty=30` specifies that 30 days of this file are currently empty. That is, the file is resized with no more than 30 days of data in the file to leave room for 30 more days out of a total of 60 days of space. `yes` specifies that the

resizing operation should take place regardless of current empty space.

The next example shows how you might use the **resize** command in batch mode to ensure that log files do not fill up during the upcoming week (forcing **scopeux** to resize them). You could schedule a `cron` script that specifies a minimum amount of space such as 7 days—or perhaps 10 days, just to be safe.

The following `shell` script accomplishes this:

```
echo "detail off"                              >> utilin
echo "scan"                                    >> utilin
echo "resize global      empty=10 maybe"       >> utilin
echo "resize application empty=10 maybe"       >> utilin
echo "resize process     empty=10 maybe"       >> utilin
echo "resize device      empty=10 maybe"       >> utilin
echo "quit"                                    >> utilin
utility < utilin > utilout 2> utilerr
```

Note

If you use the above script, remember to stop **scopeux** before running the script. See the "Starting Up MeasureWare Agent" chapter in your *MeasureWare Agent: Installation and Configuration Guide* for information about stopping and starting **scopeux.**

Specifying `maybe` instead of `yes` avoids any resizing operations if 10 or more days of empty space currently exist in any log files. Note that the maximum file size defaults to the current maximum file size for each file. This allows the files to be resized to new maximum sizes without affecting this script.

SCAN

Use the **scan** command to read a log file and write a report on its contents. (For a detailed description of the report, see Appendix A, "Utility Scan Report Details.")

Syntax

```
scan [logfile]
```

How to Use It

The **scan** command requires a log file to be opened. The log file scanned is the first of one of the following:

- The log file named in the **scan** command itself.
- The last log file opened by any previous command.
- The default log file that resides in the *datafiles* directory.

  In this case, interactive users are prompted to override the default log file name if so desired.

Note

The following related commands are discussed briefly in this section and in more detail elsewhere in this chapter under the command names: **detail**, **list**, **start**, and **stop**.

The following commands affect the operation of the **scan** function:

detail	Specifies the amount of detail in the report. The default, `detail on`, specifies full detail.
list	Redirects the report listing to another file. The default is to list to the standard list device.

start	Specifies the date and time of the first log file record you want to scan. The default is the beginning of the log file.
stop	Specifies the date and time of the last log file record you want to scan. The default is the end of the log file.

The **scan** command report consists of 12 sections. You can control which sections are present in the report by using the **detail** command.

The following sections are printed only if `detail on` (the default) is specified:

- Initial `parm` file global information and system configuration information
- Initial `parm` file application definitions
- `parm` file global changes
- `parm` file application addition/deletion notifications
- Collector off-time notifications
- Application-specific summary reports

The following four sections are always printed (even if `detail off` is specified):

- Scan start and stop actual dates and times
- Collector coverage summary
- Log file contents summary
- Log file empty space summary

The following section is always printed if application data was scanned (even if `detail off` is specified):

- Application overall summary

The following section is always printed if process data was scanned (even if `detail off` is specified):

- Process log reason summary

Note

Application and process data are scanned and included in the report *only* if you are using MeasureWare Server Agent.

Example

This example shows the commands you would use to scan all the current log files in addition to `logglob` and display a report on the logging details. The report includes details on the initial `parm` file settings plus any changes to these settings. An overall summary of disk space consumed by each type of logging is also displayed. In this example, the listing is defaulted to the `stdout` file.

```
utility>
logfile logglob
detail on
scan
```

SH

Use **sh** to enter a shell command without exiting **utility** by typing **sh** or an exclamation point (**!**) followed by a shell command.

Syntax

```
sh or ! [shell command]
```

Parameters

`sh ls`	Executes the `ls` command and returns to **utility**
`!ls`	Same as above
`!ksh`	Starts a Korn shell. Does not return immediately to **utility**. Use CTRL-d Return or type `exit` when ready to return to **utility**.

If you omit the shell command, you are prompted to supply it. For example:

```
sh
enter shell command: ls
```

How to Use It

Following the execution of the single command, you automatically return to the **utility** program. If you want to issue multiple shell commands without returning to **utility** after each one, you can start a new shell.

SHOW

Use the **show** command to list the names of the files that are open and the status of the **utility** parameters that can be set.

Syntax

```
show [all]
```

For example,

```
Logfile:  logglob
List:     "stdout"
Detail:   ON for ANALYZE, SCAN, and PARMFILE functions

The default starting date & time = 05/01/95 11:50 AM (FIRST + 0)
The default stopping date & time = 05/25/95 11:59 PM (LAST  - 0)
The default shift = 12:00 AM - 12:00 AM
```

Note

The default shift time is shown for information only. Shift time cannot be changed in **utility**.

Adding the optional parameter **all** prints more information about the log file if one is open. For example,

```
Logfile:   logglob
    Global       File: logglob
    Application File: logappl
    Process      File: logproc
    Device       File: logdev
    Transaction File: logtran
    Index        File: logindx
    System ID:  Homer
    System Type 7013/20      S/N 0005      O/S 4.2.1
    Data Collector: UX A.03.01
    File created:   06/14/95
    Data Covers:    27 days to 07/10/95
    Shift is:       All Day


Data records available are:
   Global Application Process Disk Volume

Maximum file sizes:
   Global=2.0 Application=5.0 Process=5.0 Device=10.0 Transaction=0.0 MB


List:     "stdout"
Detail:  ON for ANALYZE, PARMFILE and SCAN functions
The default starting date & time = 05/01/95 11:50 AM    (FIRST + 0)
The default stopping date & time = 07/10/95 11:59 PM    (LAST  - 0)
The default shift = 12:00 AM - 12:00 AM
```

Note logappl and logproc log files are supported only in MeasureWare Server Agent.

START

Use the **start** command to specify the beginning of the subset of a log file that you want to scan or analyze.

start lets you start the scan or analyze process at data that was logged at a specific date and time.

The default starting date and time is set to the date and time of the first record of any type in a log file that has been currently opened with the **logfile** command. Otherwise, the default is undefined (0 Greenwich Mean Time).

Syntax

```
start [ date [ time ]
        today [ -days ] [ time ]
        last [ -days ] [ time ]
        first [ +days ] [ time ] ]
```

Parameters

date — The date format depends on the native language configured on the system being used. If you do not use native languages or have set the default language as C, the date format is mm/dd/yy (month/day/year) or 02/28/94 for February 28, 1994.

time — The time format also depends on the native language being used. For C, the format is hh:mm AM or hh:mm PM (hour:minute in 12-hour format with the AM/PM suffix) such as 07:00 AM for 7 o'clock in the morning. Twenty-four hour time is accepted in all languages. For example, 23:30 would be accepted for 11:30 PM.

If the date or time is entered in an unacceptable format, an example in the correct format is shown.

If no start time is given, midnight (12:00 AM) is assumed. A starting time of midnight for a given day starts at the *beginning* of that day (00:00 on a 24-hour clock).

`today` — The `today` parameter represents the current date. The parameter `today-`*days* specifies the number of days *prior to* today's date. For example, `today-1` indicates yesterday's date.

`last` — The `last` parameter represents the last date contained in the log file. The parameter `last-`*days* specifies the number of days *prior to* the last date in the log file.

`first` — The `first` parameter represents the first date contained in the log file. The parameter `first+`*days* specifies the number of days *after* the first date in the log file.

Note

If you are not sure whether native language support is installed on your system, you can force **utility** to use the C date and time formats by issuing the following statement before running **utility**:

```
LANG=C; export LANG
```

or in C Shell

```
setenv LANG C
```

How to Use It

The **start** command is useful if you have a very large log file and do not want to scan or analyze the entire file. You can also use it to specify a specific time period for which data is scanned. For example, you can scan a log file for data that was logged for a 14-day period beginning 14-days before the present date.

You can use **start** with the **stop** command to further limit the log file records you want to scan.

Example

This example shows a scan of a current raw log file that starts with records logged from July 5, 1995 at 8:00 AM until the present date and time.

```
utility>
logfile logglob
detail on
start 7/5/95 8:00 AM
scan
```

STOP

Use the **stop** command to specify the end of a subset of a log file that you want to scan or analyze.

stop lets you terminate the scan or analyze process at data that was logged at a specific date and time.

The default stopping date and time is set to the date and time of the last record of any type in a log file that has been currently opened with the **logfile** command. (Otherwise, the default is undefined (0 Greenwich Mean Time.)

Syntax

```
stop [ date [ time ]
     | today [ -days ] [ time ]
     | last [ -days ] [ time ]
     | first [ +days ] [ time ] ]
```

Parameters

date — The date format depends on the native language configured on the system being used. If you do not use native languages or have set the default language as C, the date format is mm/dd/yy (month/day/year) or 02/28/95 for February 28, 1995.

time — The time format also depends on the native language being used. For C, the format is hh:mm AM or hh:mm PM (hour:minute in 12-hour format with the AM/PM suffix) such as 07:00 AM for 7 o'clock in the morning. Twenty-four hour time is accepted in all languages. For example, 23:30 would be accepted for 11:30 PM.

If the date or time is entered in an unacceptable format, an example in the correct format is shown.

If no stop time is given, one minute before midnight (11:59 PM) is assumed. A stopping time of midnight (12:00 AM) for a given day stops at the *end* of that day (23:59 on a 24-hour clock).

`today` — The `today` parameter represents the current date. The parameter `today-`*days* specifies the number of days *prior to* today's date. For example, `today-1` indicates yesterday's date.

`last` — The `last` parameter can be used to represent the last date contained in the log file. The parameter `last`-days specifies the number of days *prior to* the last date in the log file.

`first` — The `first` parameter can be used to represent the first date contained in the log file. The parameter `first`+days specifies the number of days *after* the first date in the log file.

Note

If you are not sure whether native language support is installed on your system, you can force **utility** to use the C date and time formats by issuing the following statement before running **utility**:

```
LANG=C; export LANG
```

or in C Shell

```
setenv LANG C
```

How to Use It

The **stop** command is useful if you have a very large log file and do not want to scan the entire file. You can also use it to specify a specific time period for which data is scanned. For example, you can scan a log file for seven-days worth of data that was logged starting a month before the present date.

You can use **stop** with the **start** command to further limit the log file records you want to scan.

Example

This example shows a scan of 14 days worth of data. The scan starts with records logged from July 5, 1995 at 8:00 AM and stops at the last record logged July 18, 1995 at 11:59 PM.

```
utility>
logfile logglob
detail on
start 7/5/95 8:00 AM
stop 7/18/95 11:59 PM
scan
```

TERSE

Use the **terse** command to control how much text is displayed on the screen.

Syntax

```
terse [on]
      [off]
```

`on`	Displays text that contains fewer details.
`off`	Displays text that contains more details.

How to Use It

The **terse** command affects the amount of detail that is displayed in user prompts, as well as output to the screen from the **list** and **show** commands.

4

Using the Extract Program

The MeasureWare Agent **extract** program has two main functions: it lets you extract data from raw log files, summarize it, and write it to extracted log files. It also lets you export log file data for use by and analysis products such as PerfView.

Two types of log files are used by MeasureWare Agent:

- **scopeux** log files, which contain data collected in MeasureWare Agent by the **scopeux** collector.
- DSI log files, which contain self-describing data collected outside of MeasureWare Agent and are created by MeasureWare Agent's DSI programs.

You have three different ways to control the **extract** program:

- Insert parameters in the command line.
- Supply parameters and commands interactively.
- Run the program in batch mode.

Use the **extract** program to perform the following tasks:

- You can extract subsets of data from raw log files into an extracted log file format that is suitable for placing in archives, for transport between systems, and for transport to the analysis workstation for analysis by PerfView.
- You can summarize detailed global and application data into hourly summaries for compact storage of data covering very long time periods. For example,

one year's worth of global summary data requires less than 4 megabytes of disk space.

- You can manage archived log file data by extracting or exporting data from extracted format files, appending data to existing extracted log files, and subsetting data by type, date, and shift (hour of day).
- You can **export** data from raw or extracted log files into `ASCII`, `BINARY`, `DATAFILE`, or `WK1` formats suitable for reporting and analysis or for importing into spreadsheets or similiar analysis packages.

Running the Extract Program

There are three ways to run the **extract** program:

- Command line

  You can control **extract** by using parameters in the command line that invoke the program.

- Interactive mode

  You can supply parameters and commands interactively by executing the program with `stdin` set to an interactive terminal or workstation.

  If you are an experienced user, you can quickly select only those commands required for a given task. If you are a new user, you may want to specify "guide" to receive more assistance in using **extract**. In guided mode, you are asked to select from a list of options in order to perform a task. While in guided mode, the interactive commands that accomplish each task are listed as they are executed, so you can see how they are used. You can quit or re-enter guided mode at any time.

- Batch mode

  You can run the program and redirect `stdin` to a file that contains commands and parameters.

The syntax for the command line interface is similiar to typical UNIX command line interfaces on other programs and is described in detail in this chapter. The syntax for interactive and batch modes is the same and is described next.

Errors and missing data are handled differently for interactive mode than for command line and batch mode, because you can supply additional data or correct mistakes in interactive mode, but not in command line and batch mode.

For interactive and batch mode the command syntax is the same: a command followed by one or more parameters. Parameters can be entered in any order; if a parameter has a value associated with it, the value must be entered immediately after the corresponding parameter.

There are two types of parameters—*required* parameters (for which there are no defaults) and *optional* parameters (for which defaults are provided). How MeasureWare Agent handles these parameters depends on the mode in which it is running.

Parameters in Interactive Mode

If an *optional* parameter is missing, the program displays the default parameter and lets you either confirm it or override it.

If a *required* parameter is missing, the program prompts you to enter the parameter.

Parameters in Batch Mode

If an *optional* parameter is missing, the program uses the default values.

If a *required* parameter is missing, the program terminates.

Extract Command Line Interface

In addition to the interactive and batch mode command syntax, parameters and commands can be passed to the **extract** program through the command line interface. The command line interface fits into the typical UNIX environment by allowing the **extract** program to be easily invoked by shell scripts and allowing its input and output to be redirected into UNIX pipes.

Input and Output File Redirection

extract allows the following files to be redirected using normal UNIX shell command syntax.

`stdout`	Standard output file. All reports and extensive listings.
`stderr`	Standard errors file. All error messages and interactive prompts.
`stdin`	Standard input file. All interactive input.

For example, to take commands from a file called `extin` and direct reports to a file called `extout` while directing all error messages to a file called `exterr` (Korn shell example), type:

```
extract < extin > extout 2> exterr
```

Command line arguments are listed in the following table.

Table 4-1. Command Line Arguments

Command	Parameters		Description
`-b`	`date`	`time`	Sets starting date and time. (See **start** command description.)
`-e`	`date`	`time`	Sets ending date and time. (See **stop** command description.)
`-B`		`UNIX start time`	Sets starting UNIX time. (See **begin** command description.)
`-E`		`UNIX end time`	Sets ending UNIX time. (See **end** command description.)
`-s`	`time-time`	`NOWEEKENDS`	Start time, end time, weekends. (See **shift** command description.)
`-l`	`logfile`		Specifies input log file. (See **logfile** command description.)
`-r`	`reportfile`		Specifies a report file for export function. (See **report** command description.)

`-C`	`classname`	`opt`	Specifies self-describing (DSI) and user-defined data to export. (See **class** command description.)
			`opt =`
			`detail`
			`summary`
			`both`

Table 4-1. Command Line Arguments (continued)

Command	Parameters	Description
sep	**char**	Sets export column separator character. (See **separator** command description.)
sum	**seconds**	Sets export interval in seconds. (See **interval** command description.)
-h		Sets export headings off. (See **heading** command description.)
-H		Sets export headings on. (See **heading** command description.)
-m		Sets display of missing exported records off. (See **missing** command description.)
-M		Sets display of missing exported records on. (See **missing** command description.)
-novalue	**value**	Sets null indicator for a missing exported value. (See **novalue** command description.)
-ut		Displays exported date and time in UNIX format. (See **UNIX** command description.)
-fd		Displays exported date and time in readable format. (See **format** command description.)
-we	**1.... 7**	Sets days to exclude from export; 1=Sunday. (See **weekdays** command description.)

Table 4-1. Command Line Arguments (continued)

Command	Parameters	Description
`-gapkcdznGADZN`		Selects types of data to extract/export: `g` = global detail (See **global** command description.) `a` = application detail (See **application** command description.) `p` = process detail (See **process** command description.) `k` = process killed (See **process** command description.) `d` = disk device detail (See **disk** command description.) `c` = configuration detail (See **configuration** command description.) `z` = lvolume detail (See **lvolume** command description.) `n` = netif detail (See **netif** command description.) `G` = global summary (See **global** command description.) `A` = application summary (See **application** command description.) `D` = disk device summary (See **disk** command description.) `Z` = lvolume summary (See **lvolume** command description.) `N` = netif summary (See **netif** command description.)

Table 4-1. Command Line Arguments (continued)

Command	Parameters		Description
-T			Sets terse output to **stdout**. (See **terse** command description.)
-v			Sets verbose output to **stdout**.
-f	**filename**	**,new** **,append** **,purge**	Sends extract or export data to a file. If no filename, sends extract data to **rxlog** and export data to **xfr***_logfile_**.ext**. (See **output** command description.)
-xp	**xopt**		Exports data to external format files. (See **export** command description.)
-xt	**xopt**		Extracts data to MeasureWare Agent UNIX internal format. (See **extract** command description.)
			xopt = **dwmy (Day Week Month Year)** **dwmy-[offset]** **dwmy [absolute]**
-xw	**week**		Extracts a calendar week's data. (See **weekly** command description.)
-xm	**month**		Extracts a calendar month's data. (See **monthly** command description.)
-xy	**year**		Extracts a calendar year's data. (See **yearly** command description.)
?			Prints command line syntax. (See **help** command description.)

When you are evaluating parameters and executing commands entered on the command line, the following rules apply:

- Errors and missing data are handled exactly as in the corresponding batch mode command. That is, missing data will be defaulted if possible and all errors cause the program to terminate immediately.
- Echoing of commands and command results is disabled unless the `-v` argument is used to enable verbose mode.
- If no valid action is specified (`-xp`, `-xw`, `-xm`, `-xy`, or `-xt`), **extract** starts reading commands from its `stdin` file after all parameters have been processed.
- If an action is specified (`-xp`, `-xw`, `-xm`, `-xy`, or `-xt`), the program will execute those commands after all other parameters are evaluated, regardless of where they were positioned in the list of parameters.
- If an action is specified in the command line, **extract** will not read from its `stdin` file; instead it will terminate following the action:

```
extract -f rxdata -r rept1 -xp d-1 -G
```

Which translates into:

`-f rxdata`	Outputs to a file named `rxdata`
`-r rept1`	File `rept1` contains the report format desired
`-xp d-1`	Exports data for this day minus 1 (yesterday)
`-G`	Exports global summary data

Note that the actual exporting is not done until the end so the `-G` parameter is processed before the export is done.

Also notice that the log file was not specified so it defaults to the file `logglob` in the *datafiles* directory.

Since an action was specified (`-xp`), once the export is finished the **extract** program terminates without reading from its `stdin` file. In addition, verbose mode was not set with the `-v` command so all extraneous output to `stdout` is eliminated.

Extract Commands

This section describes the **extract** commands. It includes a table showing command syntax, a table of commands for extracting and exporting data, command reference section describing the commands in alphabetical order.

Note

Commands and parameters for **extract** can be entered with any combination of uppercase and lowercase letters. Only the first three letters of the command's name are required, except for the **weekdays** command that requires you to enter the whole name. For example, the command `application detail` can be abbreviated as `app det`.

Examples of various tasks using the **extract** program can be found in the program's online `help` facility.

Extract Command Syntax Summary

The following table summarizes the syntax of **extract** commands and their parameters.

Table 4-2.
Extract Commands: Syntax and Parameters

Command	Parameter

application	`on`
	`detail`
	`summary`
	`both`
	`off`

begin	`UNIX` *start-time*

class	`detail`
	`summary`
	`both`
	`off`

configuration	`on`
	`detail`
	`off`

disk	`on`
	`detail`
	`summary`
	`both`
	`off`

end	`UNIX` *end-time*
exit **e**	
export	`day` [*dd*] [*-days*] `week` [*ww*] [*-weeks*] `month` [*mm*] [*-months*] `year` [*yy*] [*-years*]
extract	`day` [*dd*] [*-days*] `week` [*ww*] [*-weeks*] `month` [*mm*] [*-months*] `year` [*yy*] [*-years*]

Table 4-2.
Extract Commands: Syntax and Parameters
(continued)

Command	Parameter
format	

global	`on`
	`detail`
	`summary`
	`both`
	`off`

guide	
heading	`on` `off`
help	*topic*
interval	*seconds*
list	*filename* *
logfile	*logfile*

lvolume	`on`
	`detail`
	`summary`
	`both`
	`off`

menu **?**	
missing	`on` `off`
monthly	*yymm* *mm*

Table 4-2.
Extract Commands: Syntax and Parameters
(continued)

Command	Parameter
netif	on detail summary both off
novalue	*value*
output	*outfile* ,new ,purge ,append
process	on detail [app= #[-#],..] off killed
quit **q**	
report	[*reportfile*] ,show
separator	*char_separator*
sh **!**	*shell command*
shift	*starttime - stoptime* all day noweekends
show	all

Table 4-2.
Extract Commands: Syntax and Parameters
(continued)

Command	Parameter
start	*date* [*time*] `today` [*-days*] [*time*] `last` [*-days*] [*time*] `first` [*+days*] [*time*]
stop	*date* [*time*] `today` [*-days*] [*time*] `last` [*-days*] [*time*] `first` [*+days*] [*time*]
terse	`on` `off`

transaction	`on`
	`detail`
	`summary`
	`both`
	`off`

UNIX	
weekdays	`1.....7`
weekly	*yyww* *ww*
yearly	*yyyy* *yy*

Extract Commands for Extracting and Exporting

The following table lists the commands that are used for extracting and exporting data and the types of log files used (**scopeux** log files or DSI log files).

Table 4-3. Extract Commands: Extracting and Exporting Data

Command	Extract Data	Export Data	Scopeux Log Files	DSI Log Files
application	x	x	x	
begin	x	x	x	x
class		x	x	x
configuration		x	x	
disk	x	x	x	
end	x	x	x	x
export		x	x	x
extract	x		x	
format		x	x	x
global	x	x	x	
heading		x	x	x
interval		x	x	x
logfile	x	x	x	x
lvolume	x	x	x	
missing		x	x	x
monthly	x		x	
netif		x	x	
novalue		x	x	x
output	x	x	x	x
process	x	x	x	
report		x	x	x
separator		x	x	x
shift	x		x	x
start	x	x	x	x
stop	x	x	x	x
transaction	x	x	x	
UNIX		x		x
weekdays		x	x	x
weekly	x		x	
yearly	x		x	

APPLICATION

Use the **application** command to select the type of application data that is being extracted or exported.

The default is **application off**.

Note

The **application** command is used only in MeasureWare Server Agent.

Syntax

```
application [ on      ]
            | detail  |
            | summary |
            | both    |
            [ off     ]
```

Parameters

`on` or `detail` — The `on` *or* `detail` parameters specifies that raw, 5-minute detail data should be extracted or exported.

When using PerfView, detail data must be included in an extracted file before drawing application graphs with points every 5 minutes.

`summary` — The `summary` parameter specifies that data should be summarized by:

- the number of seconds specified with the **interval** command (export only)
- the number of minutes specified with the `SUMMARY` option in the selected report file (export only)
- the default summary interval of one hour (export or extract)

Summarization can significantly reduce the size of the resulting extracted or exported data, depending on the summarization interval used. For example, hourly summary data is about one-tenth the size of 5-minute detail data.

`both` — The `both` parameter specifies that detail data and summary data are to be extracted or exported.

This option maintains the access speed gained with hourly summary records and allows PerfView to produce application graphs with points every 5 minutes.

The disk space required to extract both `detail` and `summary` data is about 10 percent more than the disk space needed for extracted detail data alone.

`off` — The `off` parameter specifies that no application data is to be extracted or exported.

Example

In this example, you use the **application** command to export detailed application log file data:

```
extract>
logfile logglob
global off
application detail
report myrept
export
quit
```

BEGIN

Use **begin** to set the starting time in UNIX format for extracting or exporting data.

Syntax

`begin` [*start-time*]

Parameters

start-time — The UNIX time of the first interval to extract or export (seconds since 1/1/70 00:00:00).

How to Use It

You must use this command if you are exporting data with the PT_START variable in a `roll by action` statement in the class specification. (See the "Class Specifications" section in the *MeasureWare Agent: Data Source Integration Guide*.)

Example

In this example, you use the **begin** command to specify March 4, 1994 8:00 am in UNIX time as the start time of the first interval to be extracted.

```
extract>
logfile logglob
begin 762768000
end 762800400
output myout
global summary
export
quit
```

CLASS

Use the **class** command to specify the type of DSI data to be exported.

The default is **class off**.

Syntax

```
class [classname] [detail | summary | both | off]
```

Parameters

classname — Name of a group of similarly classified metrics.

`detail` — The `detail` parameter specifies how much detail data is exported according to the time set in the DSI log file. (See the *MeasureWare Agent: Data Source Integration Guide*)

`summary` — The `summary` parameter specifies that data should be summarized by:

- the number of seconds specified with the **interval** command (export only)
- the number of minutes specified with the `SUMMARY` option in the selected report file (export only)
- the default summary interval of one hour (export or extract)

Summarization can significantly reduce the size of the resulting extracted or exported data, depending on the summarization interval used. For example, hourly summary data is about

one-tenth the size of 5-minute detail data.

both The both parameter specifies that both detail data and summary data are to be exported.

off The off parameter specifies that no DSI data is to be exported.

Examples

To get DETAIL data on a DSI log file that contains a class named ACCTG_INFO, issue the following command:

```
CLASS ACCTG_INFO detail
```

Once the log file is specified by the user and opened by **extract**, the ACCTG_INFO class is verified to exist in the log file and can subsequently be exported.

Other variations of this command are:

```
CLASS ACCTG_INFO DETAIL
class ACCTG_INFO detail
cla ACCTG_INFO det
```

Commands can be either uppercase or lowercase. Class names are always upshifted and then compared.

CONFIGURATION

Use the **configuration** command to specify whether or not to export system configuration information.

The default is **configuration off**.

Syntax

```
configuration [ on     ]
              [ detail ]
              [ off    ]
```

Parameters

on or `detail`	The on or `detail` parameters specify that all configuration records should be exported.
`off`	The `off` parameter specifies that no configuration data is to be exported.

All configuration information available in the log file is exported. Any `begin`, `end`, `shift`, `start`, `stop` or `noweekends` commands that are used with the **configuration** command are ignored.

Note

The **configuration** command affects only the **export** function. It does not affect the **extract** function since the **extract** function *always* extracts system configuration information.

Example

In this example, you use the **configuration** command to export system configuration information.

```
extract>
logfile logglob
configuration on
report myrept
export
quit
```

DISK

Use the **disk** command to select the type of disk device data that is being extracted or exported.

The default is **disk off**.

Syntax

```
        [ on      ]
        | detail  |
disk    | summary |
        | both    |
        [ off     ]
```

Parameters

`on` or `detail` — The `on` or `detail` parameters specify that raw, 5-minute detail data should be extracted or exported.

`summary` — The `summary` parameter specifies that data should be summarized by:

- the number of seconds specified with the **interval** command (export only)
- the number of minutes specified with the `SUMMARY` option in the selected report file (export only)
- the default summary interval of one hour (export or extract)

Summarization can significantly reduce the size of the resulting extracted or exported data, depending on the summarization interval used. For example, hourly summary data is about one-tenth the size of 5-minute detail data.

	Summarization reduces the size of the disk device data to about one-tenth the size of the detail data.
`both`	The `both` parameter specifies that detail data and summary data are to be extracted or exported.
`off`	The `off` parameter specifies that no disk device data is to be extracted or exported.

Example

In this example, you export disk summary data that was collected starting September 5, 1995.

```
extract>
logfile logglob
global off
disk summary
start 9/5/95
export
quit
```

END

Use the **end** command to set the ending date and time in UNIX format for exporting data.

Syntax

end [*end-time*]

Parameters

end-time The last interval to export in UNIX time (seconds since 1/1/70 00:00:00).

How to Use It

You must use this command if you are exporting data with the PT_END variable in a **roll by action** statement in the class specification. (See the "Class Specifications" section in the *MeasureWare Agent: Data Source Integration Guide*.)

Example

In this example, you use the **end** command to specify March 4, 1994 4:00 pm in UNIX time as the end time of the last interval to export.

```
extract>
logfile logglob
begin 762768000
end 762800400
output myout
global summary
export
quit
```

EXIT

Use the **exit** command to end the **extract** program. The **quit** (or **q**) command is an acceptable alternative to the program's **exit** command.

Syntax

```
exit
e
```

EXPORT

Use the **export** command to start the process of exporting data.

Syntax

```
export [ day   [ dd ] [ yyddd ] [ -days ]   ]
       [ week  [ ww ] [ yyww ] [ -weeks ]  ]
       [ month [ mm ] [ yymm ] [ -months ] ]
       [ year  [ yy ] [ yyyy ] [ -years ]  ]
```

Parameters

Use one of the following parameters to export data for a particular interval.

`day`	single day
`week`	single week, Monday through Sunday
`month`	single month, first through last calendar day
`year`	single year, first through last calendar day

If no parameters are used with the **export** command, the interval used for the exported data is set by the **start** and **stop** commands.

How to Use It

There are four ways to specify a particular interval (`day`, `week`, `month`, `year`).

- Current interval

  Specify the keyword only. (For example: `month` means the current month)

- Previous interval

Specify the keyword, a minus, and the number of intervals before the current one desired. For example, `day-1` is yesterday, `week-2` is two weeks prior to the current week.

- Absolute interval

  Specify the keyword and a positive number. The number indicates the absolute interval desired in the current year. For example, `day 2` is January 2 of the current year.

- Absolute interval plus year

Specify the keyword and a large positive number. The number should consist of the last two digits of the year and the absolute interval number in that year. In this format the absolute day would have 5 digits (94002) means January 2, 1994) and all other keywords would have four digits (month 9404 means April of 1994).

If not previously specified, the **logfile** command defaults to `logglob` in the *datafiles* directory and the **report** command defaults to `reptfile` in the *configuration* directory.

Note

Throughout this manual, the names *configuration* and *datafiles* are used to indicate platform-specific directories in MeasureWare Agent. See the **man** page for **scopeux** to see which of these directories is applicable to your MeasureWare Agent system.

The settings or defaults for all other parameters are used. For details on their actions, see descriptions of the **application, configuration, global, process, disk, lvolume, netif, transaction, output, shift, start**, and **stop** commands.

Note The **application** and **process** commands are used only in MeasureWare Server Agent.

The **export** command creates up to 14 different files based on the types of data and level of summarization selected.

xfrdGLOBAL.ext	Global detail data file
xfrsGLOBAL.ext	Global summary data file
xfrdAPPLICATION.ext	Application detail data file
xfrsAPPLICATION.ext	Application summary data file
xfrdPROCESS.ext	Process detail data file
xfrdDISK.ext	Disk device detail data file
xfrsDISK.ext	Disk device summary data file
xfrdLVOLUME.ext	Volume detail data file
xfrsLVOLUME.ext	Volume summary data file
xfrdNETIF.ext	Netif detail data file
xfrsNETIF.ext	Netif summary data file
xfrdTRANSACTION.ext	Transaction detail data file
xfrsTRANSACTION.ext	Transaction summary data file
xfrdCONFIGURATION.ext	Configuration detail data file

where `ext = asc, dat, bin,` or **wk1**

Note The `application` and `process` data types are used only in MeasureWare Server Agent.

The default file names are created from the class name. The prefix is either `xfrd` or `xfrs` depending if the data is detailed or summary data. The extension is the specified `asc (ascii), bin (binary), dat (datafile),` or **wk1**

(spreadsheet) data format. For example, classname = ACCTG_INFO would have export file names of:

xfrdACCTG_INFO.wk1	detailed wk1 data for ACCT_INFO
xfrsACCTG_INFO.asc	summarized ascii data for ACCT_INFO

For more information about exporting data, see "Overview of the Export Function" in this chapter.

Example

In this example, you export log file data collected yesterday (8:00 AM to 5:00 PM) selecting all global metrics because no report file has been specified.

```
extract>
start today-1 8:00 am
stop  today-1 5:00 pm
global summary
export
exit
```

EXTRACT

Use the **extract** command to start the process of copying data into an extracted file format. Extracted files can be used for archiving or for analysis by analyzer programs.

Note

The **extract** command cannot be used to process data from DSI log files.

Syntax

```
extract [ day [dd] [yyddd] [-days]
        | week [ww] [yyww] [-weeks]
        | month [mm] [yymm] [-months]
        | year [yy] [yyyy] [-years] ]
```

Parameters

Use one of the following parameters to extract data for a particular interval:

day	single day
week	single week, Monday through Sunday
month	single month, first through last calendar day
year	single year, first through last calendar day

If no parameters are used with the extract command, the interval used for data extraction is set by the **start** and **stop** commands.

How to Use It

There are four ways to specify a particular interval (**day**, **week**, **month**, **year**).

- Current interval

Specify the keyword only (**month** means the current month).

- Previous interval

  Specify the keyword, a minus, and the number of intervals before the current one desired. For example, **day-1** is yesterday, **week-2** is two weeks prior to the current week.

- Absolute interval

  Specify the keyword and a positive number. The number indicates the absolute interval desired in the current year. For example, **day 2** is January 2 of the current year.

- Absolute interval plus year

  Specify the keyword and a large positive number. The number should be composed of the last two digits of the year and the absolute interval number in that year. In this format, the absolute day would have 5 digits (95002 means January 2, 1995) and all other keywords would have four digits (month 9504 means April of 1995).

The **extract** command starts data extraction. If not specified, the **logfile** and **output** commands assume the following defaults when the **extract** command is executed:

logfile	**logglob** (in the *datafiles* directory)
output	**rxlog,new**

The settings or defaults for all other parameters are used. For details on their actions, see descriptions of the **application, global, process, disk, lvolume, netif, transaction, shift, start,** and **stop** commands.

Note

The **application** and **process** commands are used only in MeasureWare Server Agent.

Examples

In the first example, you extract data from raw log files using the **extract** command's default settings. Extract the last 30 full days of global detail data from the default log file.

```
rm -f rxlog
extract>
extract
quit
```

In the second example, you copy data from one extracted log file (`rxjan`) to another (`rxsum`), and summarize global detail data into hourly summary data. Assume that **extract** is running.

```
logfile rxjan
output rxsum
global summary
extract
```

In the third example, you append data from raw log files to the existing `rxsum` extracted log file. Original specifications for extracting data from the `rxsum` file are maintained. Data extraction is stopped on February 28, 1995.

```
logfile logglob
output rxsum,append
stop 02/28/95
extract
```

In the fourth example, you create a new extracted log file called `rxjan95`. Purge any existing file that has this name. Extract all raw log file data, details, and summaries from January 1, 1995 to January 31, 1995.

```
logfile logglob
output rxjan95,purge
start 01/01/95
stop  01/31/95
global both
application both
transaction both
process detail
disk both
lvolume both
extract
```

In the last example, you extract data from the raw log files for January 1, 1995 to December 31, 1995. Extract only global and application summary data from 8:00 AM to 5:00 PM, excluding Saturday and Sunday. The **logfile** command defaults to `logglob` in the *datafiles* directory, so it is not provided.

```
output alldata,purge
start 01/01/95
stop 12/31/95
shift 8:00 am - 5:00 pm noweekends
global summary
application summary
extract
```

FORMAT

Use the **format** command to display the date and time in readable format in exported files.

Syntax

```
format
```

Example

In this example, you export data from raw log files collected since yesterday. Headings are on in the report file **myrept** and the date and time are displayed in readable format in the exported files.

```
extract>
logfile logglob
start today-1
report myrept
heading on
format
global summary
export
quit
```

GLOBAL

Use the **global** command to specify the amount of global data to be extracted or exported.

The default is **global detail.**

Syntax

```
          [ on      ]
          | detail  |
global    | summary |
          | both    |
          [ off     ]
```

Parameters

`detail` or `on` — The `detail` or `on` parameter specifies that raw detail collected at 5-minute intervals is to be extracted or exported.

Detail data must be extracted if you want to draw PerfView global graphs with points every 5 minutes.

`summary` — The `summary` parameter specifies that data should be summarized by:

- the number of seconds specified with the **interval** command (export only)
- the number of minutes specified with the `SUMMARY` option in the selected report file (export only)
- the default summary interval of one hour (export or extract)

Summarization can significantly reduce the size of the resulting extracted or exported data, depending on the summarization interval used. For example, hourly summary data is about

one-tenth the size of 5-minute detail data.

Summarized data is graphed by PerfView more quickly since fewer data records are needed to produce a graph.

If only global summaries are extracted, PerfView global graphs cannot be drawn with data points every 5 minutes.

`both` — The `both` parameter specifies that detail data *and* summary data are to be extracted or exported.

This option maintains the access speed gained with the hourly summary records while permitting you to draw PerfView global graphs with points every 5 minutes.

The disk space required to extract both detail and summary data is about 8 percent more than the space needed for extracting detail data alone.

`off` — The `off` parameter specifies that no global data is to be extracted or exported.

Note

The `off` parameter is not recommended with the current PerfView product because you must have global data to properly understand overall system behavior. PerfView global graphs cannot be drawn unless the extracted file contains at least one type of global data.

Example

In this example, you are specifying no global detail because you don't want global detail data exported. (By default, **global detail** is set in interactive mode.)

```
extract>
logfile logglob
global off
transaction detail
report myrept
export
quit
```

GUIDE

Use **guide** to enter guided commands mode.

Syntax

```
guide
```

How to Use It

The guided command interface guides you through the various **extract** commands and prompts you to perform the most common functions that are available. It also guides you through the process of creating an export report file.

Guided mode does not provide all possible combinations of parameter settings. It selects settings that should produce useful results for the majority of users. You can obtain full control over **extract**'s functions through the regular command interface.

- To enter guided commands mode from the regular command interface, type `guide`.
- To accept the default value for a parameter, press Return.
- To terminate guided commands mode and return to the regular command interface, type `q` at any `extract guide>` prompt.

HEADING

Use the **heading** command to set headings on and off in the report that is printed with exported data.

Syntax

```
heading [on]
        [off]
```

Example

In this example, you export data from raw log files collected since five days ago. The report file contains headings and the date and time are displayed in readable format in the exported files.

```
extract>
logfile logglob
start last-5
report myrept
heading on
format
global summary
export
quit
```

Note

Headings can also be controlled using the HEADINGS parameter in the report file. However, if both the **heading** command and the HEADINGS parameter are specified, the **heading** command takes precedence.

HELP

Use the **help** command to access the online `help` facility. You enter parameters to obtain information on **extract** commands and tasks or on **help** itself.

Syntax

`help` [*topic*]

How to Use It

You can navigate to different topics by entering a key word. If more than one page of information is available, the display pauses and waits for you to press Return before continuing. To exit the `help` system and return to **extract**, type `q` or `quit`.

You can also request help on a specific topic. For example,

```
help tasks
```
or
```
help shift
```

When you use this form of the **help** command, you receive the help text for the specified topic and remain in the **extract** command entry context. Since you do not enter the `help` subsystem interactively, you do not have to type `quit` before entering the next **extract** command.

INTERVAL

Use the **interval** command to set the interval (in seconds) used to summarize data as it is exported. The summarization is in addition to the summarization performed when the data was logged.

Syntax

```
interval [seconds]
```

Example

In this example, you export raw log file data collected since the day before yesterday. Headings are on and the interval is set to one hour.

```
extract>
logfile logglob
start today-2
report myrept
heading on
interval 3600
global summary
export
quit
```

Note

The summarization interval can also be controlled using the **SUMMARY** parameter in the report file. However, if you specify both the **interval** command and the **SUMMARY** parameter, the **interval** command takes precedence.

LIST

Use the **list** command to specify the list file for all **extract** program reports.

Syntax

```
list [file]
     [*  ]
```

How to Use It

You can use **list** at any time while using **extract** to specify the list device. It uses a file name or list device name to output the user-specified settings. If the list file already exists, the output is appended to it.

The data that is sent to the list device is also displayed on your screen.

While **extract** is running, type:

```
list outfilename
```

To return the listing device to the user terminal, type:

```
list stdout
```

or

```
list *
```

To determine the current list device, type the **list** command without parameters as follows:

```
list
```

If the list file is not `stdout`, most commands are echoed to the list file as they are entered.

Example

In this example, you set the list device to **mylist**. The results of the next commands are printed to **mylist** and displayed on your screen.

```
extract>
list mylist
global detail
shift 8:00am - 5:00pm
extract
quit
```

LOGFILE

Use the **logfile** command to open a log file. You must open a log file for all **extract** program functions. You can do this explicitly, by issuing the **logfile** command, or implicitly, by issuing the **extract** command or **export** command. If a log file name is not provided, the default log file (`logglob`) in the *datafiles* directory is used.

Syntax

`logfile` [*logfile*]

How to Use It

To open a log file, you can specify the name of either a raw or extracted log file. You cannot specify the name of a file created by the **export** command. If you specify an *extracted* log file name, all information is obtained from this single file. If you specify a *raw* log file name, you must specify the name of the global log file before you can access the raw log file. This is the *only* raw log file name you should specify.

If the log file is not in your current working directory, you must provide its path.

The global log file and other raw log files must be in the same directory. They have the following names:

`logglob`	global log file
`logappl`	application log file
`logproc`	process log file
`logdev`	device log file
`logtran`	transaction log file
`logindx`	index log file

Note `logappl` and `logproc` log files are used only in MeasureWare Server Agent.

The general contents of the log file are displayed when the log file is opened.

Caution Do not *rename* raw log files! When accessing these files, the program assumes that the standard log file names are in effect. If you must rename log files to place log files from multiple systems on the same system for analysis, you should first *extract* the data and then rename the extracted log files.

Example

The following example is a listing of raw log files that are open.

```
Global      file: logglob
Application file: logappl
Process     file: logproc
Device      file: logdev
Index       file: logindx

System ID: Homer
System Type  7013/20 S/N 0005 O/S 4.2.1
Data Collector: UX A.09.01
Data Covers:   18 days to 05/18/95
File Created:  05/00/95
Shift is:      All Day
Data records available are:
     Global Application Process Disk Volume
Maximum file sizes:
     Global=2.0 Application=5.0 Process=5.0 Device=10.0 Transaction=0.0 MB

The first GLOBAL      record is on 05/01/95 at 10:50 AM
The first NETIF       record is on 05/01/95 at 11:50 AM
The first APPLICATION record is on 05/08/95 at 12:00 AM
The first PROCESS     record is on 05/17/95 at 12:03 AM
The first DEVICE      record is on 05/01/95 at 11:50 AM
The Transaction data file is empty
The default starting date & time = 05/01/95 11:50 AM   (LAST    -30)
The default stopping date & time = 05/18/95 11:59 PM   (LAST  -   0)
```

You can verify which log file you opened with the **show** command, described later.

You can open another log file at any time by entering another **logfile** command. Any currently opened log file is closed before a new log file is opened.

LVOLUME

Use the **lvolume** command to select the type of logical volume data that is being extracted or exported.

The default is **lvolume off**.

Syntax

```
lvolume [ on      ]
        | detail  |
        | summary |
        | both    |
        [ off     ]
```

Parameters

on or detail The on or detail parameter specifies that raw, 5-minute detail data should be extracted or exported.

summary The summary parameter specifies that data should be summarized by:

- the number of seconds specified with the **interval** command (export only)
- the number of minutes specified with the SUMMARY option in the selected report file (export only)
- the default summary interval of one hour (export or extract)

Summarization can significantly reduce the size of the resulting extracted or exported data, depending on the summarization interval used. For example, hourly summary data is about one-tenth the size of 5-minute detail data.

`both`	The `both` parameter specifies that detail data and summary data are to be extracted or exported.
`off`	The `off` parameter specifies that no logical volume data is to be extracted or exported.

Example

In this example, you create a new extracted log file called rx295 and purge any existing file that has that name. Extract all logical volume log file data from February 1, 1995 through February 28, 1995.

```
extract>
logfile logglob
output rx295,purge
start 02/01/95
stop 02/28/95
global both
lvolume both
extract
quit
```

MENU

Use the **menu** command to print a list of the available **extract** commands.

Syntax

```
menu
```

The example on the next page shows a list of the available commands if you are using MeasureWare Server Agent. (If you are using MeasureWare Desktop Agent, the **application** and **process** commands are excluded from the list.)

```
Command   Parameters   Function
HELP      [topic]      Get information on commands and options
GUIDE     Enter guided commands mode for novice users
LOGFILE   [logname]    Specify a log file to be processed
LIST      [filename/*] Specify the listing file
OUTPUT    [filename][,NEW/PURGE/APPEND] Specify a destination file
REPORT    [filename]],SHOW]   Specify a REPORT Format file for "EXPORT"
GLOBAL       [DETAIL/SUMMARY/BOTH/OFF] Extract GLOBAL         records
APPLICATION  [DETAIL/SUMMARY/BOTH/OFF] Extract APPLICATION    records
PROCESS      [DETAIL/OFF/KILLED][APP=] Extract PROCESS        records
DISK         [DETAIL/SUMMARY/BOTH/OFF] Extract DISK DEVICE    records
LVOLUME      [DETAIL/SUMMARY/BOTH/OFF] Extract Logical VOLUME records

NETIF        [DETAIL/SUMMARY/BOTH/OFF] Extract Logical NETIF  records
CONFIG       [DETAIL/OFF]              Export  CONFIGURATION  records
CLASS     classname[DETAIL/SUMMARY/BOTH/OFF] Export classname records
TRANSACTION  [DETAIL/SUMMARY/BOTH/OFF] Extract TRANSACTION    records

START     [startdate time] Specify a starting date and time for SCAN
STOP      [stopdate  time] Specify an ending  date and time for SCAN
SHIFT     [starttime - stoptime] [NOWEEKENDS] Specify daily shift times
SHOW      [ALL]            Show the current program settings

EXPORT    [d/w/m/y][-offset] Copy log file records to HOST format files
EXTRACT   [d/w/m/y][-offset] Copy selected records to output (or append) file
WEEKLY    [ww/yyww]  Extract one calendar  week's data with auto file names
MONTHLY   [mm/yymm]  Extract one calendar month's data with auto file names
YEARLY    [yy/yyyy]  Extract one calendar  year's data with auto file names

TERSE     [ON/OFF]  Set level of prompting and output details

SEPARATOR [char_separator] Used to set column separator used when exporting
INTERVAL  [seconds]  Sets the interval in seconds to be used when exporting
BEGIN     [start time UNIX format] UNIX start time used when exporting
END       [stop time UNIX format]  UNIX stop  time used when exporting
HEADING   [ON/OFF]  Set headings on/off in report exported
MISSING   [ON/OFF]  Set display of missing exported data on or off
NOVALUE   [value]   Set null indicator for missing exported value
WEEKDAYS  [1...7]   Set days to exclude from export 1=Sunday
UNIX                Format date/time of exported data in UNIX format
FORMAT              Format date/time in readable format
! or SH   [command] Execute a system command
MENU or ? List the command menu (this listing)
EXIT or Q Terminate the program
```

MISSING

Use the **missing** command to set on or off the display of missing exported records.

Syntax

missing [*on* / *off*]

How to Use It

By default, missing exported records are not displayed. To display missing records, specify **missing on**.

Example

In this example, you export raw log file data collected from seven days prior to the last date in the log file. In the report file **myrept**, headings are on by default and records with all missing values are not displayed.

```
extract>
logfile logglob
start last-7
report myrept
missing off
global summary
export
quit
```

Note

Display of missing records can also be controlled using the **MISSING** parameter in the report file. However, if you use both the **missing** command and the **MISSING** report file parameter, the **missing** command takes precedence.

MONTHLY

Use the **monthly** command to specify data extraction based on a calendar month. During execution, this command sets the start and stop dates to the proper dates, based on the month and year of the data extracted.

The name of the output file consists of the letters `rxmo` followed by the last two digits of the year and the two-digit number of the month being extracted. For example, March 1995 would be output to a file named `rxmo9503`.

Syntax

`monthly` [*yymm* | *mm*]

Parameters

`monthly`	Extracts data from the current (default) month.
`monthly` *mm*	Extracts data for a specific month from the current year's data (where *mm* is a number from 01 to 12).
`monthly` *yymm*	Extracts data for a specific month *and* year (where *yymm* is a single number consisting of the last two digits of the year and the two-digit month number). For example, to extract data for February 1995, specify `monthly 9502`.

If you do not specify the log file before executing the **monthly** command, the default log file is `logglob` in the *datafiles* directory.

How to Use It

The type of data extracted and summarized follows the normal rules for the **extract** command and can be set

before executing the **monthly** command. These settings are honored unless a monthly output file already exists. If it does, data is appended to it based on the type of data that was originally selected.

The **monthly** command has a special feature. It opens the *previous* month's extracted file and checks to see if it is filled—whether it contains data extracted up to the last day of the month. If not, the **monthly** command appends data to this file to complete the previous month's extraction.

For example, a **monthly** command is executed on May 7, 1995. This creates a log file named `rxmo9505` containing data from May 1 through the current date (May 7).

On June 4, 1995, another **monthly** command is executed. Before the `rxmo9506` file is created for the current month, the `rxmo9505` file from the previous month is opened and checked. When it is found to be incomplete, data is appended to it to complete the extraction through May 31, 1995. Then, the `rxmo9506` file is created to hold data from June 1, 1995 to the current date (June 4).

As long as you execute the **monthly** command at least once a month, this feature will complete each month's file before creating the next month's file. When you see two adjacent monthly files—for example, `rxmo9505` (May) and `rxmo9506` (June)—you can assume safely that the first file is complete for that month, and it can be archived and purged.

Note

The **monthly** and **extract month** commands are similiar in that they both extract one calendar month's data. The **monthly** command ignores the setting of the **output** command, using instead predefined output file names. It also attempts to append missing data to the previous month's extracted log file if it is still present on the system. The **extract month** command, on the other hand, uses the settings of the **output** command. It cannot append data to the previous month's extracted file since it does not know its name.

Example

In this example, you extract raw log file data from May 1995 (year 95 month 05).

```
extract>
logfile logglob
global off
application detail
monthly 9505
quit
```

NETIF

Use the **netif** command to specify the type of logical netif (LAN) data that is being exported.

This command is used only when exporting data. Netif data is *always* extracted when global data is specified. In other words, to extract netif data, specify `global detail`, `global both` or `global summary`.

The default is **netif off**.

Syntax

```
          [ on      ]
          | detail  |
netif     | summary |
          | both    |
          [ off     ]
```

Parameters

`on` or `detail` — The `on` or `detail` parameter specifies that raw, 5-minute detail data should be exported.

`summary` — The `summary` parameter specifies that data should be summarized by:

- the number of seconds specified with the **interval** command (export only)
- the number of minutes specified with the `SUMMARY` option in the selected report file (export only)
- the default summary interval of one hour (export or extract)

Summarization can significantly reduce the size of the resulting extracted or exported data, depending on the summarization interval used. For

example, hourly summary data is about one-tenth the size of 5-minute detail data.

`both`	The `both` parameter specifies that detail data and summary data are to be extracted or exported.
`off`	The `off` parameter specifies that no netif data is to be extracted or exported.

Example

In this example, you export netif data that was collected in June, 1995.

```
extract>
logfile logglob
global off
netif detail
export month 9506

quit
```

NOVALUE

Use the **novalue** command to set a null indicator for a missing value when exporting data.

Syntax

```
novalue  [value]
```

Example

In this example, you export raw log file data collected five days prior to the last date in the log file. Missing values are set to -999.

```
extract>
logfile logglob
start last-5
report myrept

missing on
novalue -999

global summary
export
quit
```

Note The missing data value can also be controlled using the `MISSING` parameter in the report file.

OUTPUT

Use the **output** command to specify the name of the output file for the **extract** or **export** functions.

The optional second parameter specifies the action to be taken if an output file with the same name exists.

Syntax

output [*filename*] [,new | ,purge | ,append]

Parameters

,new	The ,new parameter specifies that the output file must be a new file. This is the default action in batch mode. If a file with the same name exists, the batch job terminates.
,purge	The ,purge parameter specifies that any existing file should be purged to make room for the new output file.
,append	The ,append parameter specifies that an existing extracted file should have data appended to it. If no file exists with the output file name specified, a new file is created.

How to Use It

If you do not specify an action in batch mode, the default action ,new is used. In interactive mode, you are prompted to enter an action if a duplicate file is found.

The default output file names are:

For **extract:** `rxlog`

For **export:**

```
xfrdGLOBAL.ext
xfrsGLOBAL.ext
xfrdAPPLICATION.ext
xfrsAPPLICATION.ext
xfrdPROCESS.ext
xfrdDISK.ext
xfrsDISK.ext
xfrdLVOLUME.ext
xfrsLVOLUME.ext
xfrdNETIF.ext
xfrsNETIF.ext
xfrdTRANSACTION.ext
xfrsTRANSACTION.ext
xfrdCONFIGURATION.ext
```

where `ext = asc (ascii), dat (datafile), bin (binary)`, or `wk1` (spreadsheet).

Note `application` and `process` log files are only supported in MeasureWare Server Agent.

Note You can override the default output file names for exported files using the `OUTPUT` parameter in the report file.

A special file name, `stdout` (or "*"), can be used with the **export** operation to direct the output to the `stdout` file (normally the terminal, although this can be redirected using shell commands).

```
output stdout
```

or

```
output *
```

To return the output to its default settings, type:

```
output default
```

or

```
output -
```

Note

You cannot output **extract** operation files to `stdout`, because they are incompatible with `ASCII` devices. You should also not output `BINARY` or `WK1` formats of the **export** operation to the `stdout` file for the same reason.

Note

Care should be taken to avoid appending *extracted* data to an existing *exported* data file and to avoid appending *exported* data to an existing *extracted file*. Attempts to append the wrong data type will result in an error message.

Examples

In this example, you specify the default output file, `rxlog` for application summary data that you are extracting. The `,purge` option specifies that any existing output file should be purged.

```
extract>
logfile logglob
output rxlog,purge
global off

application summary
extract month 9505
quit
```

PROCESS

Use the **process** command to specify whether or not to extract or export process data.

The default is **process off**.

Note

The **process** command is used only in MeasureWare Server Agent.

Syntax

```
process [ on       ]
        [ detail   ] [application=#[-#], ... ]
        [ off      ]
        [ killed   ]
```

Parameters

`on` — The on parameter specifies that process data *should* be extracted or exported.

`detail` — Specifying `process detail` is the same as specifying `process on`.

`off` — The `off` parameter specifies that process data *should not* be extracted or exported.

`killed` — The `killed` parameter specifies only processes that have an interest reason that includes `killed`. (Processes that terminated in the measurement interval.)

`application` — The `application` keyword specifies only processes that belong to selected applications. An application can be entered as a single number or as a range of application numbers (7-9 means applications 7, 8, and 9). The application number is determined by the order of the application definition in the

parm file when the data was collected. If you are specifying multiple applications, separate each one with a comma.

Note

Process data can increase the size of an extracted log file significantly. If you plan to copy the log file to a workstation for analysis, you might want to limit the amount of process data extracted.

Example

The following example specifies processes that terminated in an interval and belong to application 1, 4, 6, 7, 8, or 10.

```
process killed application=1,4,6-8,10
```

Use the **utility** program's **scan** command to find the application numbers for specific applications.

QUIT

Use the **quit** *or* **q** command to end the **extract** program. It is an acceptable alternative to the **exit** *or* **e** command used to end the **extract** program.

Syntax

```
quit
q
```

REPORT

Use the **report** command to specify the report definition file to be used by the **export** function. For the location of the default report file, `reptfile`, see the **man** page for **extract**.

If you are in interactive mode and specify no report file, all metrics for the data types requested will be exported in ASCII format.

Syntax

`report` [*reportfile*] [`,show`]

Parameters

`,show` — The `,show` parameter specifies that the field positions and starting columns should be listed for all metrics contained in the report file. This listing can be used when export files are processed by other programs.

How to Use It

When you issue this command, you are prompted by a message that asks whether or not you want to validate metrics in the report layout with the previously specified log file. Validation ensures that the metrics specified in the report file exist in the log file. This allows you to check for possible errors or typos in the report file. If no validation is performed, this action is deferred to when you perform an export.

Note

The `,show` parameter of the **report** command discussed here is different from the **show** command discussed later.

SEPARATOR

Use the **separator** command to set the column separator that separates metrics during data export.

Syntax

```
separator [char-separator]
```

Parameters

char-separator Can be a comma, a vertical line (|), etc. The default is a blank space.

Example

In this example, you export raw log file data collected from seven days prior to the last date in the log file. In the report file **myrept**, headings are on (default) and the column separator is a vertical line (|).

```
extract>
logfile logglob
start last-7
report myrept
separator "|"
global summary
export
quit
```

Note

You can also specify the column separator using the SEPARATOR parameter in the report file. However, if you use both the **separator** command and the SEPARATOR report file parameter, the **separator** command takes precedence.

SH

Use the **sh** command to enter a shell command without exiting **extract** by typing **sh** or an exclamation point (**!**) followed by a UNIX shell command.

Syntax

sh *or* **!** [*shell command*]

How to Use It

Following the execution of the single command, you automatically return to the **extract** program. If you want to issue multiple shell commands without returning to **extract** after each one, you can start a new shell.

sh ls	Executes the `ls` command and returns to **extract**.
! ls	Same as above.
! ksh	Starts a Korn shell. Does *not* return immediately to **extract**. Type **exit** or CTRL-d Return to return to the **extract** program.

If you omit the command, you are prompted to supply it. For example,

```
sh
enter system command: ls
```

SHIFT

Use the **shift** command to limit data extraction to certain hours of the day corresponding to work shifts and to exclude weekends (Saturday and Sunday). The default is **shift all day**—extract data for all day, every day including weekends.

Syntax

```
shift [ starttime-stoptime ] [noweekends]
      [ all day            ]
```

Parameters

The *starttime* and *stoptime* parameters are entered in the same format as the time in the **start** command. Shifts that span midnight are permitted. If *starttime* is scheduled *after* the *stoptime*, the shift will start at the start time and proceed past midnight, ending at the *stoptime* of the next day.

`all day` The `all day` parameter specifies the default shift of 12:00 AM–12:00 AM (or 00:00–00:00 on a 24-hour clock).

`noweekends` The `noweekends` parameter specifies the exclusion of data which was logged on Saturdays and Sundays. If `noweekends` is entered in conjunction with a shift that spans midnight, the weekend will consist of those shifts that *start* on Saturday or Sunday.

Example

In this example, you extract data from raw log files that was collected between 10:00 am and 4:00 pm on all days starting June 15, 1995.

```
extract>
logfile logglob
global off
disk summary
shift 10:00 am - 4:00 pm
start 6/15/95
extract
quit
```

SHOW

Use the **show** command to list the names of the opened files and the status of the **extract** parameters that can be set.

Syntax

```
show [all]
```

Note

The **show** command discussed here is different from the **,show** parameter of the **report** command discussed earlier.

Examples

Issuing **show** by itself produces a list that may look like this:

```
show
Logfile:  logglob
Output:   Default
Report:   Default
List:     "mylist"

The default shift = 12:00 AM - 12:00 AM

GLOBAL.....DETAIL..... records will be processed
APPLICATION.....NO records will be processed
PROCESS.....NO records will be processed
DISK DEVICE.....NO records will be processed
LVOLUME                NO records will be processed
TRANSACTION... .....NO records will be processed
NETIF.....NO records will be exported
Configuration.....NO records will be exported
```

The entry for **NETIF** only applies to exported data. The entry for **GLOBAL** implies that both global and netif data are extracted.

Note

Application and **process** data types are only supported in MeasureWare Server Agent.

Issuing the optional all parameter causes more information about the log file to be printed.

```
Logfile: logglob

    Global      file: logglob
    Application file: logappl
    Process     file: logproc
    Device      file: logdev
    Index       file: logindx
    System ID:  Homer
    System Type  7013/20     S/N 0005     O/S 4.2.1
    Data Collector: UX A.03.01
    File created:   09/02/95
    Data Covers:    26 to 09/27/95
    Shift Is:       All Day

Data records available are:
   Global Application Process Disk Volume

Maximum file sizes:
   Global=2.0 Application=5.0 Process=5.0 Device=10.0 Transaction 0.0 MB

Output:  Default
Report:  Default
List:    "mylist1"

GLOBAL..........DETAIL..... records will be processed
APPLICATION.................NO records will be processed
PROCESS.....................NO records will be processed
DISK DEVICE.................NO records will be processed
LVOLUME                      NO records will be processed
TRANSACTION... .............NO records will be processed
NETIF.......................NO records will be exported
Configuration...............NO records will be exported

Export report specifications:
   Interval  = 3600, Separator =  " "
   Missing data will not be displayed
   Headings will be displayed
   Date/time will be formatted
   Days to exclude: None
```

START

Use the **start** command to set a starting date and time for the **extract** and **export** functions. The default starting date is the date 30 full days before the last date in the log file, *or* if less than 30 days are present, the date of the earliest record in the log file.

Syntax

```
start [ date [ time ]                          ]
      [ today [ -day ] [ time ]                ]
      [ last [ -days ] [ time ]                ]
      [ first [ +days ] [ time ]               ]
```

Parameters

date — The *date* format depends on the native language that is configured on your system. If you do not use native languages or you have set C as the default language, the date format is *mm/dd/yy* (month/day/year) such as 02/28/95 for February 28, 1995.

time — The *time* format also depends on the native language used. For the C language, the format is *hh:mm* AM or *hh:mm* PM (hour:minute in a 12-hour format with the AM or PM suffix). For example, 07:00 AM is 7 o'clock in the morning.

24 hour time is accepted in all languages. For example, 23:30 would be accepted for 11:30 PM.

If the format of the date or time is unacceptable, you are prompted with an example in the correct format.

If no start time is given, midnight (12:00 AM) is assumed. A starting time of midnight for a given day starts at the *beginning* of that day (00:00 on a 24-hour clock).

`today` — The `today` parameter specifies the current day. The qualification of the parameter, such as `today-`*days*, specifies the number of days *prior to* today's date. For example, `today-1` indicates yesterday's date and `today-2`, the day before yesterday.

`last` — The `last` parameter can be used to represent the last date contained in the log file. The parameter `last-`*days* specifies the number of days *prior* to the last date in the log file.

`first` — The `first` parameter can be used to represent the first date contained in the log file. The parameter `first+`*days* specifies the number of days *after* the first date in the log file.

Note

If you are not sure whether native language support is installed on your system, you can force **extract** to use the C date and time formats by issuing the following statement before you run **extract**:

```
LANG=C ; export LANG
```

or in C Shell

```
setenv LANG C
```

How to Use It

The following commands override the starting date set by the **start** command.

weekly
monthly
yearly
extract (If `day`, `week`, `month`, or `year` keyword is used)
export (If `day`, `week`, `month`, or `year` keyword is used)

Example

In this example, you use the **start** command to specify July 5, 1995 8:00 am as the start time of the first interval to be extracted.

```
extract>
logfile logglob
start 7/5/95 8:00am
output myout
global summary
extract
quit
```

STOP

Use the **stop** command to terminate the **extract** function on a specified date and time.

The default stopping date and time is the *last* date and time recorded in the log files.

Syntax

```
stop [ date [time]
     | today [-days] [time]
     | last [-day] [time]
     | first [+days] [time] ]
```

Parameters

The formats for the **stop** command and **start** command are the same and depend on the language being used.

If no stop time is given, one minute before midnight (11:59 PM) is assumed. A stopping time of midnight (12:00 AM)for a given day stops at the *end* of that day (23:59 on a 24-hour clock).

`today` The `today` parameter specifies the current day. The qualification of the parameter, such as `today-`*days*, specifies the number of days *prior to* today's date. For example, `today-1` indicates yesterday's date and `today-2` the day before yesterday.

`last` The `last` parameter can be used to represent the last date contained in the log file. The parameter `last-`*days* specifies the number of days *prior to* the last date in the log file.

`first` The `first` parameter can be used to represent the first date contained in the log file. The parameter `first+`*days*

specifies the number of days *after* the first date in the log file.

How to Use It

The following commands override the stopping date set by the **stop** command.

weekly
monthly
yearly
extract (If **day, week, month**, or **year** keyword is used)
export (If **day, week, month**, or **year** keyword is used)

Example

In this example, you use the **stop** command to specify July 5, 1995 5:00 pm as the stopping time of the last interval to extract.

```
extract>
logfile logglob
start 7/5/95 8am
stop 7/5/95 5pm
output myout
global summary
extract
quit
```

TERSE

Use the **terse** command to control how much text is displayed on the screen.

Syntax

```
terse [on]
      [off]
```

Parameters

on	Displays text that contains fewer details (the default).
off	Displays text that contains more details.

How to Use It

The **terse** command affects the amount of detail that is displayed in user prompts, as well as output to the screen from the **list** and **show** commands.

Example

In this example, `global summary` text is suppressed by `terse on`, so nothing is displayed.

```
extract>
terse on
global summary
extract
quit
```

TRANSACTION

Use the **transaction** command to select the type of MeasureWare Agent Transaction Tracker data that is being extracted or exported.

Syntax

```
transaction [ on      ]
            [ detail  ]
            [ summary ]
            [ both    ]
            [ off     ]
```

Parameters

on or `detail`	The on or `detail` parameter specifies that raw, 5-minute detail data is to be extracted or exported.
`summary`	The `summary` parameter specifies that raw data is to be summarized into one data point per hour before being extracted or exported.
`both`	The `both` parameter specifies that both 5-minute detail data and hourly summary data is to be extracted or exported.
`off`	The `off` parameter specifies that no transaction data is to be extracted or exported.

Example

In this example, you create a new extracted log file called **rxmay95**. Purge any existing file that has this name. Extract all raw transaction log file data from May 1, 1995 to May 31, 1995.

```
extract>
logfile logglob
output rxmay95,purge
start 05/01/95
stop  05/31/95
global both
transaction both
extract
quit
```

UNIX

Use the **UNIX** command to display the date and time in UNIX format in DSI log files only.

Syntax

```
UNIX
```

UNIX time is always displayed as seconds since 1970.

```
1/1/70 00:00:00 = 0 UNIX time
4/4/95 01:00    = 796953600 UNIX time
```

Example

In this example, you export DSI data collected starting five days prior to the last date in the log file. Headings are on in the report file `myrept` and exported times are in UNIX format.

```
extract>
class acctg_info detail
start last-5
report myrept
unix
export
quit
```

WEEKDAYS

Use the **weekdays** command to exclude data for specific days from being exported (day 1 = Sunday).

Syntax

```
weekdays [1|2|....7]
```

How to Use It

If you want to export data from only certain days of the week, use this command to exclude the days from which you *do not* want data. Days have the following values:

```
Sunday    = 1
Monday    = 2
Tuesday   = 3
Wednesday = 4
Thursday  = 5
Friday    = 6
Saturday  = 7
```

For example, if you want to export data that was logged only on Monday through Thursday, *exclude* data from Friday, Saturday, and Sunday from your export.

Examples

In this example, you exclude data logged on Wednesdays, Thursdays, and Fridays from your export.

```
extract>
logfile logglob
global detail
report myrept
weekdays 456
export
quit
```

In the next example, you exclude data logged on Saturdays and Sundays from your export.

```
extract>
logfile logglob
global detail
report myrept
weekdays 17
export
quit
```

WEEKLY

Use the **weekly** command to specify data extraction based on a calendar week. A week is defined as seven days starting on Monday and ending on Sunday.

During execution, this command sets the start and stop dates to the proper dates, based on the week and year of the extracted data.

Syntax

`weekly` [*yyww* | *ww*]

Parameters

`weekly`	Extracts the current week's data (the default).
`weekly` *ww*	Extracts data for a specific week from *this* year's data (where *ww* is any number from 01 to 53).
`weekly` *yyww*	Extracts data for a specific week *and* year (where *yyww* is a single number consisting of the last two digits of the year and the two-digit week-of-the-year number). For example, the 20th week of 1995 would be `weekly 9520`.

If you do not specify the log file before executing the **weekly** command, it defaults to the `logglob` default file in the *datafiles* directory.

How to Use It

The name of the output file consists of the letters `rxwe` followed by the last two digits of the year and the two-digit week number for the week being extracted. For example, the 12th week of 1995 (from Monday, March 20 to Sunday, March 26) would be output to a file named `rxwe9512`.

The type of data extracted and summarized follow the normal rules for the **extract** command and can be set before executing the **weekly** command. These settings are honored unless a weekly output file already exists. If it does, data is appended to it, based on the type of data selected originally.

The **weekly** command has a special feature. It opens the *previous* week's extracted file and checks to see if it is filled—whether it contains data extracted up to the last day of the week. If not, the **weekly** command appends data to this file to complete the previous week's extraction.

For example, a **weekly** command is executed on Thursday, May 18, 1995. This creates a log file named `rxwe9520` containing data from Monday, May 15 through the current date (May 18).

On Wednesday, May 24, 1995, another **weekly** command is executed. Before the `rxwe9521` file is created for the current week, the `rxwe9520` file from the previous week is opened and checked. When it is found to be incomplete, data is appended to it to complete the extraction through Sunday, May 21, 1995. Then, the `rxwe9521` file is created to hold data from Monday, May 22, 1995 to the current date (May 24).

As long as you execute the **weekly** command at least once a week, this feature will complete each week's file before creating the next week's file. When you see two adjacent weekly files (for example, `rxwe9520` and `rxwe9521`), you can assume safely that the first file is complete for that week, and it can be archived and purged.

Notes

The weeks are numbered based on their starting day. Thus, the first week of the year (week 01) is the week starting on the *first* Monday of that year. Any days before that Monday belong to the last week of the previous year.

The **weekly** and **extract week** commands are similiar in that they both extract one calendar week's data. The **weekly** command ignores the setting of the **output** command, using instead predefined output file names. It also attempts to append missing data to the previous week's extracted log file if it is still present on the system. The **extract week** command, on the other hand, uses the settings of the **output** command. It cannot append data to the previous week's extracted file because it does not know its name.

Example

In this example, you extract the current week's data and complete last week's extracted file, if it is present.

```
extract>
global both
application both
process detail
weekly
quit
```

The output file is named `rxwe` followed by the current year (*yy*) and week of the year (*ww*).

YEARLY

Use the **yearly** command to specify data extraction based on a calendar year.

During execution, the command sets the start and stop dates to the proper dates, based on the year being extracted.

Syntax

`yearly` [*yyyy* | *yy*]

Parameters

`yearly`	Extracts the current year's data (the default).
`yearly` *yy*	Extracts a specific year's data (where *yy* is a number from 00 to 99). The specifications 00 to 60 assume the years 2000 to 2060, whereas 61 to 99 assume the years 1961 to 1999.
`yearly` *yyyy*	To extract a specific year's data (where *yyyy* is the full-year numbered 1961 to 2060).

If you do not specify the log file before executing the **yearly** command, it defaults to the `logglob` file in the *datafiles* directory.

How to Use It

The name of the output file consists of the letters `rxyr` followed by the four digits of the year being extracted. Thus, data from 1995 would be output to a file named `rxyr1995`.

The type of data extracted and summarized follow the normal rules for the **extract** command and can be set before executing the **yearly** command. These settings are

honored unless a yearly output file already exists. If it does, data is appended to it, based upon the type of data selected originally.

The **yearly** command has a special feature. It opens the *previous* year's extracted file and checks to see if it is filled—whether it contains data extracted up to the last day of the year. If not, the **yearly** command appends data to this file to complete the previous year's extraction.

For example, a **yearly** command is executed on December 15, 1994. This creates a log file named `rxyr1994` that contains data from January 1, 1994 to the current date (December 15).

On January 5, 1995, another **yearly** command is executed. Before the `rxyr1995` file is created for the current year, the `rxyr1994` file from the previous year is opened and checked. When it is found to be incomplete, data is appended to it to complete its extraction until December 31, 1994. Then, the `rxyr1995` file is created to hold data from January 1, 1995 to the current date (January 5).

As long as you execute the **yearly** command at least once a year, this feature will complete each year's file before creating the next year's file. When you see two adjacent yearly files (for example, `rxyr1992` and `rxyr1993`), you can assume safely that the first file is complete for that year, and it can be archived and purged.

Notes

The previous paragraph is true *only* if the raw log files are sized large enough to hold *one full year* of data. It would be more common to size the raw log files smaller and execute the **yearly** command more often (such as once a month).

The **yearly** and **extract year** commands are similiar in that they both extract one calendar year's data. The **yearly** command ignores the setting of the **output** command, using instead predefined output file names. It also attempts to append missing data to the previous year's extracted log file if it is still present on the system. The **extract year** command, on the other hand, will use the settings of the **output** command. It cannot append data to the previous year's extracted file since it does not know its name.

Example

In this example, you append data to the existing yearly summary file (or create it, if necessary). Add application and global summary data.

```
extract>
global sum
application sum
process off
yearly
quit
```

A file named **rxyr** followed by the current year (*yyyy*) is used as the output file.

Overview of the Export Function

The **export** command converts MeasureWare Agent raw, extracted, or DSI log file data into exported files. Exported files can be used in a variety of ways, such as reports, custom graphics packages, databases, and user-written analysis programs.

The process is summarized in Figure 4-1.

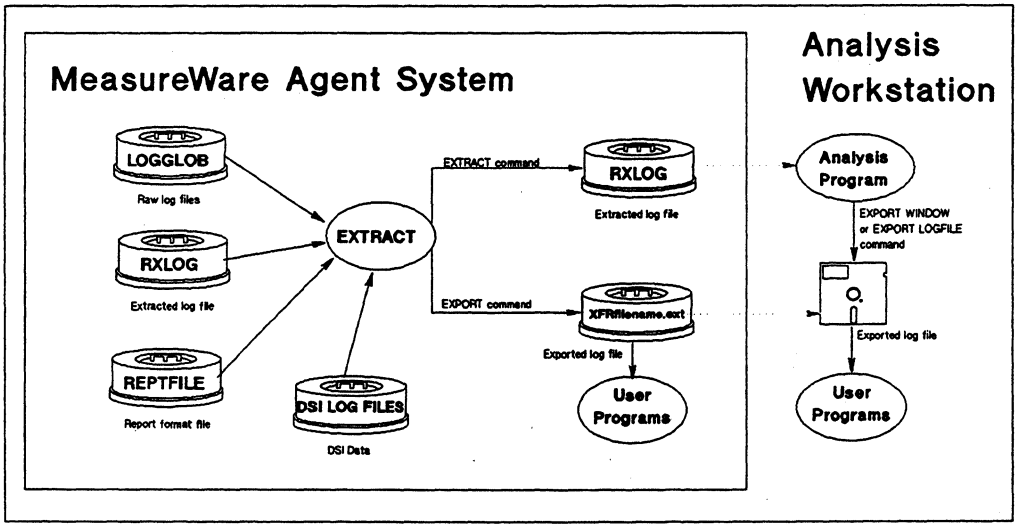


Figure 4-1. The Export Function

How to Export Data

In the simplest form, you can export data by specifying the default log file and the default report file, then starting the export function. The default report file allows you to export files similar to the current PerfView **export** function. Use these commands to export data this way:

```
extract>
logfile
report
export
```

The exported data is placed in a file called `xfrdGLOBAL.asc`, the format suitable for printing.

If you want to export something other than this default set of data, you can use other commands and files in conjunction with the **export** command.

- You can export the following types of data:

`global`	5-minute and hourly summaries.
`application`	5-minute and hourly summaries.
`process`	One-minute details.
`disk device`	5-minute and hourly summaries.
`lvolume`	5 minute and hourly summaries.
`transaction`	5-minute and hourly summaries.
`configuration`	One record containing `parm` file information, and system configuration information, for each time the collector started.
`class`	Intervals and summaries for DSI log files.
`netif`	5-minute and hourly summaries.

Note

`Application` and `process` data types are only supported in MeasureWare Server Agent.

- You can specify what data items are needed for each type of data.
- You can specify starting and ending dates for the data along with the shift and weekend exclusion filters.
- You specify the desired format for the exported data in an `ASCII` report file. This file can be created using any standard host editor program or you can use the default `reptfile` file. (See the **man** page for **extract** for the location of `reptfile`.)

Sample Export Tasks

Two sample report files are furnished with MeasureWare Agent. `repthist` and `reptall`. These files are located in the same directory as `reptfile`. You can use `repthist` and `reptall` to perform common reporting tasks or as a starting point for custom tasks, such as the task described below.

Generating a CPU Report on a Printer.

The `repthist` report file contains the specifications to generate a character graph of CPU and disk usage for the system over time. This graph consists of printable characters that can be printed on any device capable of 132 column printing. The following commands generate a graph of the last seven days and should produce approximately two pages (34 pages if 5-minute detail is selected instead of hourly summaries).

```
extract>
report repthist
global summary
start today-7
export
exit
```

At this point the data is in a file named `xfrsGLOBAL.asc`. To copy it to the printer, type:

```
lp xfrsGLOBAL.asc
```

To remove the file after printing, type:

```
rm -f xfrsGLOBAL.asc
```

Producing a Customized Export File.

If the previous task is similar to what you want to do, make a copy of the report file and customize it using the **extract** program's **guide** command. In guided mode, you copy the `reptall` file from the *configuration* directory and read the **scopeux** or DSI log file specified to dynamically create the list of class and metric names. The report file contains every possible item for each type of data so you need only uncomment those items that are of interest to you. This is easier than retyping the report file.

Note

Using the report with all items selected creates an output file with records too wide for practical use. This is normal and simply indicates that you should select a subset of data to export.

Export Data Files

- If you used the **output** command to specify the name of an output file prior to issuing the **export** command, all exported data will be appended to this single file.
- If you are running the **extract** program interactively and want to export data directly to your terminal or workstation (standard output file), specify **output stdout** prior to issuing the **export** command.
- If the output file is an existing extracted format file, the default output files are used for the **export** function.

- If the output file is set to the default, the exported data is separated into as many as 14 different files depending on the type of data being exported.

The **export** log file names are:

`xfrdGLOBAL.ext`	Global detail data file.
`xfrsGLOBAL.ext`	Global hourly summary data file.
`xfrdAPPLICATION.ext`	Application detail data file.
`xfrsAPPLICATION.ext`	Application hourly summary data file.
`xfrdPROCESS.ext`	Process detail data file.
`xfrdDISK.ext`	Disk device detail data file.
`xfrsDISK.ext`	Disk device hourly summary data file.
`xfrdLVOLUME.ext`	Volume detail data file.
`xfrsLVOLUME.ext`	Volume summary data file.
`xfrdNETIF.ext`	Netif detail data file.
`xfrsNETIF.ext`	Netif summary detail data file.
`xfrdTRANSACTION.ext`	Transaction detail data file.rev
`xfrsTRANSACTION.ext`	Transaction summary data file.
`xfrdCONFIGURATION.ext`	Configuration data file.

where `ext= asc (ascii), bin (binary), dat (datafile),` or `wk1 (spreadsheet).`

Note

`Application` and `process` data types are only supported in MeasureWare Server Agent.

In summary, the following commands affect the actions of the **export** function:

global	**lvolume**	**stop**
application	**class**	**shift**
process	**netif**	**logfile**
configuration	**transaction**	**report**
disk	**start**	**output**

For more information about these commands, see their descriptions in this chapter.

Note

No log file is created *unless* you specify items for the type of data in the report file or issue the proper command (**global, application, process, disk, lvolume, configuration, transaction, class** or **netif**) prior to issuing the **export** command.

Report File Syntax

The report file contains the following information:

```
REPORT "report title"

        [ASCII      ]
        [DATAFILE   ]
FORMAT  [BINARY     ]
        [WK1 or     ]
        [SPREADSHEET]

HEADINGS [ON ]
         [OFF]

SEPARATOR="char"

SUMMARY=value

MISSING=value

LAYOUT = SINGLE|MULTIPLE

OUTPUT = filename

DATA TYPE datatype

items
```

Parameters

REPORT Specifies the title for the report. See the following section, "Report Title," for more information.

FORMAT Specifies the data formats.

ASCII

ASCII format is best for copying files to a printer or terminal. It does not enclose fields with double quotes.

DATAFILE

The **DATAFILE** format is similar to **ASCII** format but all non-numerical fields are enclosed in double quotes. Since double quotation marks prevent strict column alignment, files in **DATAFILE** format are not recommended for printing directly. The **DATAFILE** format is the easiest format to import into most spreadsheets and graphics packages.

BINARY

The **BINARY** format is more compact because numerical values are represented as binary integers. It is the most suitable format for input into user-written analysis programs because it needs the least conversion, and it maintains the highest metric accuracy. It is not suitable for printing directly.

WK1 or **SPREADSHEET**

The **WK1** or **SPREADSHEET** format is the WK1 Lotus-1-2-3 file format.

HEADINGS Specifies whether or not to include column headings in the report file. If **HEADINGS OFF** is specified, no headings are written to the file. The first record in the file is exported data. If **HEADINGS ON** is specified, **ASCII** and **DATAFILE** formats have the report title plus column headings for each column of metrics written before the first data records. Headings in **BINARY** formats contain the description of the metrics in the file. **WK1** formats always contain column headings.

SEPARATOR	Specifies the character that is printed between each field in the **ASCII** and **DATAFILE** formats. The default separator character is a blank space, but many programs prefer a comma as the field separator. You can set the separator to any printing or nonprinting character.
SUMMARY	Specifies the number of minutes for each summary interval. The value determines how much time is included in each record for summary records. The default interval is 60 minutes. The summary value can be set between 5 and 1440 minutes.
MISSING	Specifies the data value to be used in place of missing data. The default value for missing data is zero. You can select another value in order to differentiate missing from zero data. Missing records are, by default, excluded from exported data. To display missing exported records, use the **missing** command in **extract** or the command line argument **-m**.
LAYOUT	Specifies either single or multiple layouts per record output for multi-instance data types such as **application, disk**, or **lvolume**. Single layout writes one record for every disk or logical volume that was active during the time interval. Multiple layout writes one record for each time interval, with part of that record reserved for each configured disk or logical volume.
OUTPUT	Specifies where exported data is to be output. This can be overridden by specifying the name interactively. It can be specified for each class or data type it is exporting by placing **OUTPUT** *filename* just after the line indicating the *data type* that starts the list of exported data items. Any valid file name can be specified with **OUTPUT**.

`DATA TYPE`	Selects one of the exportable data types: **`global`, `application`, `process`, `disk`, `lvolume`, `transaction`, `configuration`, `netif`** or **`class`** name. This starts a section of the report file that lists the data items to be copied when this type of data is exported.
items	Specifies the data to be included in the exported file. Item names are listed, one per line, in the order you want them listed in the resulting file. You must select the proper *data type* before listing *items*. The same report file can include item lists for as many data types as you want. Each *data type* will be referenced *only if* you choose to export that type of data.

Note

`Application` and `process` data types are only supported in MeasureWare Server Agent.

The `OUTPUT` and `LAYOUT` parameters can be used more than once within a report file. For example, the following are valid commands:

```
data type disk
  layout multiple
  output mydiskfile
     (disk metrics ...)

data type lvolume
  layout single
  output myvolume
     (lvolume metrics ...)
```

Note

You cannot specify different layouts within a single data type. For example, you cannot specify `data type disk` once with `layout = multiple` and again with `layout = single` within the same report file.

You can have more than one report file on your system. Each one can define a set of exported file formats to suit a particular need. You specify the report file to be used with the **report** command when you run the **extract** program.

Report Title

The following items can be substituted in the `report title` string:

`!date`	The date the **export** function was performed.
`!time`	The time the **export** function was performed.
`!logfile`	The fully qualified name of the source log file.
`!collector`	The name and version of the collector program.
`!system_id`	The identifier of the system that collected the data.

For example, the string

```
report "!system_id data from !logfile on !date !time"
```

generates a report title similar to

```
Barkley data from logglob on 02/02/95 08:30 AM
```

Example of Exporting Data

Suppose you want to create a custom graph or report containing exported `global` and `application` data. You would do the following:

1. Determine what data items you need from each data type and in what format you will access them.

   For this example, you want an `ASCII` file without headings and with fields separated by commas.

2. Create and save the following `ASCII` file. Call it `report1`.

```
REPORT "sample report file (report1)"
FORMAT ASCII
HEADINGS OFF
SEPARATOR=","


DATA TYPE GLOBAL
   GBL_CPU_TOTAL_UTIL
   GBL_DISK_PHYS_IO_RATE

DATA TYPE APPLICATION
   APP_CPU_TOTAL_UTIL
   APP_DISK_PHYS_IO_RATE
   APP_ALIVE_PROCESSES
```

3. Run the **extract** utility.

```
>extract
```

4. Issue the **report** command to specify the report file generated.

```
report report1
```

5. Specify `global summary` data and `application summary` data using the **global** and **application** commands.

```
global summary
```

```
application summary
```

6. Issue the **export** command to tell the procedure to "go".

```
export
```

7. Because you didn't tell the program from where it should get the performance data, you are prompted to do so. In this example, since the default log file is correct, just press (Return).

   The output looks like this:

```
exporting global data .........50%.......100%
exporting application data ....50%.......100%

The exported file contains 31 days of data from 09/01/95 to 10/01/95

                          examined  exported
data type                  records   records     space
-----------------------  ---------- --------- ---------
global      summaries                    736   0.20 Mb
application summaries                   2560   0.71 Mb
                                              ---------
                                               0.91 Mb
```

This completes the creation of the custom report.

The two files you have just created—`xfrsGLOBAL.asc` and `xfrsAPPLICATION.asc`—contain the global and application summary data in the specified format.

Resulting Exported Files

Exported files are created with the following characteristics:

- Maximum number of bytes in each record: 32000.
- Maximum disks supported in the disk data type fields: 256.
- Maximum LANs supported: 4.
- Maximum applications supported: 32.

The contents of each file are:

`report title line`	If `report title` and `headings on` were specified.
`names (application, disk, or lvolume)`	If `headings on` was specified along with a multiple layout applications file.
`heading line1`	If `headings on` was specified.
`heading line2`	If `headings on` was specified
`first data record`	
`second data record`	

Report title and heading lines are not repeated in the file.

Notes on ASCII and DATAFILE Formats

The data in these format files should be in printable `ASCII` format. `ASCII` and `DATAFILE` formats are identical except that in the latter, all non-numeric fields are enclosed with double quotes. Even the `DATAFILE` header information is enclosed with double quotes.

The `ASCII` file format does not enclose fields with double quotes. Therefore, the data in `ASCII` files will be properly aligned when printed.

Numerical values are formatted based on their range and internal accuracy. Since all fields will not be the same length, be sure to select the separator you want to use to start each field.

The user-specified separator character (or the default blank space) separates the individual fields in ASCII and DATAFILE formats. Blank spaces, used as separators, can be visually more attractive if you plan to print the report. Other characters can be more useful as separators if you plan to read the report file with another program.

Using the comma as a separator is acceptable to many applications, but some data items may contain commas *that are not separators*. These commas can confuse analysis programs. The date and time formats can contain different special characters based on the native language specified when you execute the **extract** program.

Note

To use a nonprinting special character (such as Tab) as a separator, enter it into your report format file immediately following the first double quote in the **separator** parameter. You might have to turn on the Display Functions feature before your terminal can enter the character properly.

Hints

- Most spreadsheets accept files in DATAFILE format using `separator=","`.
- Many spreadsheet packages accept a maximum of 256 columns in a single sheet. Use care when exporting multiple layout types of data because it is easy to generate more than 256 total items. You can use the output of the `report` *reportfile*`,show` command to determine if you are likely to see this problem.
- If you have a printer that supports underlining, you can create a more attractive printout by specifying ASCII format and the vertical bar character (`separator="|"`) and then printing the report with underlining turned on.

Notes on BINARY Format

In BINARY format files, numerical values are written as 32-bit integers. This can save space by reducing the overall file size, but your program must be able to read BINARY files. We do not recommend copying a BINARY format file to a printer or a terminal.

In BINARY format, non-numerical data is written the same as it was in the ASCII format except separator characters are not used. To properly use a BINARY format file, you should use the record layout report printed by **extract** when you specify `report` *reportfile*`,show`. This report gives you the starting byte for each item specified.

To maintain maximum precision and avoid nonstandard, BINARY floating-point representations, all numerical values are written as scaled, 32-bit integers. Some items might be multiplied by a constant before they are truncated into integer format.

For example, the number of seconds the CPU was used is multiplied by 1000 before being truncated. To convert the value in the exported file back to the actual number of seconds, divide it by 1000. For ease in conversion, specify `headings on` to write the scale factors to the exported file. The report title and special header records are written to BINARY format files to assist in programmatic interpretation.

Binary Header Record Layout

Each record in a BINARY format exported file contains a special 16-byte record header preceding any user-specified data. The `report` *reportfile*`,show` command includes the following four fields that make up this record header:

Table 4-4. Binary Record Header Format

Binary Record Header Metrics	
Record Length	Number of bytes in the record, including the 16 byte record header.
Record ID	A number to identify the type of record (see below).
Date_Seconds	Time since January 1, 1970 (in seconds).
Number_of_vars	Number of repeating entries in this record.

The Record ID metric uniquely identifies the type of data contained in the record. Current Record ID values are:

```
 -1   Title Record
 -2   First header Record      (Contains Item Numbers)
 -3   Second header Record     (Contains Item Scale Factors)
 -4   Application Name Record (for Multiple Instance Application Files)
 -5   Transaction Name Record (for Multiple Instance Transaction Files)
 -7   Disk Device Name Record (for Multiple Instance Disk Device Files)
 -8   Logical Volume Name Record (for Multiple Instance LVolume Files)
 -9   Netif Name Record        (for Multiple Instance Netif Files)

  1   Global Data Record       ( 5 minute detail  record)
101   Global Data Record       (60 minute summary record)
  2   Application Data Record  ( 5 minute detail  record)
      (Not applicable in MeasureWare Desktop Agent)
102   Application Data Record (60 minute summary record)
      (Not applicable in MeasureWare Desktop Agent)
  3   Process Data Record      ( 1 minute detail record)
      (Not applicable in MeasureWare Desktop Agent)
 20   Configuration Data Record
  7   Disk Device Data Record ( 5 minute detail record)
107   Disk Device Data Record (60 minute summary record)
  8   Logical Volume Data Record  ( 5 minute detail record)
108   Logical Volume Data Record  (60 minute summary record)

  4   Transaction Data Record ( 5 minute detail record)
104   Transaction Data Record (60 minute summary record)
 11   Netif Data Record        ( 5 minute detail record)
111   Netif Data Record        (60 minute summary record)

ClassID       Class Data Record  ( 5 minute detail record)
ClassID +100  Class Data Record  (60 minute summary record)
```

The **Date_Seconds** metric is identical to the user selectable **Date_Seconds** metric and is provided to ensure that records can be scanned easily for desired dates and times.

The **Number_of_vars** metric indicates how many groups of repeating fields are contained in the record. For global and configuration records, this metric is the number of disk drives configured on the system. If repeating netif metrics are chosen for these record types, the maximum number of netif devices (four) are exported regardless of the number configured.

For Multiple Instance application records, the **Number_of_vars** metric is the number of applications configured. For Multiple Instance disk device records, the **Number_of_vars** metric is the number of disk devices configured. For all header records, this metric is the maximum number of repeating groups allowed.

BINARY format files have special formats for the title and header records. These records contain the information needed to describe the contents of the file so that a program can properly interpret it. If **headings off** is specified, only data records will be in the file. If **headings on** is specified, the following records will precede all data records.

Table 4-5. Binary Header Records

Binary Header Records	
`Title Record`	This record (Record ID -1) is written whenever **headings on**, regardless of whether the user specified a report title. It contains information about the log file and its source.
`First Header Record`	The first header record (Record ID -2) contains a list of unique item identification numbers corresponding to the items contained in the log file. The position of the item ID numbers can be used to determine the position and size of each exported item in the file.
`Second Header Record`	The second header record (Record ID -3) contains a list of scale factors which correspond to the exported items. See the discussion of "Scale Factors" below for more details.
`Application Name Record`	This record (Record ID -4) will only be present in application data files that utilize the Multiple Layout format. It lists the names of the applications that correspond to the groups of application metrics in the rest of the file.
`Transaction Name Record`	This record (Record ID -5) will only be present in Transaction Tracker data files that utilize the Multiple Layout format. It lists the names of the transactions that correspond to the groups of Transaction Tracker metrics in the rest of the file.
`Disk Device Name Record`	This record (Record ID -7) will only be present in disk device data files that utilize the Multiple Layout format. It lists the names of disk devices that correspond to the groups of disk device metrics in the rest of the file.
`Logical Volume Name Record`	This record (Record ID -8) will only be present in logical volume data files that utilize the Multiple Layout format. It lists the names of logical volumes that correspond to the groups of logical volume metrics in the rest of the file.

Table 4-5. Binary Header Records (continued)

`Netif Name Record`	This record (Record ID -9) will only be present in netif (LAN) data files that utilize the Multiple Layout format. It lists the names of netif devices that correspond to the groups of netif device metrics in the rest of the file.

Binary Title Record

The Title Record for `BINARY` files contains information designed to assist programmatic interpretation of the exported file's contents. This record will be written to the exported file whenever `headings on` is specified.

The contents of the Binary Title Record are:

```
Record Length            4 byte Int    Length of Title Record
Record ID                4 byte Int    -1
Date_Seconds             4 byte Int    Date exported file was created
Number_of_vars           4 byte Int    Maximum number of repeating variables
Size of Fixed Area       4 byte Int    Bytes in nonvariable part of record
Size of Variable Entry   4 byte Int    Bytes in each variable entry
GMT Time Offset          4 byte Int    Seconds offset from Greenwich Mean Time
Daylight Savings Time    4 byte Int    =1 indicates Daylight Savings Time
System ID               40 Characters, System Identification
Collector Version       16 Characters, Name & version of the data collector
Log File Name           72 Characters, Name of the source log file
Report Title           100 Characters, User specified report title
```

The `Date_Seconds`, `GMT Time Offset`, and `Daylight Savings Time` metrics in the Binary Title Record apply to the system and time when the export file was created. If this is not the same system that logged the data, these fields cannot properly reflect the data in the file.

Binary Item Identification Record

The first header record (record ID -2) in the binary file contains the unique item numbers for each item exported. Each Item ID is a 4-byte integer number that can be cross referenced using the `rxitemid` file supplied with this product. The Item ID fields are aligned with the data fields they represent in the rest of the file. All

binary exported data items will occupy a multiple of 4 bytes in the exported file and each will start on a 4-byte boundary. If a data item requires more then 4 bytes of space, then its corresponding item ID field will be zero filled on the left.

For example, the process metric Program requires 16 bytes. Its data and item ID records would be:

```
Header 1 (Item Id Record) ...|   0|   0|   0|12012|
Process Data Record          |Prog|ram_|name| _aaa|
```

Binary Scale Factor Record

The second header record (record ID -3) in the binary file contains the scale factors for each of the exported items. Numeric items are exported to binary files as 32-bit (4-byte) integers in order to minimize problems with the way in which different computer architectures implement floating point. Before being truncated to fit into the integer format, most items are multiplied by a fixed scale factor. This allows floating point numbers to be expressed as a fraction, using the scale factor as a denominator.

Each scale factor is a 32-bit (4-byte) integer to match the majority of data items. Special values for the scale factors are used to indicate non-numeric and other special valued metrics.

Special Scale Factors

Non-numeric metrics, such as ASCII fields, have zero scale factors. A negative 1 scale factor should not occur, but if it does it indicates an internal error in the **extract** program and should be reported.

The DATE format is MPE CALENDAR format in the least significant 16 bits of the field (the 16 bits farthest right). The scale factor for date is 512. Scaling this as a 32-bit integer (dividing by 512) isolates the year as the

integer part of the date and the day of the year (divided by 512) as the fractional part.

TIME is a 4-byte binary field (hour, minute, second, tenths of seconds). The scale factor for `time` is 65536. Dividing it by 65536 forms a number where the integer part is the (hour * 256) + `minute`.

It is easier to handle a `Date_Seconds` value in a binary file.

Application Name Record

When application data is exported in the Multiple Layout format, a special Application Name Record is written to identify the application groups. For binary format files, this record has record ID -4. It consists of the binary record 16-byte header and a 20-byte application name for each application which was defined at the starting date of the exported data.

If applications are added or deleted during the time covered in the data extraction, the Application Name Record is repeated with the new application names.

Transaction Name Record

When Transaction Tracker transaction data is exported in the Multiple Layout format, a special Transaction Name Record is written to identify the transaction name. For binary format files, this record has a record ID -5. It consists of the binary record 16-byte header and a 20-byte transaction name for each transaction that was configured at the starting date of the exported data.

If no Transaction Tracker transactions are added during the time covered in the data extraction, the Transaction Name Record will be repeated with the new transaction name appended to the end of the original list. Transactions that are deleted after the start of the

data extraction will not be removed from the Multiple Layout data record.

Disk Device Name Record

When disk device data is exported in the Multiple Layout format, a special Disk Device Name Record is written to identify the disk device name. For binary format files, this record has a record ID -7. It consists of the binary record 16-byte header and a 20-byte disk device name for each disk device that was configured at the starting date of the exported data.

If disk devices are added during the time covered in the data extraction, the Disk Device Name Record will be repeated with the new disk device name appended to the end of the original list. Disk devices that are deleted after the start of the data extraction will not be removed from the Multiple Layout data record.

Logical Volume Name Record

When logical volume data is exported in the Multiple Layout format, a special Logical Volume Name Record is written to identify the logical volume name. For binary format files, this record has a record ID -8. It consists of the binary record 16-byte header and a 20-byte logical volume name for each logical volume that was configured at the starting date of the exported data.

If logical volumes are added during the time covered in the data extraction, the Logical Volume Name Record will be repeated with the new logical volume name appended to the end of the original list. Logical volumes that are deleted after the start of the data extraction will not be removed from the Multiple Layout data record.

Netif Name Record

When netif data is exported in the Multiple Layout format, a special Netif Name Record is written to identify the netif device name. For binary format files, this record has a record ID -11. It consists of the binary record 16-byte header and a 20-byte netif device name for each netif device that was configured at the starting date of the exported data.

If netif devices are added during the time covered in the data extraction, the Netif Name Record will be repeated with the new netif device name appended to the end of the original list. Netif devices that are deleted after the start of the data extraction will not be removed from the Multiple Layout data record.

5

Performance Alarms

This chapter describes what an alarm is, the syntax used to define an alarm, how an alarm works, and how alarms can be used to monitor performance.

You can use MeasureWare Agent to define alarms. These alarms notify you when **scopeux** or DSI metrics meet conditions that you have defined.

To define alarms, you specify conditions on each MeasureWare Agent system that when met or exceeded, trigger an alert or action. You define alarms in the MeasureWare Agent `alarmdef` file.

As data is logged by **scopeux**, DBMS collection servers, or other user-created collectors, it is compared to the alarm definitions to determine if a condition is met. When this occurs an alert or action can be triggered.

With the real-time **alarm generator** you can configure where you want alert notifications sent and whether you want local actions performed. Alert notifications can be sent to your central PerfView analysis system where you can see graphs of metrics that characterize the performance of your systems or applications. SNMP traps can be sent to HP OpenView or to any SNMP daemon. Local actions can be performed on your MeasureWare Agent system. Alert notifications can also be sent to HP OperationsCenter.

You can use the **utility** program's **analyze** command to analyze historical log file data against the alarm definitions and report the results.

Processing Alarms

As performance data is collected by MeasureWare Agent, it is compared to the alarm conditions defined in the `alarmdef` file to determine whether the conditions have been met. When a condition is met, an alarm is generated and the actions defined for alarms (ALERTs, PRINTs, and/or EXECs) are performed. You can set up how you want the alarm information communicated once an alarm is triggered. For example, you can:

- send information to the PerfView analysis software and create an alarm icon in the OpenView map associated with PerfView
- send SNMP traps to OpenView or any SNMP daemon
- send messages to OperationsCenter
- execute a UNIX command on the local system such as **mailx** to send yourself a message

How Alarms Are Processed

When you first start up MeasureWare Agent, the **perflbd** program starts repository servers for each data source configured in the `perflbd.rc` file and then starts the **alarm generator**. (Every data source mentioned in your alarm definition must have a corresponding entry in `perflbd.rc`.) See the *MeasureWare Agent: Installation and Configuration Guide* for information on **perflbd** and starting and stopping the **alarm generator**.

As data is collected in the log files, it is compared to the alarm definitions in the `alarmdef` file. When an alarm condition is met, the actions defined in the alarm definition are carried out.

Actions can include:

- local actions performed via UNIX commands
- messages sent to PerfView and OperationsCenter

Alarm Generator

The MeasureWare Agent **alarm generator** handles the communication of alarm notifications. The **alarm generator** consists of the **agdbserver** that manages the **agdb** database. The **agdb** database contains a list of PerfView analysis nodes, (if any), that you want to communicate alarm notifications to and various on/off flags you set to define if and where you want the alarm notifications sent. It also contains a list of SNMP nodes. The **agsysdb** program manages the **agdb** database.

When you start up MeasureWare Agent, the **perflbd** program starts the repository servers, the **agdbserver**, and the **alarm generator**. The **alarm generator** then reads the database to determine where and whether to send alarm notifications. It also checks to see if an OperationsCenter agent is on the system.

Use the following command line option to list where alert notifications will be sent:

```
agsysdb -l
```

Communicating Alarm Notifications to PerfView

Use PerfView to set up (or delete) which MeasureWare Agents you want to receive alarms from (see the PerfView online help for an explanation of how to do this). The PerfView analysis system then gets entered into the **agdb** database on the MeasureWare Agent system. This tells the **alarm generator** to send alert notifications to that PerfView analysis system. If the ALERT is part of an ALARM statement, the message sent to PerfView includes the text of the ALERT, alarm TYPE, and SEVERITY. If OpenView is also set up on that PerfView analysis system, an alarm icon is created in the OpenView map. (See the PerfView online help for an explanation of how this works).

Sending SNMP Traps to OpenView

To send SNMP traps to OpenView or any SNMP daemon, you must add your system name to the agdb database in MeasureWare Agent as follows:

```
agsysdb -add systemname
```

Every ALERT generated will cause an SNMP trap to be sent to the system you defined. The trap text will contain the same message as the ALERT.

To stop sending SNMP traps to a system, you must delete the system name from the agdb database as follows:

```
agsysdb -delete systemname
```

Sending Messages to OperationsCenter

You can have alert notifications sent to OperationsCenter if there is an OperationsCenter agent on the same system as MeasureWare Agent. The OperationsCenter agent communicates with the central OperationsCenter system.

By default, if the OperationsCenter agent is running on the MeasureWare Agent system, the **alarm generator** does *not* execute local actions that are defined in any alarms in the EXEC statement. Instead, it sends a message to OperationsCenter's event browser. If the OperationsCenter agent is *not* running on the MeasureWare Agent system, the **alarm generator** does not try to send alert notifications to OperationsCenter and local actions are executed.

You can change the default to stop sending information to OperationsCenter, even though an OperationsCenter agent is running on the MeasureWare Agent system, by typing:

```
agsysdb -opc OFF
```

Note

If you install OperationsCenter after MeasureWare Agent and the **alarm generator** are already started, you must restart the **alarm generator** using the `mwa restart server` script.

Executing Local Actions

Without an OperationsCenter agent running on the MeasureWare Agent system, local actions in EXEC statements will be executed.

You can change the default to turn off local actions as follows:

```
agsysdb -actions off
```

If you want local actions to *always* execute even if the OperationsCenter agent is running, type:

```
agsysdb  -actions always
```

The following table lists the settings for sending information to OperationsCenter and for executing local actions:

Flags	OperationsCenter Agent Running	OperationsCenter Agent Not Running
OperationsCenter Flag		
`off`	No alert notifications sent to OperationsCenter.	No alert notifications sent to OperationsCenter.
`on`	Alert notifications sent to OperationsCenter.	No alert notifications sent to OperationsCenter.
Local Actions Flag		
`off`	No local actions executed.	No local actions executed.
`always`	Local actions executed even if OperationsCenter agent is running.	Local actions executed.
`on`	Local actions sent to OperationsCenter.	Local actions executed.

Errors in Processing Alarms

The last error that occurred when sending an alarm is logged in the `agdb` database. To read this database, type:

```
agsysdb -l
```

The following information is displayed:

```
MeasureWare alarming status:


  OpC message : on   Last error: <error number>
  Exec actions: on   (See status.alarmgen file for errors)


  Analysis system:  <hostname>, Key=<ip address>
   PerfView : no      Last error: <error number>
   SNMP     : yes     Last error: <error number>
```

Using Utility to Analyze Historical Data

You can use the **analyze** command in **utility** to analyze alarm conditions in a log file. This is different from the processing of real-time alarms explained above because you are comparing historical data in the log file to the alarm definitions in the **alarmdef** file to determine what alarm conditions would have been triggered. For more information on using the **analyze** command, see the "Analyze Command" section in Chapter 3 of this manual.

Examples of Alarm Information in Historical Data

The following examples show what is reported when you analyze alarm conditions in historical data.

For the first example, START, END, and REPEAT statements have been defined in the alarm definition. An *alarm-start* event is listed every time an alarm has met all of its conditions for the specified duration. When these conditions are no longer satisfied, an *alarm-end* event is listed. If an alarm condition is satisfied for a period long enough to generate another alarm without having first ended, a *repeat* event is listed.

Each event listed shows the date and time, alarm number, and the alarm event. The actions *are not performed*, but they are listed with any requested parameter substitutions in place. EXECs are listed.

```
09/11/95 11:15  ALARM [1] START
  CRITICAL: CPU test 99.97%

09/11/95 11:20  ALARM [1] REPEAT
  WARNING: CPU test 99.996%

09/11/95 11:25  ALARM [1] END
  RESET: CPU test  22.86%
  EXEC: end.script
```

If you are using a color workstation, the following output is highlighted:

CRITICAL statements are red

WARNING statements are yellow

RESET statements are green

The next example shows an alarm summary that is displayed after alarm events are listed. The first column lists the alarm number, the second column lists the number of times the alarm condition occurred, and the third column lists the total duration of the alarm condition.

```
Performance Alarm Summary:

  Alarm   Count   Minutes
      1     574      2865
      2       0         0

Analysis coverage using "alrmdef2":

Start: 09/04/95 08:00    Stop: 09/06/95 23:59

Total time analyzed:  Days: 2  Hours: 15  Minutes: 59
```

Components of Alarm Definitions

An alarm occurs when one or more of the conditions you define continues over a specified duration. The alarm definition can include an action to be performed at the start or end of the alarm.

A condition is a comparison between two or more items. The compared items can be metric names, constants, or variables. For example,

```
ALARM gbl_cpu_total_util > 95 FOR 5 MINUTES
```

An action can be specified to be performed when the alarm starts, ends, or repeats. The action can be one of the following:

- an ALERT, which sends a message to PerfView or OperationsCenter, or an SNMP trap to OpenView or an SNMP daemon
- an EXEC, which performs a UNIX command, or
- a PRINT, which sends a message to `stdout` when processed using the **utility** program.

For example,

```
ALARM gbl_swap_space_util > 95 FOR 5 MINUTES
    START
        RED ALERT "Global swap space is nearly full"
    END
        RESET ALERT "End of global swap space full condition"
```

You can create more complex actions using Boolean logic, loops through multiple-instance data such as application or process, and variables. See the alarm syntax reference for details on these statements.

You can also use the INCLUDE statement to identify additional alarm definition files you want used. For example, you may want to break up your alarm definitions into smaller files.

Alarm Syntax Reference

This section shows the statements that are available in the alarm syntax. You may want to look at the **alarmdef** file for examples of how the syntax is used to create useful alarm definitions.

Alarm Syntax

```
ALARM  condition [[AND,OR]condition]
  [FOR duration[SECONDS, MINUTES, INTERVALS]]

  [TYPE="string"]
  [SEVERITY=integer]
  [START action]
  [REPEAT [EVERY duration [SECONDS, MINUTES, INTERVALS]] action]
  [END action]

[RED, CRITICAL, YELLOW, WARNING, RESET] ALERT message


EXEC "UNIX command"


PRINT message

IF condition
   THEN action
   [ELSE action]

{APPLICATION, PROCESS, DISK, TRANSACTION, LVOLUME, NETIF} LOOP action

INCLUDE "filename"


USE "data source name"


 [VAR] name =  value


 ALIAS name = "replaced-name"


SYMPTOM  variable [ TYPE = {CPU, DISK, MEMORY, NETWORK}]
```

```
RULE condition  PROB probability
[RULE condition PROB probability]
  .
  .
```

Alarm Syntax Conventions

Conventions

- Braces ({}) indicate that one of the choices is required.
- Brackets ([]) indicate an optional item.
- Items separated by commas within brackets or braces are options. Choose only one.
- Italics indicate a variable name that you replace.
- All syntax keywords are in uppercase.

Common Elements of the Alarm Syntax

The following elements are used in several statements in the alarm syntax and are documented below for your reference.

Comments

You can precede comments either by double forward slashes (//) or the pound sign (#). In both cases, the comment ends at the end of the line. For example,

```
# any text or characters
```

or

```
// any text or characters
```

Compound Statement

Compound statements allow a list of statements to be executed as a single statement. A compound statement is a list of statements inside braces ({}). Use the compound statement with the "IF Statement", the "LOOP Statement", and the START, REPEAT, and END clauses of the "ALARM Statement". Compound statements cannot include ALARM and SYMPTOM statements.

```
{
 statement
 statement
}
```

In the example below, `highest_cpu = 0` defines a variable called `highest_cpu`. The `highest_cpu` value is saved and notifies you only when that `highest_cpu` value is exceeded by a higher `highest_cpu` value.

```
highest_cpu = 0
IF  gbl_cpu_total_util > highest_cpu  THEN
  // Begin compound statement
  {
  highest_cpu = gbl_cpu_total_util
  ALERT "Our new high CPU value is ", highest_cpu, "%"
  }
  // End compound statement
```

Conditions

A condition is defined as a comparison between two items.

item1 {>, <, >=, <=, ==, !=}*item2*
  [AND, OR[*item3* {>, <, >=, <=, ==, !=}*item4*]]

where ("==" means "equal", and "!=" means "not equal".

Conditions are used in the ALARM, IF, and SYMPTOM statements. An item can be a metric name, a numeric constant, an alphanumeric string enclosed in quotes, an alias, or a variable. When comparing alphanumeric items, only == or != can be used as operators.

Constants

Constants can be either numeric or alphanumeric. An alphanumeric constant must be enclosed in double quotes. For example:

```
345
345.2
"Time is"
```

Constants are useful in expressions and conditions. For example, you may want to compare a metric against a constant numeric value inside a condition to generate an alarm if it is too high, such as

```
gbl_cpu_total_util > 95
```

Expressions

Arithmetic expressions perform one or more arithmetic operations on two or more operands. You can use an expression anywhere you would use a numeric value. Legal arithmetic operators are:

```
+, -, *, /
```

Parentheses can be used to control which parts of an expression are evaluated first.

For example:

```
Iteration + 1
gbl_cpu_total_util - gbl_cpu_user_mode_util
( 100 - gbl_cpu_total_util ) / 100.0
```

Interval

An interval is the period of time since the last measurement. An interval will either be the shortest duration in any alarm condition defined, five minutes if process data is not collected, or one minute if it is.

Metric Names

When you use a metric name in an alarm definition, the current value of the metric is substituted. In MeasureWare Agent, see the list of metric names in **extract** online help. In PerfView, choose On Metrics from the PerfView help menu to display a list of metrics by platform. Metric names must be typed exactly as they appear in the help systems, except for case sensitivity.

It is recommended that you use fully-qualified metric names if the metrics are from a data source other than the SCOPE data source (such as DSI metrics).

For application metrics, you must associate a metric with a specific application, except when using the "LOOP Statement". To do this, specify the application name followed by a colon, and then the metric name. For example, `other_apps:app_cpu_total_util` specifies the total CPU utilization for the application `other_apps`.

The format for specifying a fully qualified metric is:

```
data_source:instance(class):metric_name
```

A global metric in the SCOPE data source requires no qualification. For example,

```
metric_1
```

An application metric, which is available for each application defined in the SCOPE data source, requires the application name:

```
application_1:metric_1
```

If you use an application name that has an embedded space, you must replace the space with an underscore (_). For example, `application 1` must be changed to `application_1`. See the "ALIAS Statement" section for

more information on using names that contain special characters, or names where case is significant.

If you had a disk named "other" and an application named "other", you would need to specify the class as well as the instance:

```
other(disk):metric_1
```

A global metric in an extracted log file (where `scope_extract` is the data source name) would be specified this way:

```
scope_extract:application_1:metric_1
```

A DSI metric would be specified this way:

```
dsi_data_source:dsi_class:metric_name
```

Disk data class names cannot be used directly in alarm definitions because they contain asterisks. Any names containing special characters must be aliased before they are specified.

In the following example, the disk class name `c410d5s*` is aliased to give it a unique name. The alias is then used in an ALARM statement:

```
ALIAS c410d5s = "c410d5s*"
ALARM c410d5s:bydisk_space_util > 65
```

Messages

A message is the information sent by a PRINT or ALERT statement. It can consist of any combination of quoted alphanumeric strings, numeric constants, expressions, and variables. The elements in the message are separated by commas. For example,

```
RED ALERT "cpu utilization=", gbl_cpu_total_util
```

Numeric constants, metrics, and expressions can be formatted for width and number of decimals.

For example:

metric names [|*width*[|*decimals*]]

```
gbl_cpu_total_util|6|2   formats as '100.00'
(100.32 + 20)|6          formats as '   120'
```

ALARM Statement

The ALARM statement defines a condition or set of conditions and a duration for the conditions to be true. Within the ALARM statement, you can define actions to be performed when the alarm condition starts, repeats, and ends. Conditions or events that you might want to define as alarms include:

- when global swap space has been nearly full for 5 minutes
- when the memory paging rate has been too high for 1 interval
- when your CPU has been running at 75 percent utilization for the last ten minutes

Syntax

```
ALARM  condition [[AND,OR]condition]
  [FOR duration[SECONDS, MINUTES, INTERVALS]]

  [TYPE="string"]
  [SEVERITY=integer]
  [START action]
  [REPEAT [EVERY duration [SECONDS, MINUTES, INTERVALS]] action]
  [END action]
```

- The ALARM statement must be a top-level statement. It cannot be nested within any other statement. However, you can include several ALARM conditions in a single ALARM statement. If the conditions are linked by AND, all conditions must be true to trigger the alarm. If they are linked by OR, any one condition will trigger the alarm.
  - TYPE is a quoted string of up to 8 characters. If you are sending alarms to PerfView, you can use TYPE to categorize alarms and to specify the name of a graph template to use.

- SEVERITY is an integer from 0 to 32767. If you are sending alarms to PerfView, you can use this to categorize alarms.

- START, REPEAT, and END are *keywords* used to specify what action to take when alarm conditions are met, met again, or stop. You should always have at least one of START, REPEAT, or END in an ALARM statement. Each of these keywords is followed by an *action*.

- *action* - The action most often used with an ALARM START, REPEAT or END is the ALERT statement. However, you can also use the EXEC statement to mail a message or run a script, or a PRINT statement if you are analyzing historical log files with **utility**. Any syntax statement is legal except another ALARM.

  START, REPEAT, and END actions can be compound statements. For example, you can use compound statements to provide both an ALERT and an EXEC.

- Conditions - A condition is defined as a comparison between two items.

  ```
  item1 {>, <, >=, <=, ==, !=}item2
     [AND, OR[item3 {>, <, >=, <=, ==, !=}item4]]
  ```

  where "==" means "equal", and "!=" means "not equal".

  An item can be a metric name, a numeric constant, an alphanumeric string enclosed in quotes, an alias, or a variable. When comparing alphanumeric items, only == or != can be used as operators.

  You can use compound conditions by specifying the "OR" and "AND" operator between sub-conditions. For example,

  ```
  ALARM gbl_cpu_total_util > 90 AND
     gbl_pri_queue > 1 for 5 minutes
  ```

Note

WARNING! Do not use metrics that are logged at different intervals in the same statement. For example, you should not loop on a process (logged at 1-minute intervales) based on the value of a global metric (logged at 5-minute intervals) in a statement like this:

```
IF global_metric THEN
  PROCESS LOOP...
```

The different intervals cannot be synchronized as you might expect, so results will not be valid.

You can use metrics logged at different intervals with the **utility** program's **analyze** command against static (historical) log files because the data can be synchronized.

- `[FOR` *duration* `SECONDS, MINUTES, INTERVALS]` specifies the time period the condition must remain true to trigger an alarm. The **alarm generator** evaluates the SYMPTOMS and ALARMS at each interval. For data collected by **scopeux**, an "interval" is the shortest of:
  - 5 minutes, if process data is not being collected on the system.
  - 1 minute, if process data is being collected.
  - the shortest duration in any alarm definition.

  DSI data can be collected at intervals of 5 seconds or longer.

  Use caution when specifying durations of less than one minute, particularly when there are multiple data sources on the system. Performance can be seriously degraded if each data source must be polled for data at very small intervals. The duration must be a multiple of the longest collection interval of the metrics mentioned in the alarm condition.

- [REPEAT EVERY *duration* SECONDS, MINUTES, INTERVALS] specifies the time period before the alarm is repeated.

How It Is Used

The alarm cycle begins on the first interval that all of the ANDed, or one of ORed alarm conditions have been true for at least the specified duration. At that time, the **alarm generator** executes the START *action*, and on each subsequent interval checks the REPEAT condition. If enough time has transpired, the *action* for the REPEAT clause is executed. (This continues until one or more of the alarm conditions becomes false.) This completes the alarm cycle and the END *statement* is executed if there is one.

In order for PerfView to be notified of the alarm, you should use the ALERT statement within the START and END statements. If you do not specify an END ALERT, the **alarm generator** will automatically send one to PerfView and OperationsCenter and send an SNMP trap to any SNMP daemon.

Example of a Typical ALARM Statement

The following ALARM example sends a red alert when the swap utilization is high for 5 minutes. It is similar to an alarm in the default `alarmdef` file. Do not add this example to your `alarmdef` file without removing the default, or your subsequent alert messages may be confusing.

```
ALARM  gbl_swap_space_util > 90 FOR 5 MINUTES
  START
    RED ALERT "swap utilization is very high "
  REPEAT EVERY 15 MINUTES
    RED ALERT "swap utilization is still very high "
  END
    RESET ALERT "End of swap utilization condition"
```

This ALARM example tests the metric `gbl_swap_space_util` to see if it is greater than 90. Depending on how you configured the **alarm generator**, the ALERT can be sent to the Alarms window in PerfView, to OpenView via an SNMP trap, or as a message to OperationsCenter. See the "Alarm Generator" section for details. If you have PerfView configured correctly, the RED ALERT statement places the "swap utilization still very high" message in the PerfView Alarms Window.

The REPEAT statement checks for the `gbl_swap_space_util` condition every 15 minutes. As long as the condition is greater than 90, the REPEAT statement sends the `"swap utilization is still very high"` message.

When the `gbl_swap_space_util` condition goes below 90, the RESET ALERT statement with the `"End of swap utilization condition"` message is sent wherever it is configured to go.

Example of Using Compound Actions

The following example defines a compound action within the ALARM statement. This example shows you how to cause a message to be mailed when an event occurs.

```
ALARM gbl_cpu_total_util > 90 FOR 5 MINUTES
  START  {
    RED ALERT "Your CPU is bottlenecked."
    EXEC "echo 'cpu is too high' | mail root"
      }
  END

    RESET ALERT "CPU crisis is over."
```

The ALERT can trigger an SNMP trap to be sent to OpenView or to an SNMP daemon, or a message to be sent to OperationsCenter. The EXEC can trigger a mail message to be sent as a local action on your MeasureWare Agent system, depending on how you

have configured your **alarm generator**. See the "Alarm Generator" section for details. If you have PerfView set up to receive alarms from this system, the RED ALERT statement places the "Your CPU is bottlenecked" message in the PerfView Alarms window and causes a message to be mailed.

By default, if the OperationsCenter agent is running, the local action will not execute. Instead, it will be sent as a message to OperationsCenter.

Example of Using Multiple Conditions

You can have more than one test condition in the ALARM statement. In this case, each statement must be true for the ALERT to be sent.

```
ALARM gbl_cpu_total_util > 90
  AND gbl_cpu_sys_mode_util > 50 FOR 5 MINUTES

START
  RED ALERT "The CPU is busy and System Mode CPU
   utilization is high."
END
  RESET ALERT "The CPU alert is now over."
```

The ALARM example above tests the metric `gbl_cpu_total_util` and the metric `gbl_cpu_sys_mode_util`. If both conditions are true, the RED ALERT statement sends a red alert. When either test condition becomes false, the RESET is sent.

```
ALARM gbl_cpu_total_util > 90
  OR
      gbl_cpu_sys_mode_util > 50
      FOR 10 MINUTES
START
  RED ALERT "Either total CPU utilization or system mode CPU
    utilization is high"
```

The ALARM example above tests the metric `gbl_cpu_total_util` and the metric `glb_cpu_sys_mode_util`. If either condition is true, the RED ALERT statement sends a red alert.

ALERT Statement

The ALERT statement allows a message to be sent to PerfView, OpenView, any SNMP daemon, or OperationsCenter. It also allows the creation and deletion of the alarm icons in the OpenView map associated with PerfView and controls the color of the alarm icons, depending on the severity of the alarm.

The ALERT statement is most often used as an action within an ALARM. It could also be used within an IF statement to send a message as soon as a condition is detected instead of after the duration has passed. If an ALERT is used outside of an ALARM or IF statement, the message will be sent at every interval.

Syntax

```
[RED, CRITICAL, YELLOW, WARNING, RESET] ALERT message
```

- RED is synonymous with CRITICAL and YELLOW is synonymous with WARNING. These keywords turn the alarm icon in the OpenView map associated with PerfView red or yellow. They also send the message with other information to the PerfView Alarms window. YELLOW is the default.
- RESET records the *message* in the PerfView Alarm window and deletes the alarm icon in the OpenView map associated with PerfView. A RESET ALERT without a message is sent automatically when an ALARM condition ENDs if you do not define one in the alarm definition.
- *message* - A combination of strings and numeric values used to create a message. Numeric values can be formatted with the parameters [|*width*[|*decimals*]].

How It Is Used

The ALERT can also trigger an SNMP trap to be sent to OpenView or any SNMP daemon, or a message to be sent to OperationsCenter, depending on how

you have configured your **alarm generator**. See the "Alarm Generator" section for details. If you have PerfView configured to receive alarms from this system, the ALERT sends a message to the PerfView Alarms window.

If an ALERT statement is used outside of an ALARM statement, the alert message will show up in the PerfView alarms window but no icon will be created in the OpenView map.

For alert messages sent to OperationsCenter, the WARNINGS will be displayed in blue in the message browser.

ALERT Example

An example ALERT statement is:

```
RED ALERT "CPU utilization = ", gbl_cpu_total_util
```

If you have PerfView and OpenView, this statement creates a red alarm icon in the OpenView map associated with PerfView and sends a message to the PerfView Alarms window that reads:

```
CPU utilization = 85.6
```

EXEC Statement

The EXEC statement allows you to specify a UNIX command to be performed on the local system. You could use the EXEC command, for example, to send a mail message to the MIS staff each time a certain condition is met.

EXEC should be used within an ALARM or IF statement so the UNIX command is performed only when specified conditions are met. If an EXEC statement is used outside of an ALARM or IF statement, the action will be performed at every interval.

EXEC statements will execute only if the related ALARM or IF statement refers to a metric.

Syntax

```
EXEC "UNIX command"
```

- *UNIX command* - a UNIX command to be performed on the local system.

How It Is Used

The EXEC can trigger a local action on your local system, depending on how you have configured your **alarm generator**. For example, you can turn local action on or off. If you have configured your **alarm generator** to send information to OperationsCenter, local actions will not usually be performed. See the "Alarm Generator" section for details.

Note

Be careful when using the EXEC statement with UNIX commands or scripts that have high overhead if they will be performed at every interval.

The **alarm generator** executes the command and waits until it completes before continuing. Be careful not to specify commands that take a long time to complete.

EXEC Example

In the following example, EXEC performs the UNIX **mailx** command only when the `gbl_disk_util_peak` metric exceeds 20.

```
IF gbl_disk_util_peak > 20 THEN
EXEC "echo 'MeasureWare Agent detects high disk utilization' | mailx root"
```

PRINT Statement

The PRINT Statement allows you to print a message to stdout. The PRINT statement is only useful when using MeasureWare Agent's **utility** program. The **alarm generator** ignores the PRINT statement.

Syntax

```
PRINT message
```

- *message* - A combination of strings and numeric values that create a message. Numeric values can be formatted with the parameters [|*width*[|*decimals*]]. Alphanumeric components of a message must be enclosed in quotes.

PRINT Example

```
PRINT "The total time the CPU was not idle is", gbl_cpu_total_time |6|2, "seconds"
```

When executed, this statement prints a message such as the following:

```
The total time the CPU was not idle is .95.00 seconds
```

IF Statement

Use the IF statement to define a condition using IF-THEN logic. It can be used within the ALARM statement. Or, it can be used instead of an ALARM statement if you want to send an ALERT, EXEC a command, or PRINT a message as soon as a condition is detected. (The ALARM statement requires that a condition remain true for at least one interval before an action is performed.)

The IF statement can be used by itself or anyplace in the `alarmdef` file where IF-THEN logic is needed.

Syntax

```
IF  condition THEN action [ELSE action]
```

- IF *condition* - A condition is defined as a comparison between two items.

  ```
  item1 {>, <, >=, <=, ==, !=}item2
     [AND, OR[item3 {>, <, >=, <=, ==, !=}item4]]
  ```

  where "==" means "equal", and "!=" means "not equal".

  An item can be a metric name, a numeric constant, an alphanumeric string enclosed in quotes, an alias, or a variable. When comparing alphanumeric items, only == or != can be used as operators.

Note

WARNING! Do not use metrics that are logged at different intervals in the same statement. For instance, you should not loop on a process (logged at 1-minute intervals) based on the value of a global metric (logged at 5-minute intervals) in a statement like this:

```
IF global_metric THEN
  PROCESS LOOP ...
```

The different intervals cannot be synchronized as you might expect, so results will not be valid.

You can use metrics logged at different intervals with the **utility** program's **analyze** command against static (historical) log files, because the data can be synchronized.

- *action* - Any action, or set a variable. (ALARM is not valid in this case.)

How It Is Used

The IF statement tests the *condition*. If true, the *action* after the THEN is executed. If the *condition* is false, the action depends on the optional ELSE clause. If an ELSE clause has been specified, the *action* following it is executed, otherwise the IF statement does nothing.

IF Example

```
IF gbl_cpu_total_util > 90 THEN
YELLOW ALERT "The CPU is running hot at: ", gbl_cpu_total_util
ELSE IF gbl_cpu_total_util < 20 THEN
RESET ALERT "The CPU is idling at: ", gbl_cpu_total_util
```

In this example, the IF statement monitors CPU utilization for unusually high or low values. If you have PerfView configured correctly, the message "`The CPU is running hot at:`" is displayed in the PerfView Alarms window along with the percentage of CPU used.

If the `gbl_cpu_total_util > 90` condition is false, ELSE IF goes to the next line and checks the condition `gbl_cpu_total_util < 20`. If that condition is true, then `"The CPU is idling at:"` is displayed in the PerfView Alarms window along with the percentage of CPU used.

The ALERT can also trigger an SNMP trap to be sent to OpenView or any SNMP daemon, or a message to be sent to OperationsCenter, depending on how you have configured your **alarm generator**. See the "Alarm Generator" section for details.

LOOP Statement

The LOOP statement goes through multiple-instance data classes and for each instance it performs the *action* defined.

Syntax

```
{APPLICATION, PROCESS, DISK, TRANSACTION, LVOLUME, NETIF} LOOP action
```

- APPLICATION, PROCESS, DISK, TRANSACTION, LVOLUME, NETIF - MeasureWare Agent data classes that contain multi-instance data.
- *action* - PRINT, EXEC, ALERT, set variables.

How It Is Used

As LOOP statements iterate through each instance of the class, metric values change. For instance, the following LOOP statement prints the name of each application to `stdout` if you are using the **utility** program's **analyze** command.

```
APPLICATION LOOP
   PRINT application_name
```

A LOOP can be nested within another LOOP statement up to a maximum of five levels.

In order for the LOOP to execute, the LOOP statement must refer to one or more metrics of the same data class as the class defined in the LOOP statement.

APPLICATION LOOP Example

You can use the LOOP statement to cycle through all active applications.

The following example uses an LOOP to find the application with the highest CPU at each interval. The LOOP statement is used to set the `big_app` and `highest_cpu` variable values. Then, `highest_cpu` is tested to determine if an alert should be set.

```
big_app = ""
highest_cpu = 0
APPLICATION LOOP
  IF (app_cpu_total_util > highest_cpu) THEN
   {
   highest_cpu = app_cpu_total_util
   big_app = app_name
   }
IF (highest_cpu > 20) THEN

   RED ALERT "The application ", big_app,  "is the highest CPU user at",
    highest_cpu, "%"
```

INCLUDE Statement

Use the INCLUDE statement to include another alarm definitions file along with the `alarmdef` file.

Syntax

```
INCLUDE "filename"
```

where *filename* is the name of another alarm definitions file. The file name must always be fully qualified.

How It Is Used

The INCLUDE statement could be used to separate logically distinct sets of alarm definitions into separate files.

INCLUDE Example

For example, if you have some alarm definitions in a separate file for your Sybase database metrics and it is named

```
SYBASE.alarmdef1
```

You can include it by adding the following line to the alarm definitions in your main `alarmdef` file:

```
INCLUDE "/path/SYBASE.alarmdef1"
```

USE Statement

Use the USE statement to specify a data source other than the SCOPE data source to be referenced in your alarm definition.

Syntax

USE "*data source name*"

where *data source name* is the name to use instead of SCOPE.

How It Is Used

By default, alarms are processed using the SCOPE data source. To specify a data source other than SCOPE to use in your alarm definition, add a USE statement.

For example, if you want to validate your alarm definitions against an extracted log file without having to fully qualify the metrics with the data source name, add the USE statement to your alarm definition to define the new data source.

Another example is if you renamed your active SCOPE data source name to `scope_extract` in the `perflbd_rc` file, the metrics you specified in your ALARM statement would not be found unless you add the USE statement to specify the `scope_extract` data source. The following statements would cause the **alarm generator** to get the metric value from the `scope_extract` data source:

```
USE "scope_extract"
IF gbl_total_cpu_util > 90
```

A USE statement should always be used on any system where the default SCOPE data source is not running or where it has been renamed in the `perflbd.rc` file, to ensure that metrics are selected from the intended log file.

VAR Statement

The VAR statement allows you to define a variable and assign a value to it.

Syntax

[VAR] *name = value*

- *name* - Variables names must begin with a letter and can include letters, digits, and the underscore character. Variables names are not case-sensitive.
- *value* - If the value is an alphanumeric string, it must be enclosed in quotes.

How It Is Used

VAR assigns a value to the user variable. If the variable did not previously exist it is created.

Once defined, variables can be used anywhere in the `alarmdef` file.

Examples

Define a variable by assigning something to it. For example:

```
highest_CPU_value = 0
```

This example defines the numeric variable `highest_CPU_value` by assigning it a value of zero.

```
my_name = ""
```

This example defines the alphanumeric variable `my_name` by assigning it an empty string value.

ALIAS Statement

The ALIAS statement allows you to substitute an alias if any part of a metric name (class, instance, or metric) has a case-sensitive name or a name that includes special characters (such as disk class names). These are the *only* circumstances where the ALIAS statement should be used.

Syntax

```
ALIAS name = "replaced-name"
```

- *name* - The name must begin with a letter and can include letters, digits, and the underscore character.
- *replaced-name* - The name that must be replaced by the ALIAS statement to make it uniquely recognizable to the **alarm generator**.

How It Is Used

Because of the way the `alarmdef` file is processed, if any part of a metric name (class, instance, or metric name) can only be identified uniquely by recognizing uppercase and lowercase, you will need to create an alias. You will also need to create an alias for any name that includes special characters. For example, if you have applications called "BIG" and "big", you'll need to alias "big" to ensure that they are viewed as different applications.

You would need to define the alias somewhere in the `alarmdef` file before the first instance of the name you want substituted.

ALIAS Example

Because you cannot use special characters or upper and lower case in the syntax, using the application name "AppA" and "appa" could cause errors because the processing would be unable to distinguish between the two. You would alias "AppA" to give it a uniquely recognizable name. For example,

```
ALIAS appa_uc = "AppA"
  ALERT "CPU alert for AppA.util is", appa_uc:app_cpu_total_util
```

SYMPTOM Statement

A symptom provides a way to set a single variable value based on a set of conditions. Whenever any of the conditions is true, its probability value is added to the value of the sympton variable.

Syntax

```
SYMPTOM variable
  RULE condition PROB probability
  [RULE condition PROB probability]
      .
      .
      .
```

- The keywords SYMPTOM and RULE are used exclusively in the SYMPTOM statement and cannot be used in other syntax statements. The SYMPTOM statement must be a top-level statement and cannot be nested within any other statement. No other statements can follow SYMPTOM until all its corresponding RULE statements are finished.
- *variable* is a variable name that will be the name of this symptom. Variable names defined in the SYMPTOM statement can be used in other syntax statements, but the variable value should not be changed in those statements.
- RULE is an option of the SYMPTOM statement and cannot be used independently. You can use as many RULE options within the SYMPTOM statement as you need. The SYMPTOM variable is evaluated according to the rules at each interval.
- *condition* is defined as a comparison between two items.

item1 {>, <, >=, <=, ==, !=}*item2* [AND, OR[*item3* {>, <, >=, <=, ==, !=}*item4*]]

where "==" means "equal", and "!=" means "not equal".

An item can be a metric name, a numeric constant, an alphanumeric string enclosed in quotes, an alias, or a variable. When comparing alphanumeric items, only == or != can be used as operators.

- *probability* is a numeric constant. The probabilities for each true SYMPTOM RULE are added together to create a SYMPTOM value.

How It Is Used

The sum of all probabilities where the condition between measurement and value is true is the probability that the symptom is occurring.

SYMPTOM Example

```
SYMPTOM CPU_Bottleneck
RULE gbl_cpu_total_util > 50  PROB 25
RULE gbl_cpu_total_util > 85  PROB 25
RULE gbl_cpu_total_util > 90  PROB 25
RULE gbl_run_queue      >  3  PROB 50
IF cpu_bottleneck > 50 THEN
YELLOW ALERT "The CPU is stuck at: ", cpu_bottleneck
```

Alarm Definition Examples

The following section contains examples of alarm definitions.

Example Showing a CPU Problem

```
ALARM gbl_cpu_total_util > 90
  gbl_run_queue > 3 FOR 5 MINUTES

START
  YELLOW ALERT "CPU AT ", gbl_cpu_total_util, "% at "
REPEAT EVERY 20 MINUTES
  {RED ALERT "CPU AT ", gbl_cpu_total_util
  EXEC "/usr/bin/pager -n 555-3456"}
END ALERT "CPU at ", gbl_cpu_total_util, "% - RELAX"
```

If you have PerfView configured correctly, this example turns the alarm icon yellow in the OpenView map (whenever CPU utilization exceeds 90 percent for 5 minutes and the CPU run queue exceeds 3 for 5 minutes), and sends a message to the PerfView Alarms window.

The ALERT could also trigger an SNMP trap to be sent to OpenView or a message to be sent to OperationsCenter, depending on how you configured the **alarm generator**. See the "Alarm Generator" section for details.

If both conditions continue to hold true after 20 minutes, a red alert is generated, the alarm icon turns red in the OpenView map, and another message is sent to the PerfView Alarms window. A program is then run to page the system administrator.

When either one of the alarm conditions fails to be true, the alarm icon is deleted and a message is sent to the PerfView Alarms window showing the global CPU utilization, the time the alert ended, and a note to RELAX.

Example of Swap Utilization

```
ALARM gbl_swap_space_util > 95
  START
    RED ALERT "GLOBAL SWAP space is nearly full "
  END
    RESET ALERT "End of GLOBAL SWAP full condition"
```

If you have PerfView configured correctly, the alarm icon in the OpenView map turns red and a message is written to the PerfView Alarms window.

If you don't have PerfView, the ALERT can trigger an SNMP trap to be sent to OpenView or a message to be sent to OperationsCenter, depending on how you have configured your **alarm generator.** See the "Alarm Generator" section for details.

Customizing Alarm Definitions

You specify the conditions that generate alarms in the MeasureWare Agent `alarmdef` file that resides in the *configuration* directory. When MeasureWare Agent is first installed, the `alarmdef` file contains a set of default alarm definitions. You can use these default alarm definitions or customize them to suit your needs.

You can customize the `alarmdef` file as follows:

1. Edit the `alarmdef` file as necessary. You can look at examples of the alarm definition syntax elsewhere in this chapter.
2. Save the file.
3. Validate the alarm definitions by using the MeasureWare Agent **utility** program:

   a. Type `utility`.

   b. At the prompt type:

      ```
      checkdef
      ```

   This checks the alarm syntax and displays errors or warnings if there are any problems in the file.
4. In order for the **alarm generator** to read the new alarm definitions, type:

   ```
   mwa restart alarm
   ```

   to have the customized alarm definitions take affect.

You can use a unique set of alarm definitions for each MeasureWare Agent system, or you can choose to standardize monitoring of a group of systems by using the same set of alarm definitions across the group.

The best way to learn about performance alarms is to experiment with adding new alarms or changing the default alarm definitions.

Syntax Errors

To check for errors in your alarm definition syntax, use the **utility** program's **checkdef** command. See the description of **checkdef** in Chapter 3 of this manual.

A

Utility Scan Report Details

The **utility** program's **scan** command reads a log file and writes a report on its contents. The report's contents depends on the commands issued prior to issuing the **scan** command. (See the description of the **scan** command in Chapter 3.)

Table A-1 on the next page summarizes the information contained in all scan reports and in reports that are produced only when the **detail on** command is used (the default) with the **scan** command.

Table A-1. Information Contained in Scan Reports

Initial Values:	
Initial **parm** file global information and system configuration information.	Printed only if **detail on** is specified.
Initial **parm** file application definitions.	Printed only if **detail on** is specified and you are using MeasureWare Server Agent.
Chronological Detail:	
parm file global change notifications.	Printed only if **detail on** is specified.
parm file application addition/deletion notifications.	Printed only if **detail on** is specified and you are using MeasureWare Server Agent.
Collector off-time notification.	Printed only if **detail on** is specified.
Application-specific summary reports.	Printed only if **detail on** is specified and you are using MeasureWare Server Agent.
Summaries:	
Process log reason summary.	Always printed if process data was scanned and you are using MeasureWare Server Agent.
Scan start and stop actual dates and times.	Always printed.
Application overall summary.	Printed only if application data was scanned and you are using MeasureWare Server Agent.
Collector coverage summary.	Always printed.
Log file contents summary.	Always printed. Includes space and dates covered.
Log file empty space summary.	Always printed.

Scan Report Information

The information in a **utility** scan report is divided into three levels:

- Initial values
- Chronological details
- Summaries

Initial Values

This section describes the following initial values:

- Initial **parm** file global information
- Initial **parm** file application definitions

Initial Parm File Global Information

To obtain this report, use the **scan** command with its default **detail on.**

This report lists the contents of the **parm** file at the time of the earliest global record in the log file. Later global information change notifications are based on the values in this report. If no change notification exists for a particular parameter, it means that the parameter kept its original setting for the duration of the scan.

The example on the next page shows a report listing the contents of the **parm** file.

```
05/23/95 15:28 System ID="Homer"
scopeux/UX A.09.00 SAMPLE INTERVAL = 300,300,60 Seconds, Log version=D
Configuration: 9000/855, O/S HP-UX A.09.00 CPUs=1
Logging Global Process records
        Device= Disk FileSys records

Thresholds: CPU= 10.00%, Disk=10.0/sec, First=1.0 sec, Resp=5.0 sec, Trans=100
            Nonew=FALSE, Nokilled=FALSE, Shortlived=False (<1 sec)

HPUX Parms: Buffer Cache Size = 16384KB, NPROC = 532

Wait Thresholds: CPU=100.00%, Disk=100.00%, Memory=100.00%,

Impede=100.00%

Memory: Physical = 84.0 MB, Swap = 124304.0 MB, Available to users = 66.5 MB
There are 2 LAN interfaces: 0, 1,

05/23/95 15:28 There are 2 Disk Devices:
    Disk #1976          = "/dev/hdisk0"
    Disk #1987          = "/dev/hdisk1"
```

The date and time listed on the first line correspond to the *first date and time* in the global log file and indicate when **scopeux** was started. Data records may have been rolled out of the global log file so the date and time on this report do not necessarily indicate the *first global record* in the log file.

Initial Parm File Application Definitions

To obtain this report, use the **scan** command with its default **detail on** and have application data in the log file (MeasureWare Server Agent only).

This report lists the name and definition of each application at the time the first application record is listed in the log file. Any application addition or deletion notifications you receive are based on this initial list of applications. For example:

```
04/01/95 08:39 Application(1) = "other"
 Comment=all procs not in defined applications


04/01/95 08:39 Application(2) = "Real_Time"
 Priority=0-127
```

```
04/01/95 08:39 Application(3) = "Prog_Development"
 File=vi,ed,sed,xdb,ld,lint,cc,ccom,pc,pascomp
```

Note

During the scan, you are notified of applications that were added or deleted. This decision is based entirely on the application name. (Additions and deletions are determined by comparing the spelling, spacing, and case of the old names to the new set of logged names.) No attempt is made to detect a change in the definition of an application. If an application with a new name is detected, it is listed along with its new definition.

The date and time on this record is the last time **scopeux** was started before logging the first application record currently in the log file.

Chronological Detail

This section describes the following chronological details:

- **parm** file global change notifications
- **parm** file application addition and deletion notifications
- **scopeux** off-time notifications
- Application-specific summary report

Parm File Global Change Notifications

To obtain this report, use the **scan** command with its default **detail on.**

This report is generated any time when a record is found that **scopeux** started. If the current **parm** file collection parameters differ from the parameters when **scopeux** ran last, a global change notification can occur.

The following example shows the change notifications that occur when two new disk drives are added to the system.

```
05/13/95 17:30 The number of disk drives changed from 9 to 11
05/13/95 17:30  New disk device scsi-4 = "c4d0s*"
05/13/95 17:30  New disk device scsi-3 = "c3d0s*"
```

Parm File Application Addition/Deletion Notifications

To obtain this report, use the **scan** command with its default **detail on** and have application data in the log file (MeasureWare Server Agent only).

User-defined applications can be added or deleted each time **scopeux** is started. If an application name is found that does not match the last set of applications, an application addition, deletion, or change notification is printed. If the name of an application has not changed, it is not printed.

Note Application definitions are not checked for changes. They are listed when an application name is changed, but any change to an existing application's definition without an accompanying name change is not detected.

The following example shows that a new application was started.

```
05/13/95 17:30 Application 4 "Accounting_Users_1" was added
User=ted,rebecca,test*,nelson,gene
```

Scopeux Off-Time Notifications

To obtain this report, use the **scan** command with its default **detail on.**

If an extracted file contains only summary information, times are rounded to the nearest hour. For example:

```
04/29/95 11:00 - 04/29/95 12:34 collector off (      01:34:04)
```

The first date and time (04/29/95 11:00) indicate the last valid data record in the log file before **scopeux** was restarted. The second date and time (04/29/95 12:34) indicate when **scopeux** was restarted.

The last field (in parentheses) shows how long **scopeux** was *not* running. The format is *ddd/hh:mm:ss*, where *ddd* are days and *hh:mm:ss* are hours, minutes, and seconds. Zeros to the left are deleted.

In this example, **scopeux** was off on April 29, 1995 between 11:00 AM and 12:34 PM. The summary information shows that data was not collected for 1 hour, 34 minutes, and 4 seconds.

Application-Specific Summary Report

To obtain this report, use the **scan** command with its default **detail on** and have application data in the log file (MeasureWare Server Agent only).

This report can help you define applications. Use the report to identify applications that are accumulating either too many or too few system resources and those that could be consolidated with other applications. Applications that accumulate too many system resources might benefit by being split into multiple applications.

You should define applications in ways that help you make decisions about system performance tuning. It is unlikely that system resources will accumulate evenly across applications.

The application-specific summary report is generated whenever the application definitions change to allow you to access the data of the application defintions before and after the change.

A final report is generated for all applications. This report covers only the time since the last report and not the entire time covered by the log file. For example:

		PERCENT OF TOTAL		
APPLICATION	RECORDS	CPU	DISK	TRANS
OTHER	22385	45.7%	20.9%	63.0%
Resource_Sharing	7531	6.0%	2.2%	17.1%
SPOOLING	13813	2.4%	0.3%	0.0%
ON-LINE_COMPILES	13119	2.9%	1.7%	0.1%
BATCH_COMPILES	8429	2.9%	0.1%	2.2%
ORDER_ENTRY	387	0.1%	0.0%	0.0%
ELECTRONIC_MAIL	6251	3.8%	1.3%	9.6%
PROGRAM_DEVELOPMENT	3141	9.1%	2.4%	0.6%
RESEARCH_DEPARTMENT	3968	8.7%	2.0%	6.0%
BILL_OF_MATERIALS	336	0.6%	0.2%	0.1%
FINANCIALS	1080	5.0%	1.5%	0.5%
MARKETING_DEPT	2712	12.9%	67.3%	0.0%
GAMES	103	0.1%	0.0%	0.6%
ALL USER APPLICATIONS	73.1%	54.3%	79.1%	37.0%

Summaries

This section describes the following summaries:

- Process log reason summary
- Scan start and stop actual dates and times
- Application overall summary
- **scopeux** coverage summary
- Log file contents summary
- Log file empty space summary

Process Log Reason Summary

To obtain this report, you must have process data in the log file (MeasureWare Server Agent only).

This report helps you set the interesting process thresholds for **scopeux**. The report lists every reason a process might be considered interesting and thus get logged, along with the total number of processes logged that satisfied each condition.

The following example shows a process log reason summary report:

```
Process Summary Report: 04/13/95 3:32 PM to 05/04/95 6:36 PM
There were 93.8 hours of process data
Process records were logged for the following reasons:

Log Reason       Records   Percent   Recs/hr
---------------  -------   -------   -------
New Processes      17619     53.9%      44.7
Killed Processes   16047     49.1%      40.7
CPU Threshold       3169      9.7%       8.0
Disk Threshold      1093      3.3%       2.8

NOTE: A process can be logged for more than one reason at a time.
Record counts and percentages will not add up to 100% of the
process records.
```

If the **detail on** command is issued, this report is generated each time a threshold value is changed so you can evaluate the effects of that change. Each report covers the period since the last report. A final report, generated when the scan is finished, covers the time since the last report.

If the **detail off** command is issued, then only one report is generated covering the entire scanned period.

You can reduce the amount of process data logged by **scopeux** by modifying the `parm` file's `threshold` parameter and raising the thresholds of the interest reasons that generate the most process log records. To increase the amount of data logged, lower the threshold for the area of interest.

In the previous example, you could decrease the amount of disk space used for the process data (at the expense of having less information logged) by raising the CPU threshold or setting the `nonew` threshold.

Scan Start and Stop

This summary report is printed if any valid data was scanned. It gives actual dates and times that the scan was started and stopped. For example:

```
Scan started on          03/03/95 12:40 PM
Scan stopped on          05/11/95  1:25 PM
```

Application Overall Summary

To obtain this report, you must have application data in the log file (MeasureWare Server Agent only).

This report is an overall indicator of how much system activity is accumulated in user-defined applications, rather than in the "other" application. If a significant amount of a critical resource is not being captured by user applications, you might consider scanning the

process data for processes that can be included in user applications. For example:

```
OVERALL, USER DEFINED APPLICATIONS ACCOUNT FOR
    82534 OUT OF    112355 RECORDS      ( 73.5%)
    218.2 OUT OF     619.4 CPU HOURS    ( 35.2%)
     24.4 OUT OF      31.8 M DISC IOS   ( 76.8%)
      0.2 OUT OF       0.6 M TRANS     ( 27.3%)
```

Collector Coverage Summary

This report is printed if any valid global or application data was scanned. It indicates how well **scopeux** is being used to capture system activity. If the percentage of time **scopeux** was off is high, as in the example below, you should review your operational procedures for starting and stopping **scopeux**.

```
THE TOTAL TIME COVERED WAS        108/16:14:51 OUT OF 128/00:45:02
TIME LOST WHEN COLLECTOR WAS OFF  19/08:30:11  15.12%
THE SCOPEUX COLLECTOR WAS STARTED    45 TIMES
```

This report will be more complete if global detail data is included in the scan. If only summary data is available, you determine the time **scopeux** was stopped and started only to the nearest hour. (An appropriate warning message is printed with the report if this is the case.)

The total time covered is determined by accumulating all the interval times from the logged data. The `OUT OF TIME` value is calculated by subtracting the starting date and time from the ending date and time. This should represent the total time that could have been logged. The `TIME LOST WHEN COLLECTOR WAS OFF` metric is the total time less the covered time.

The formats for the three times mentioned are:

ddd/hh:mm:ss

where *ddd* are days and *hh:mm:ss* are hours, minutes, and seconds.

In the previous example, the total time collected was 108 days, 16 hours, 14 minutes, and 51 seconds.

Log File Contents Summary

The log file contents summary is printed *if any* valid data was scanned. It includes the log file space and the dates covered. This summary is helpful when you are resizing your log files with the **resize** command.

```
                ------Total------  --Each Full Day-- -------Dates-------   Full
Type            Records MegaBytes  Records MegaBytes  Start       Finish   Days
Global             1376      0.27    288.9     0.057 05/23/95 to 05/28/95  4.8
Application        6931      0.72   1455.0     0.152 05/23/95 to 05/28/95  4.8
Process            7318      1.14   1533.6     0.239 05/23/95 to 05/28/95  4.8
Disk               2748      0.07    567.6     0.014 05/23/95 to 05/28/95  4.8
Overhead                     0.29
               ------- ---------  ------- ---------
TOTAL             18373      2.49  3845.0     0.461
```

The columns are described as follows:

Column	Explanation
Type	The general type of data being logged. One special type `Overhead`, exists: `Overhead` is the amount of disk space occupied (or reserved) by the log file *versus* the amount actually used by the scanned data records. If less than the entire log file was scanned, `Overhead` includes the data records that were not scanned. If the entire file was scanned, `Overhead` accounts for any inefficiencies in blocking the data into the file *plus* any file-access support structures. It is normal for extracted log files to have a higher overhead than raw log files since they have additional support structures for quicker positioning.

Column	Explanation
`Total`	The total record count and disk space scanned for each type of data.
`Each Full Day`	The number of records and amount of disk space used for each 24-hour period that **scopeux** runs.
`Dates`	The first and last valid dates for the data records of each data type scanned.
`Full Days`	The number of full (24-hour) days of data scanned for this data type. `Full Days` may not be equal to the difference between the start and stop dates if **scopeux** coverage did not equal 100 percent of the scanned time.

The `TOTAL` line (at the bottom of the listed data) gives you an idea of how much disk space is being used and how much data you can expect to accumulate each day.

Log File Empty Space Summary

This summary is printed for each log file scanned. For example:

```
The Global      file is now  13.9% full with room for 61 more full days
The Application file is now  15.1% full with room for 56 more full days
The Process     file is now  23.5% full with room for 32 more full days
The Device      file is now   1.4% full with room for 2896 more full days
```

The amount of room available for more data is calculated based on the amount of unused space in the file and the scanned value for the number of megabytes of data being logged each 24-hour day (see the Log File Contents Summary). If the megabytes-scanned-per-day values appear unrealistically low, they are replaced with default values for this calculation.

Note

This report is made on a file-by-file basis. `Overhead`, which was reported as combined overhead in the previous report, is calculated for individual files in this report.

If you scan an extracted file, you get a single report line because all data types share the same extracted file.

Glossary

This glossary contains an alphabetized list of some of the terms associated with MeasureWare Agent.

alarm
An indication of a period of time in which performance meets user-specified alarm criteria. Alarm information can be sent to a PerfView analysis system and to HP OpenView and Operations Center. Alarms can be identified in historical log file data using the **utility** program.

alarm generator
Handles the communication of alarm information. It consists of the `agdbserver` and the `agdb` database that it manages. The `agdb` database contains a list of PerfView analysis nodes (if any) to which alarms are communicated, and various on/off flags that you set to define when and where the alarm information is set.

alarmdef file
The file containing the alarm definitions in which alarm conditions are specified.

application
An application is a user-defined group of related processes.

application log file
See `logappl`.

data source integration (DSI)
The technology that enables MeasureWare Agent to receive, log, and detect alarms on data from external sources such as applications, databases, networks, and other operating systems.

device
A device is an input and/or output device connected to a system. Common devices include disk drives, tape drives, printers, and user terminals.

device log file
See `logdev`.

DSI
See data source integration.

DSI log files
Log files containing self-describing data that are created by MeasureWare Agent's DSI programs.

empty space
The difference between the maximum size of a log file and its current size.

extract
A MeasureWare Agent program that allows you to extract data from raw or previously extracted log files, summarize it, and write it to extracted log files. It also lets you export data for use by analysis programs. See `rxlog`.

extracted log file
A log file created by the **extract** program. It contains user-selected date ranges and types of data. An extracted log file is formatted for optimal access by the workstation analysis tool, PerfView. This file format is suitable for input to the **extract** and **utility** programs and is the preferred method for archiving performance data.

GlancePlus
GlancePlus is an online diagnostic tool that displays current performance data directly to a user terminal or workstation. It is designed to assist you in identifying and troubleshooting system performance problems as they occur.

global
A qualifier that implies the whole system.

global log file
See `logglob`.

interesting process
A process becomes interesting when it is first created, when it ends, and when it exceeds user-defined thresholds for CPU use, disk use, response time, and so on.

logappl
The raw log file that contains summary measurements of the processes in each user-defined application.

logdev
The raw log file that contains measurements of individual device (disk, logical volume, netif) performance.

logglob
The raw log file that contains measurements of the system-wide, or global, workload.

logindx
The raw log file that contains additional information required for accessing data in the other log files.

logproc
The raw log file that contains measurements of selected "interesting" processes. *See* interesting process.

logtran
The raw log file that contains measurements of Transaction Tracker data.

mwa
A shell script that starts and stops MeasureWare Agent processes.

parm file
A MeasureWare Agent file containing the parameters used by **scopeux** to customize data collection.

perflbd.rc
A configuration file that resides in the *datafiles* or *configuration* directory. (See the **scopeux man** page for the paths of these platform-specific directories.) Each entry represents a **scopeux** or DSI data source consisting of a single log file set. (*See* repository server.)

performance alarms
See alarms.

perfstat
A shell script that shows the status of all Hewlett-Packard performance products.

PerfView
PerfView provides integrated performance management for multi-vendor distributed networks. It uses a single workstation to monitor environment performance on networks that range in size from tens to thousands of nodes.

process
Execution of a program file. It can represent an interactive user (processes running at normal, nice, or real-time priorities) or an operating system process.

process log file
See logproc.

raw log file
A file containing summarized measurements of system data. The **scopeux** program collects data into raw log files. See `logglob`, `logappl`, `logproc`, `logdev`, `logtran`, and `logindx`.

real time
The actual time in which an event takes place.

repeat time
An action that can be selected for performance alarms. Repeat time designates the amount of time that must pass before an activated and continuing alarm condition triggers another alarm signal.

repository servers
Servers that provide data to the alarm generator and the PerfView analysis product. There is one repository server for each data source consisting of a **scopeux** or DSI log file set. A default repository server is provided at start up that provides the **scopeux** raw log file set.

resize
Changing the overall size of a raw log file using the **utility** program's **resize** command.

roll back
Deleting one or more days worth of data from a log file, oldest data deleted first. Roll backs are done when a raw log file exceeds its maximum size parameter.

RUN file
The file created by the **scopeux** collector to indicate that the **scopeux** process is running. Removing the RUN file causes **scopeux** to terminate.

rxlog
The default file created by the **extract** program's **extract** command.

scopeux
The MeasureWare Agent collector program that collects data from several sources and writes (logs) this raw measurement data to raw log files.

scopeux log files
The six log files that are created by the **scopeux** collector: `logglob`, `logappl`, `logproc`, `logdev`, `logtran`, and `logdev`.

status.scope
The file created by the **scopeux** collector to record status, data inconsistencies, or errors.

system ID
The string of characters that identifies your system. The default is the host name as returned by `uname`.

Transaction Tracker
The technology used in MeasureWare Agent that allows information technology (IT) resource managers to measure end-to-end response time of business application transactions.

utility
A MeasureWare Agent program that lets you open, resize, scan, and generate reports on raw and extracted log files. You can also use it to check `parm` file syntax, check the `alarmdef` file syntax, and obtain alarm information from historical log file data.

Index

Customer Comment Card

MeasureWare Agent: User's Manual

HP Part No.: B4967-90001

E1195

We welcome your evaluation of this manual. Your comments and suggestions will help us improve our publications. Attach additional pages if necessary.

Please circle the following Yes or No:

■ Is this manual well organized?	Yes	No
■ Is the information technically accurate?	Yes	No
■ Are instructions complete?	Yes	No
■ Are concepts and wording easy to understand?	Yes	No
■ Are the examples and pictures helpful?	Yes	No
■ Are there enough examples and pictures?	Yes	No

Additional Comments: _____

_____

Please provide:

Name: _____ Title: _____

Company: _____ Address: _____

City: _____ State: _____ Zip Code/Country _____

Please send to:

HEWLETT PACKARD

Thank you for your assistance.

No postage is required. Just remove this card, fold so that the pre-addressed label is on the outside, secure and mail.

Tape closed _ _ _ _ _ PLEASE DO NOT STAPLE _ _ _ _ _ Tape closed

...Fold here.....Fold here...

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

PERMIT No. 256 ROSEVILLE CA

- POSTAGE WILL BE PAID BY ADDRESSEE -

LEARNING PRODUCTS MANAGER
HEWLETT-PACKARD COMPANY M/S 5726
SUPPORT TECHNOLOGY CENTER
8000 FOOTHILLS BOULEVARD
ROSEVILLE CA 95747-9987

Fold here Fold here

Detach or fold here ->

Recorder No. or
Manual Part No.
B4967-90001

Manufacturing No.
B4967-90001

B4967-90001

15% Recycled Paper
15% Post-Consumer Waste