HP Performance Agent

For the Windows® Operating System

Software Version: 4.70

Data Source Integration Guide

Document Release Date: September 2007
Software Release Date: September 2007

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

Trademark Notices

Microsoft® is U.S. registered trademark of Microsoft Corporation.

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

Adobe® and Adobe Acrobat® are trademarks of Adobe Systems Incorporated.

Hewlett-Packard Company, United States.

All other product names are the property of their respective trademark or service mark holders and are hereby acknowledged

Support

You can visit the HP Software Support web site at:

www.hp.com/go/hpsoftwaresupport

HP Software online support provides an efficient way to access interactive technical support tools. As a valued support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Contents

1 Overview of Data Source Integration

Data Source Integration (DSI) technology allows you to use HP Performance Agent to log data, define alarms, and access metrics from new sources of data beyond the metrics logged by the Performance Agent `scopeux` collector. Metrics can be acquired from data sources such as databases, LAN monitors, and end-user applications.

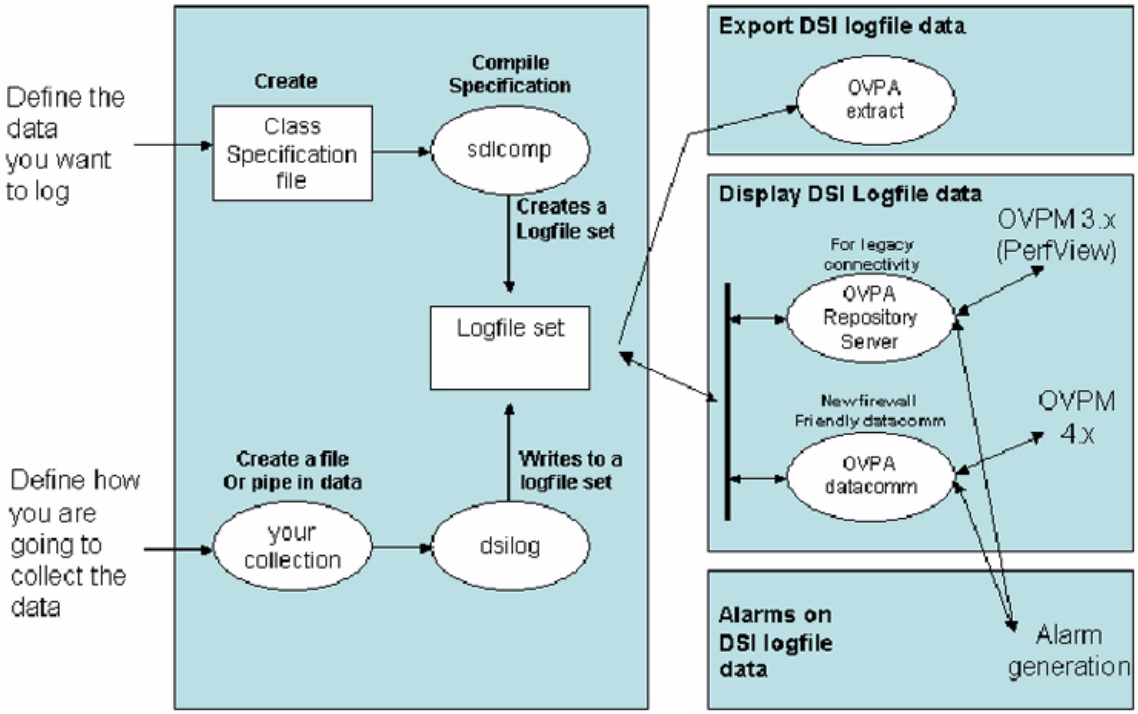
The data you log using DSI can be displayed in HP Performance Manager along with the standard performance metrics logged by the `scopeux` collector. DSI logged data can also be exported, using the Performance Agent extract program, for display in spreadsheets or similar analysis packages.

Performance Manager in this document refers to version 4.0 and beyond for UNIX and Windows platforms. Performance Manager 3.x (PerfView) will connect to Performance Agent 4.0 and beyond for all UNIX platforms except for Performance Agent for Linux. In the future, connectivity to Performance Manager 3.x will be discontinued.

How DSI Works

The following diagram shows how DSI log files are created and used to log and manage data. DSI log files contain self-describing data that is collected outside of the Performance Agent `scopeux` collector. DSI processes are described in more detail on the next page.

Figure 1 Data Source Integration Process



Using DSI to log data consists of the following tasks:

Creating the Class Specification

You first create and compile a specification for each class of data you want to log. The specification describes the class of data as well as the individual metrics to be logged within the class. When you compile the specification using the DSI compiler, `sdlcomp`, a set of empty log files are created to accept data from the `dsilog` program. This process creates the log file set that contains a root file, a description file, and one or more data files.

Collecting and Logging the Data

Then you collect the data to be logged by starting up the process of interest. You can either pipe the output of the collection process to the `dsilog` program directly or from a file where the data was stored. `dsilog` processes the data according to the specification and writes it to the appropriate log file. `dsilog` allows you to specify the form and format of the incoming data.

The data that you feed into the DSI process should contain multiple data records. A record consists of the metric values contained in a single line. If you send data to DSI one record at a time, stop the process, and then send another record, `dsilog` can append but cannot summarize the data.

Using the Data

You can use Performance Manager to display DSI log file data. Or you can use the Performance Agent extract program to export the data for use with other analysis tools. You can also configure alarms to occur when DSI metrics exceed defined conditions. For more information about exporting data and configuring alarms, see the *HP Performance Agent for Windows*.

2 DSI Configuration and Management

This chapter is an overview of what you need to do to successfully implement data source integration. The following topics are covered:

- Planning data collection
- Defining the log file format
- Creating the empty log file set
- Logging data to the log file set
- Setting up the DSI service
- Using the logged data

Planning Data Collection

Before creating the DSI class specification files and starting the logging process, you need to address the following topics:

- Understand your environment well enough to know what kinds of data would be useful in managing your computing resources
- What is data available
- Where is the data
- How can you collect data
- What are the delimiters between data items. For proper processing by `dsilog`, metric values in the input stream must be separated by blanks (the default) or a user-defined delimiter.
- What is the frequency of collection
- How much space is required to maintain logs
- Which alarms do you want to be generated and under what conditions
- What options you have for logging with the class specification and the `dsilog` process
- What is the output of the program or process that you use to access the data

Defining the Log File Format

Once you have a clear understanding of what kind of data you want to collect, create a class specification to define the data to be logged and to define the log file set that will contain the logged data. You enter the following information in the class specification.

- Data class name and ID number
- Label name (optional) that is a substitute for the class name. (For example, if a label name is present, it can be used in Performance Manager.
- Increments by which to store new and roll old data. These specifications include how you want the data summarized if you want to limit the number of records logged per hour. See How Log Files Are Organized on page 14.
- Metric names and other descriptive information, such as how many decimals to allow for metric values.
- How you want the data summarized if you want to log a limited number of records per hour.

Here is an example of a class specification:

```
CLASS SYS_STATS = 10001

LABEL "STATUS data"

INDEX BY HOUR

MAX INDEXES 12

ROLL BY HOUR

RECORDS PER HOUR 120;
```

Here is an example of one metric description with the data class just named. Many other metric descriptions could be included in this same file, each separated by a semi-colon:

```
METRICS

RUN_Q_PROCS    = 106

LABEL   "Procs in run q"

PRECISION 0;

BLOCKED_PROCS   = 107

LABEL   "Blocked Processes"

PRECISION 0 ;
```

You can include one class or multiple classes in a class specification file. When you have completed the class specification file, name the file and then save it. When you run the DSI compiler, `sdlcomp`, you use this file to create the log file set. For more information about the class specification and metric description syntax, see Chapter 3, DSI Class Specification Reference.

You set indexes to control the frequency of rolling the data, which occurs in blocks (indexes) of data rather than in individual records. The number you assign to MAX INDEXES sets the storage capacity for each class separately, even if you have specified that multiple classes should be stored in a single log file set. When the storage capacity is reached, the class is "rolled," meaning the oldest records in the class are deleted.

You can specify actions that would execute when the class is rolled. For example, you might export the data to an archive file.

How Log Files Are Organized

Log files are organized into classes. Each class, which represents one source of incoming data, consists of a group of data items (metrics) that are logged together. Each record, or row, of data in a class represents one sample of the values for that group of metrics.

The data for classes is stored on disk in log files that are part of the log file set. The log file set contains a root file, a description file, and one or more log files. All the data from a class is always kept in a single data file. However, when you provide a log file set name to `sdlcomp`, you can store multiple classes together in a single log file set or in separate log file sets. The figure below illustrates how two classes can be stored in a single log file set.

Since each class is treated as a circular log file, you can set the storage capacity for each class separately, even if you have specified that multiple classes should be stored in a single log file set. When the storage capacity is reached, the class is "rolled," which means the oldest records in the class are deleted to make room for new data.

You can specify actions, such as exporting the old data to an archive file, to be performed whenever the class is rolled.

Creating the Empty Log File Set

The DSI compiler, `sdlcomp`, uses the class specification file to create or update an empty log file set. The log file set is then used to receive logged data from the `dsilog` program.

To create the log file set, complete the following tasks:

- Run `sdlcomp` with the appropriate variables and options. For example,
  `sdlcomp [-maxclass` *value*`]` **`specification_file`**
  `[logfile_set [`*log file*`]] [`***options*`]**
- Check the output for errors and make changes as needed.

For more information about `sdlcomp`, see Sdlcomp Compiler on page 43.

Testing the Class Specification File and the Logging Process (optional)

DSI uses a program, `sdlgendata`, that allows you test your class specification file against an incoming source of randomly generated data. You can then examine the output of this process to verify that DSI can log it according to your specifications.

To test your class specification file for the logging process:

1. Feed the data that is generated by `sdlgendata` to the `dsilog` program. (See Testing the Logging Process with Sdlgendata on page 52, for details on the program options.) The syntax is:

   `sdlgendata logfile_set.class -vo | dsilog logfile_set.class`

2. Check the output to see if your class specification file matches the format of your data collection process. If the `sdlgendata` program outputs something different from your program, you have either made a mistake in how you have formatted the output, or you have made a mistake in the specification file.
3. Delete all log files generated from the testing process before collecting real data.

Logging the Data to the Log File Set

After you have created the log file set and tested it by generating data for it to store, update Performance Agent files as needed, and then start the `dsilog` process to log incoming data.

1. Update the data source configuration file, `datasources`, to add the DSI log files as data sources for generating alarms or to view from a central Performance Manager analysis system. For more information about `datasources`, see Configuration Files on page 47.

   The `perflbd.mwc` file is maintained as a link to the `datasources` file for Performance Agent on all supported Windows operating systems, except Performance Agent on 64 bit Windows. On 64 bit Windows, user has to use `perflbd.mwc` instead of datasources.

2. Modify the alarm definitions file, `alarmdef.mwc`, if you wan to alarm on specific DSI metrics. For more information, see Defining Alarms on DSI Metrics on page 47.

3. Start the collection process from the Windows NT/2000 Command Prompt.

4. Pipe the data from the collection process to `dsilog` (or some other way to get it to `stdin`) with the appropriate variables and options set. For example:

   `<program or process with variables>|dsilog logfile_set class`

The `dsilog` program is designed to receive a continuous stream of data. Therefore, it is important to structure scripts so that `dsilog` receives continuos input data. Do not write scripts that create a new `dsilog` process for each new input data point. Doing so can cause duplicate timestamps to be written to the `dsilog` file, which can cause problems for Performance Manager and perfalarm when reading the file. See Chapter 5, Examples of Data Source Integration, for examples of problematic and recommended scripts.

For more information about `dsilog` options, see DSI Logging Processes on page 50.

Setting Up the DSI Service

Performance Agent includes a service that allows DSI processes to continually log data after the user logs off the system. This service starts up automatically as long as your system is running. You can set the service startup to manual or automatic in the Windows NT/2000 Control Panel. (The default setting is automatic.)

Before using the DSI service, you need to configure the DSI configuration file, `\rpmtools\data\dsiconf.mwc`. This configuration involves entering information relating to all DSI log files that you have implemented. The information you enter includes the program or process you are using to collect the data, the name of the file where that data is stored, and the name of the data class (as defined in the DSI class specification file)

The `< rpmtools >` directory name is used throughout this document and stands for the directory in which Performance Agent is installed. The default directory is *<disk drive>*`:\Program Files\hp OpenView` but you can specify a different installation path for a first time installation

For more information, see Configuring Continuous Logging of DSI Data on page 48.

Using the Logged Data

Once you have created the DSI log files, you can use the Performance Agent graphical interface to export the data. You can also configure alarms to occur when metrics logged with DSI exceed defined conditions.

Here are some ways to use logged DSI data:

- Export the data for use in reporting tools such as spreadsheets.
- Display exported DSI data using analysis tools such as Performance Manager.
- Monitor alarms using HP Performance Manager, Operations Manager, or Network Node Manager.

For information about exporting data, see Chapter 4, DSI Program Reference of this manual and Chapter 3 of the *MeasureWare Agent for Windows NT/2000: User's Manual*. For information about defining alarm conditions, see Chapter 8 of the *MeasureWare Agent for Windows NT/2000: User's Manual*. For details on displaying DSI data and alarms in Performance Manager, Network Node Manager, and ITO, see Performance Manager online Help

You *cannot* create extracted log files from DSI log files. You can only export the data.

Moving DSI Log File Sets From One Windows System to Another

In the event you find it necessary to move a DSI log file set from one Windows system to another, we recommend that you follow the following guidelines.

These guidelines describe which types of moves do not require rebuilding a DSI log file set (safe moves) and which types of move do require rebuilding a DSI log file set (non-safe moves). Proper rebuilding procedures for non-safe moves are included in the guidelines.

In the text below, a "DSI log file set" refers to the root file, class description file, and data class files. The `logfile_set` parameter in `sdlcomp` contains the name of the DSI log file set. For additional information about the structure of DSI log file sets, see Chapter 4, DSI Program Reference.

Safe vs. Non-safe Moves

Movement from source System A to destination System B (same directory path)

This is a safe move, subject to drive letter constraints (see below). Move all member files of the DSI log file set using the transfer mechanism of choice.

Movement from source System A to destination System B (different directory path)

This is a safe move, subject to drive letter constraints (see below), if the DSI log file was originally compiled (using `sdlcomp`) without the use of an absolute path name for the `sdlcomp logfile_set` parameter.

An example of using an absolute path name for the `sdlcomp logfile_set` parameter (not safe) is:

```
sdlcomp spec1 \sample_sdl
```

An example of using a relative path name for the `sdlcomp logfile_set` parameter (safe) is:

```
sdlcomp spec1 relpath\sample_sdl
```

And, an example of using no path name with the `sdlcomp logfile_set` parameter (safe) is:

```
sdlcomp spec1 sample_sdl
```

If you do not know which syntax was used for the original compilation of the `logfile_set` parameter, use the following command to determine if an absolute path name (not safe) was used:

```
sdlutil sample_sdl -files
```

If file names are listed with absolute paths, this indicates that an absolute path name (not safe) was used originally for compilation.

Drive letter constraints

When Performance Agent opens a DSI log file set for reading on a Windows system, it is assumed that files reside on the default system drive (usually C:) unless both the drive letter and full path name were specified for the logfile_set parameter when the log file set was originally compiled. Therefore, moving a DSI log file set from System A's C: drive to System B's D: drive typically requires rebuilding of the DSI log file set and requires compilation syntax for the rebuilt log file set to be similar to the following:

```
sdlcomp spec1 D:\full\path\name\sample_sdl
```

Roll Actions

If a roll action is specified for a class and the action contains any absolute path names (such as the destination of an extracted log file), then this destination directory must exist on the system to which the DSI log file set is being moved. If for some reason it is not desirable to create this directory, the action statement will need to be modified. Modification of the action statement requires rebuilding of the DSI log file set.

Here is an example of an absolute path name in an action statement:

```
ACTION "extract -l sample_sdl -C class1 "
"-B $PT_START$ -E $PT_ENDS$ -f C:\mydir\myfile, purge”
```

Procedure for Rebuilding to Move DSI Log File Sets

As stated previously, the majority of DSI log file sets can be safely moved by relocating the member files of the set. The following procedure can be used to move the non-safe corner cases described previously. This procedure preserves the data contained in the DSI log file set.

This discussion assumes the following:

- Logfile set name (SDL name) is `sample_sdl`.
- There are two classes, named class1 and class2.

a. Export the data contents for each class using an extract Command Prompt argument. The data for each class is written to a separate data file. Example syntax is:

```
extr act -l sample_sdl -C class1 DETAIL -ut -h -f class1.data,
purge -xp
extract -l sample_sdl -C class2 DETAIL -ut -f -f class2.data,
purge -xp
```

b. Use the exact syntax as shown. The `-ut` switch provides the required timestamp format for successful reloading of data. The `-h` switch disables header information, which is also required for successful reloading.

c. If no original specification files exist, recreate a spec file for each class using sdlutil. Example syntax is:

```
sdlutil sample_sdl -decomp class1 > spec1
sdlutil sample_sdl -decomp class2 > spec2
```

d. Move the class data and spec files to the new system at the directory path where the DSI log file set is to be rebuilt.

e. For each class, compile using the class spec file. Example syntax is:

```
sdlcomp spec1 sample_sdl
sdlcomp spec2 sample_sdl
```

If the DSI log file set will reside on a drive other than the system default drive, use the following example syntax:

```
sdlcomp spec1 D:\full\path\name\sample_sdl
sdlcomp spec2 D:\full\path\name\sample_sdl
```

f For each class, reload the data using the following example syntax:

```
dsilog sample_sdl class1 -i class1.data -timestamp
dsilog sample_sdl class2 -1 class2.data -timestamp
```

g Make any required changes to the Performance Agent datasources and `dsiconf.mwc` files.

The rebuilt DSI log file set is now ready to use.

3 DSI Class Specification Reference

This chapter provides detailed information about:

- Class specifications
- Class specification syntax
- Metric descriptions in the class specifications.

Class Specifications

For each source of incoming data, you must create a class specification file to describe the format for storing incoming data. To create the file, use the class specification language described in the next section Class Specification Syntax, to create the file. The class specification file contains:

- a class description that groups the metrics for the data source. The class description assigns a name and numeric ID to the incoming data set, determines how much data will be stored, and specifies when to roll data to make room for new.
- metric descriptions for each individual data item. A metric description names and describes each data item. It also specifies the summary level to apply to data (`RECORDS PER HOUR`) if more than one record arrives in the time interval configured for the class.

To generate the class specification file, use any editor or word processor that lets you to save the file as an ASCII text file.

You specify a name for the log file set (based on your class specification file) when you run `sdlcomp` and specify the class name. When you start logging data, you must run the `dsilog` process for every class you of data you want to include in the log file set. When the class specification is compiled, it automatically creates or updates a log file set for storage of the data.

The class specification allows you to determine how many records per hour will be stored for the class, and to specify a summarization method to be used if more records arrive than you want to store. For instance, if you have requested that 12 records per hour be stored and records arrive every minute, you could have some of the data items averaged and others totaled to maintain a running count.

The DSI compiler, `sdlcomp`, creates files with the following names for a log file set (named *logfile_set_name*):

```
logfile_set_name
logfile_set_name.desc
```

`Sdlcomp` creates a file with the following default name for a class (named *class_name*):

```
logfile_set_name.class_name
```

Avoid the use of class specification file names that conflict with these naming conventions, or `sdlcomp` will fail.

Class Specification Syntax

The following example explains how to set up a class specification file, where you name a group of metrics (class name) and then define the individual metrics within the group (metric names and metric ids. You can define only one class of metrics in a class specification file.

Syntax statements shown in brackets [ ] are optional. Multiple statements shown in braces { } indicate that one of the statements must be chosen. Italicized words indicate a variable name or number you enter. Commas can be used to separate syntax statements for clarity anywhere except directly preceding the semicolon, which marks the end of the class specification and the end of each metric specification. Statements are *not* case-sensitive.

Comments start with # or //. Everything following a # or // on a line is ignored. Note the required semicolon after the class description and after each metric description. Detailed information about each part of the class specification and examples follow.

```
CLASS class_name = class_id_number
LABEL "class_label_name"

[ INDEX BY { HOUR | DAY | MONTH } MAX INDEXES number
[ [ ROLL  BY { HOUR | DAY | MONTH } [ ACTION “action” ]

[ CAPACITY { maximum_record_number } ]

[ RECORD PER HOUR number ]

;

METRICS

metric_name = metric_id_number
[ LABEL "metric_label_name" ]
[ TOTALED | AVERAGED | SUMMARIZED BY metric_name ]
[ MAXIMUM  metric_maximum_number ]
[ PRECISION {0 | 1 | 2 | 3 | 4 | 5} ]
[ TYPE TEXT LENGTH "number" ]
;
```

The metric maximum value does not limit logged values; this setting provides Performance Manager with an estimated maximum range for graphing the logged metric values.

CLASS Description

To create a class description, assign a name to a group of metrics from a specific data source, specify the capacity of the class, and designate how data in the class will be rolled when the capacity is reached.

You must begin the class description with the CLASS keyword. The final parameter in the class specification must be followed by a semicolon (;).

Syntax

```
CLASS class_name = class_id_number
[ LABEL "class_label_name" ]

[ INDEX BY { HOUR | DAY | MONTH } MAX INDEXES number
[ [ ROLL BY { HOUR | DAY | MONTH } [ACTION "action"]

[ CAPACITY {maximum_record_number} ]
[ RECORDS PER HOUR  number ]
;
```

Default Settings

The default settings for the class description are:

```
LABEL class_name

   INDEX BY DAY

MAX INDEXES 9

RECORDS PER HOUR 12
```

To use the defaults, enter only the CLASS keyword with a *class_name* and *numeric class_id_number*.

CLASS

The class name and class ID identify a group of metrics from a specific data source.

Syntax

```
CLASS  class_name = class_id_number
```

How to Use It

The *class_name* and *class_ID_number* must meet the following requirements:

- *class_name* is alphanumeric and can be up to 20 characters long. The name must start with an alphabetic character and can contain underscores (but no special characters).
- *class_ID_number* must be numeric and can be up to six digits long.
- Neither the *class_name* or the *class_ID_number* are case-sensitive.

- The *class_name* and *class_id_number* must each be unique among all the classes you define and cannot be the same as any application defined in the Performance Agent `parm.mwc` file. (For information about application parameters in the `parm.mwc` file, see Chapter 2 of the *MeasureWare Agent for Windows NT/2000: User's Manual*.

Example

```
CLASS SYS_STATS = 10001;
```

LABEL

The class label identifies the class as a whole. It is used instead of the class name in Performance Manager.

Syntax

[LABEL "*class_label_name*"**]**

How To Use It

The *class_label_name* must meet the following requirements:

- It must be enclosed in double quotation marks.
- It can be up to 48 characters long.
- It cannot be the same as any of the keyword elements of the DSI class specification such as `CAPACITY` or `ACTION`.
- If it contains a double quotation mark, precede it with a backslash (\), For example, you would enter **\"my\"** `data` if the label is `"my"` data.
- If no label is specified, the class name is used as the default.

Example

```
CLASS SYS_STATS = 10001
LABEL "STATUS data";
```

INDEX BY, MAX INDEXES, AND ROLL BY

`INDEX BY`, `MAX INDEXES`, and `ROLL BY` settings allow you to specify how to store data and when to delete it. With these settings you designate the blocks of data to store, the maximum number of blocks to store. and the size of the block of data to discard when data reaches it maximum index value.

Syntax

INDEX BY

[ROLL BY

{ HOUR | DAY | MONTH } MAX INDEXES *number*

{ HOUR | DAY | MONTH } [ACTION "*action*" **]**

How To Use It

INDEX BY settings allow blocks of data to be rolled when the class capacity is reached. The INDEX BY and RECORDS PER HOUR options can be used to indirectly to set the capacity of the class as described later in Controlling Log File Size on page 31.

The INDEX BY setting cannot exceed the ROLL BY setting. For example, INDEX BY DAY does not work with ROLL BY HOUR, but INDEX BY HOUR does work with ROLL BY DAY .

If ROLL BY is not specified and the record capacity of the class is reached, each record added to the class overwrites the oldest record. If ROLL BY is specified, when the capacity is reached, all the records logged in the oldest roll interval are freed for reuse. This causes the roll to happen less frequently.

Any specified ACTION is performed before the data is discarded (rolled). This optional ACTION could be used to export the data to another location before it is deleted. See Chapter 4, DSI Program Reference for information on exporting data.

Notes on Roll Actions

Do not specify a command in the ACTION statement that will cause a long delay, since new data won't be logged during the delay.

If the command is more than one line long, mark the start and end of each line with double quotation marks. Be sure to include spaces where necessary inside the quotation marks to ensure that the various command line options remain separated when the lines are concatenated.

If the command contains a double quotation mark, precede it with a backslash (\). If the action contains a directory path, use double-backslashes for the path delimiter; for example, instead of **C:\mydata\myfile**, use **C:\\mydata\\myfile**. If you are using the DSI service, always use full path names in your ACTION statement.

The ACTION statement is limited to 199 characters or less.

Within the ACTION statement, you can use macros to define the time window of the data to be rolled out of the log file. These macros are expanded by dsilog. You can use PT_START to specify the beginning of the block of data to be rolled out in UNIX time (seconds since 1/1/70 00:00:00) and PT_END to specify the end of the data in UNIX time. These are particularly useful when combined with the Performance Agent's extract program in the Command Prompt to export the data before it is overwritten.

If MAX INDEXES is 1 and a roll action is specified, sdlcomp sets MAX INDEXES to 2.

Examples

The following examples may help to clarify the relationship between the INDEX BY, MAX INDEXES, and RECORDS PER HOUR clauses.

The following example indirectly sets the CAPACITY to 144 records (1 hour [index value setting] multiplied by 12 records per hour [equals one index] multiplied by the maximum of 12 indexes equals 144 records).

```
CLASS SYS_STATS = 10001

LABEL "STATUS data"

INDEX BY HOUR

MAX INDEXES 12

RECORDS PER HOUR 12;
```

The following example indirectly sets the CAPACITY to 1440 records (1* 12 * 120).

```
CLASS SYS_STATS = 10001
LABEL "STATUS data"
INDEX BY HOUR
MAX INDEXES 12
RECORDS PER HOUR 120;
```

The following example shows ROLL BY HOUR.

```
CLASS SYS_STATS = 10001
LABEL "STATUS data"
INDEX BY HOUR
MAX INDEXES 12
ROLL BY HOUR
RECORDS PER HOUR 120;
```

The following example causes all the data currently identified for rolling (excluding weekends) to be exported to a file called sys.sdl before the data is overwritten. Note that the last lines of the example are enclosed in double quotation marks to indicate that they form a single command.

```
CLASS SYS_STATS = 10001
LABEL "STATUS data"
INDEX BY HOUR
MAX INDEXES 12
ROLL BY HOUR
ACTION "extract -l sdl_new -C SYS_STATS "
"-B $PT_START$ -E $PT_END$ -f sys.sdl, purge -we 17"
RECORDS PER HOUR 120;
```

Other Examples

Some suggested index settings below offer you examples for considering how much data you want to store. The second table offers a detailed explanation of other settings with the ROLL BY setting added

Table 1 Index Settings Examples.

Index by	Maximum Indexes	Amount of data stored
Hour	72	3 days
Hour	168	7 days
Hour	744	31 days
Day	365	1 year
Month	12	1 year

Table 2 Examples of ROLL BY Settings

INDEX BY	MAX INDEXES	ROLL BY	MEANING
Day	9	Day	Nine days of data will be stored in the log file. Before logging day 10, day 1 is rolled out. These are the default values for index and max indexes.
Hour	72	Hour	72 hours (three days) of data will be stored in the log file. Before logging hour 73, hour 1 is rolled out. Thereafter, at the start of each succeeding hour, the "oldest" hour is rolled out.
Hour	168	Day	168 hours (seven days) of data will be stored in the log file. Before logging hour 169 (day 8), day 1 is rolled out. Thereafter, at the start of each succeeding day, the "oldest" day is rolled out.
Hour	744	Month	744 hours (31 days) of data will be stored in the log file. Before logging hour 745 (day 32), month 1 is rolled out. Thereafter, before logging hour 745, the "oldest" month is rolled out. For example: `dsilog` is started on April 15 and logs data through May 16 (744 hours). Before logging hour 745 (the first hour of May 17), `dsilog` will roll out the data for the month of April (April 15 - 30).

INDEX BY	MAX INDEXES	ROLL BY	MEANING
Day	30	Day	30 days of data will be stored in the log file. Before logging day 31, day 1 is rolled out. Thereafter, at the start of each succeeding day, the "oldest" day is rolled out. For example: If `dsilog` is started on April 1 and logs data all month, then the April 1st data will be rolled out when May 1st (day 31) data is to be logged.
Day	62	Month	62 days of data will be stored in the log file. Before logging day 63, month 1 is rolled out. Thereafter, before logging day 63 the "oldest" month is rolled out. For example: If `dsilog` is started on March 1 and logs data for the months of March and April, there will be 61 days of data in the log file. Once `dsilog` logs May 1st data (the 62nd day), the log file will be full. Before `dsilog` can log the data for May 2nd, it will roll out the entire month of March.
Month	2	Month	Two months of data will be stored in the log file. Before logging the third month, month 1 is rolled out. Thereafter, at the start of each succeeding month, the "oldest" month is rolled out. For example: If `dsilog` is started on January 1 and logs data for the months of January and February. Before `dsilog` can log the data for March, it will roll out the month of January.

Controlling Log File Size

You determine how much data is stored in each class and how much data to discard to make room for new data. Class capacity is calculated from `INDEX BY` (hour, day, or month), `RECORDS PER HOUR`, and `MAX INDEXES`. The following examples show the results for different settings:

In this example, the class capacity is 288 (24 indexes * 12 records per hour).

```
INDEX BY HOUR
MAX INDEXES 24
RECORDS BY HOUR 12
```

In this example, the class capacity is 504 (7 days * 24 hours per day * 3 records per hour).

```
INDEX BY DAY
MAX INDEXES 7
RECORDS PER HOUR 3
```

In this example, the class capacity is 14880 (2 months * 31 days per month * 24 hours per day * 10 records per hour).

```
INDEX BY MONTH
MAX INDEX 2
RECORDS PER HOUR 10
```

If you do not specify values for `INDEX BY`, `RECORDS PER HOUR`, and `MAX INDEXES`, DSI uses the defaults for the class descriptions. See Default Settings under Class Specifications on page 24 earlier in this chapter.

The `ROLL BY` option lets you determine how much data to discard each time the class record capacity is reached. The setting for `ROLL BY` is constrained by the `INDEX BY` setting in that the `ROLL BY` unit (hour, day, or month) cannot be smaller than the `INDEX BY` unit.

The following example illustrates how rolling occurs given the sample

```
INDEX BY DAY
MAX INDEXEX 6
ROLL BY DAY
```

Class_3 Example log
Day 2-21 records
Day 3-24 records
Day 4-21 records
Day 5-24 records
Day 6-21 records

In the above example, the class capacity is limited to six days of data by the setting:

```
MAX INDEXES 6.
```

The deletion of data is set for a day's worth by the setting:

```
ROLL BY DAY.
```

When the seventh day's worth of data arrives, the oldest day's worth of data is discarded. Note that in the beginning of the logging process, no data is discarded. After the class fills up for the first time at the end of 7 days, the roll takes place once a day.

RECORDS PER HOUR

The `RECORDS PER HOUR` setting determines how many records are written to the log file every hour. The default number for `RECORDS PER HOUR` is 12 to match MeasureWare's measurement interval of data sampling once every 5 minutes (60 minutes/12 records = logging every five minutes).

The default number or the number you enter could require the logging process to summarize data before it becomes part of the log file. The method used for summarizing each data item is specified in the metric definition as described in Summarization Method on page 37 later in this chapter.

Syntax

[RECORDS PER HOUR *number*]

How To Use It

The logging process uses this value to summarize incoming data to produce the number of records specified. For example, if data arrives every minute and you have set RECORDS PER HOUR to **6** (every 10 minutes), 10 data points are summarized to write each record to the class. See some common RECORDS PER HOUR settings below:

```
RECORDS PER HOUR 6   ‡ 1 record/10 minutes
RECORDS PER HOUR 12  ‡ 1 record/5 minutes
RECORDS PER HOUR 60  ‡ 1 record/minute
RECORDS PER HOUR 120 ‡ 1 record/30 seconds
```

Notes

RECORDS PER HOUR can be overridden by the -s seconds option in dsilog. However, overriding the original setting could cause problems with Performance Manager graphing the data.

If dsilog has received no metric data for a for an entire logging interval, a missing data indicator is logged for that metric. DSI can be forced to use the last value logged with the -asyn option in dsilog. For a description of the -asyn option, see DSI Logging Processes on page 50.

Example

In this example, a record will be written every 10 minutes.

```
CLASS SYS_STATS = 10001
LABEL "STATUS data"
RECORDS PER HOUR 6;
```

CAPACITY

CAPACITY is the number of records to be stored in the class.

Syntax

[CAPACITY (*maximum_record_number***)]**

How To Use It

Class capacity is derived from the settings RECORDS PER HOUR, INDEX BY, and MAX INDEXES. The CAPACITY setting is ignored unless a capacity larger than the derived value of these other settings is necessary. If this situation occurs, the MAX INDEXES setting is increased to provide the specified capacity.

Example

```
INDEX BY DAY
MAX INDEXES 9
RECORDS PER HOUR 12
CAPACITY 3000
```

In the above example, the derived class capacity is 2,592 records (9 days * 24 hours per day * 12 records per hour).

Because 3,000 is greater than 2,592, `sdlcomp` increases `MAX INDEXES` to `11`, resulting in the class capacity of 3,168. After compilation, you can see the resulting `MAX INDEXES` and `CAPACITY` values by running `sdlutil` with the `-decomp` option.

METRIC Descriptions

The metric descriptions in the class specification file are used to define the individual data items for the class. The metric description equates a metric name with a numeric identifier. The metric description also specifies the summarization method to use when the number of records per hour exceeds the number specified in the `RECORDS PER HOUR` setting.

There is a maximum limit of 100 metrics in the `dsilog` format file

METRICS

```
metric_name = metric_id_number
[LABEL "metric_label_name" ]
[TOTALED | AVERAGED | SUMMARIZED BY metric_name]
[MAXIMUM metric_maximum_number]
[PRECISION { 0 | 1 | 2 | 3 | 4 | 5 } ]
TYPE TEXT LENGTH "number"
```

For numeric metrics you can specify the summarization method (`TOTALED`, `AVERAGED`, `SUMMARIZED BY`), a `MAXIMUM`, and `PRECISION`. For text metrics you can specify only the `TYPE TEXT LENGTH`.

METRIC

The metric name and ID number identify the metric being collected.

Syntax

`METRIC`

metric_name **=** *metric_id_number*

How To Use It

The metrics section must start with the **`METRICS`** keyword before the first metric definition. Each metric must have a metric name that meets the following requirements:

- Must not exceed 20 characters
- Must begin with an alphabetic character
- Can contain only alphanumeric characters and underscores
- Is not case-sensitive.

The metric also has a metric ID number that must not be longer than 6 characters.

The *metric_name* and *metric_id_number* must each be unique among all the metrics you define in the class. The combination *class_name:metric_name* must be unique for this system, and it cannot be the same as any *application_name:metric_name*.

Each metric description is separated from the next by a semicolon (;).

You can reuse metric names from any other class whose data is stored in the same log file set if the definitions are identical as well (see How Log Files Are Organized on page 14). To reuse a metric definition that has already been defined in another class in the same log file set, specify just the *metric_name* without the *metric_id_number* or any other specifications. If any of the options are to be set differently than the previously defined metric, the metric must be given a unique name and number and redefined.

The order of the metric names in this section of the class specification determines the order of the fields when you export the logged data. If the order of incoming data is different from the order you list in this specification, or you do not want to log all the data in the incoming data stream, see Chapter 4, DSI Program Reference for information about how to map the metrics to the correct location.

A timestamp metric is automatically inserted as the first metric in each class. If you want the timestamp to appear in a different position in exported data, include the short form of the internally defined metric definition (`DATE_TIME;`) in the position you want it to appear. To omit the timestamp and use a UNIX timestamp (seconds since 1/1/70 00:00:00) that is part of the incoming data, choose the `-timestamp` option when starting the `dsilog` process.

The simplest metric description, which uses the metric name as the label and the defaults of `AVERAGED`, `MAXIMUM 100`, and `PRECISION 3` decimal places, requires the following description:

```
METRICS
metric_name = metric_id_number
```

You must compile each class using `sdlcomp` and then start logging the data for that class using the `dsilog` process, regardless of whether you have reused metric names.

Example

```
VM = 11200;
```

VM is an example of a metric reusing a metric definition that has already been defined in another class in the same log file set.

LABEL

The metric label identifies the metric in graphs and exported data.

Syntax

`[LABEL "`*`metric_label_name`***`"]`**

How To Use It

Specify a text string, surrounded by double quotation marks, to label the metric in graphs and exported data. Up to 48 characters are allowed. If no label is specified, the metric name is used to identify the metric.

If the label contains a double-quotation mark, precede it with a backslash (\). For example, you would enter **`\"my\"`** data if the label is `"my"` data.

The metric label cannot be the same as any of the elements of the DSI class specification syntax such as `CAPACITY` or `ACTION`.

Example

```
METRICS
RUN_Q_PROCS    = 106
LABEL   "Procs in run q";
```

Summarization Method

The summarization method determines how to summarize data if the number of records exceed the number set in the RECORDS PER HOUR option of the CLASS section. For example, you would want to total a count of occurrences, but you would want to average a rate. A summarization method can apply only to numeric metrics.

Syntax

[TOTALED | AVERAGED | SUMMARIZED BY *metric_name*]

How To Use It

SUMMARIZED BY should be used when a metric is not being averaged over time, but over another metric in the class. For example, assume you have defined metrics TOTAL_ORDERS and LINES_PER_ORDER. If these metrics are given to the logging process every five minutes but records are being written only once each hour, to correctly summarize LINES_PER_ORDER to be (total lines / total orders), the logging process must perform the following calculation every five minutes:

- Multiply LINES_PER_ORDER * TOTAL_ORDERS at the end of each five-minute interval and maintain the result in an internal running count of total lines.
- Maintain the running count of TOTAL_ORDERS.
- At the end of the hour, divide total lines by TOTAL_ORDERS.

To specify this kind of calculation, you would specify LINES_PER_ORDER as SUMMARIZED BY TOTAL_ORDERS.

If no summarization method is specified, the metric defaults to AVERAGED.

Example

```
METRICS
ITEM_1_3 = 11203
LABEL  "TOTAL_ORDERS"
TOTALED;
ITEM_1_5  = 11205
LABEL  "LINES_PER_ORDER"
SUMMARIZED BY ITEM_1_3;
```

MAXIMUM

The metric maximum value identifies how large the number might be. It is valid only for numeric metrics and is meant to be used for estimating a maximum value range for graphing the metric in Performance Manager.

Syntax

[MAXIMUM *metric_maximum_number***]**

How To Use It

Specify the expected maximum value for any metric. This value does not specify the largest acceptable number for logged data. (See the table in the following section for the largest acceptable numbers according to precision settings.)

The MAXIMUM setting is primarily used to estimate graphing ranges in the analysis software about the initial size of a graph containing the metric and to determine a precision if PRECISION is not specified. The default is 100. Zero is always used as the minimum value because the kinds of numbers expected to be logged are counts, average counts, rates, and percentages. See the next section, PRECISION, for maximum values that can be logged.

Example

```
METRICS
RUN_Q_PROCS    = 106
LABEL   "Procs in run q"
MAXIMUM 50;
```

PRECISION

PRECISION identifies the number of decimal places to be used for metric values. If PRECISION is not specified, it is calculated based on the MAXIMUM specified. If neither is specified, the default PRECISION value is 3. This setting is valid only for numeric metrics.

Syntax

[PRECISION {0|1|2|3|4|5}]

How To Use It

The PRECISION setting determines the largest value that can be logged. Use **PRECISION 0** for whole numbers.

Table 3 Precision Setting Values

PRECISION	Number of decimal places	Largest acceptable number	MAXIMUM
0	0	2,147,483,647	> 10,000
1	1	214,748,364.7	1001 to 10,000
2	2	21,474,836.47	101 to 1,000
3	3	2,147,483.647	11 to 100
4	4	214,748.3647	2 to 10
5	5	21,474.83647	1

Example

```
METRICS
RUN_Q_PROCS    = 106
LABEL   "Procs in run q"
PRECISION 1;
```

TYPE TEXT LENGTH

The three keywords `TYPE TEXT LENGTH` specify that a metric is textual rather than numeric. This setting determines the number of characters allowed for the metric. Text is defined as any character other than `^z, \n`, or the separator, if any.

Because the default delimiter between data items for `dsilog` input is a blank space, you will need to change the delimiter if the text contains embedded spaces. Use the `dsilog -c char` option to specify a different separator as described in Chapter 4.

Syntax

`[TYPE TEXT LENGTH` *number*`]`

How To Use It

The LENGTH must be greater than zero and less than 256.

Summarization method, `MAXIMUM`, and `PRECISION` cannot be specified with text metrics. Because text cannot be summarized, which means that `dsilog` will take the first logged value in an interval and ignore the rest.

Example

```
METRICS
text_1 = 16
LABEL "first text metric"
TYPE TEXT LENGTH 20;
```

Sample Class Specification

```
CLASS SYS_STATS = 10001
LABEL "STATUS data"
INDEX BY HOUR
MAX INDEXES 12
ROLL BY HOUR
RECORDS PER HOUR 120;


METRICS

RUN_Q_PROCS    = 106
LABEL   "Procs in run q"
PRECISION 0;

BLOCKED_PROCS   = 107
LABEL   "Blocked Processes"
PRECISION 0;

SWAPPED_PROCS   = 108
LABEL   "Swapped Processes"
PRECISION 0;

AVG_VIRT_PAGES  = 201
LABEL   "Avg Virt Mem Pages"
PRECISION 0;

FREE_LIST_SIZE  = 202
LABEL   "Mem Free List Size"
PRECISION 0;

PAGE_RECLAIMS   = 303
LABEL   "Page Reclaims"
PRECISION 0;


ADDR_TRANS_FAULTS = 304
LABEL   "Addr Trans Faults"
PRECISION 0;

PAGES_PAGED_IN  = 305
LABEL   "Pages Paged In"
PRECISION 0;

PAGES_PAGED_OUT  = 306
LABEL   "Pages Paged Out"
PRECISION 0;

PAGES_FREED    = 307
LABEL   "Pages Freed per Sec"
PRECISION 0;

MEM_SHORTFALL   = 308
LABEL   "Exp Mem Shortfall"
PRECISION 0;

CLOCKED_PAGES   = 309
LABEL   "Pgs Scanned per Sec"
PRECISION 0;

DEVICE_INTERRUPTS = 401
LABEL   "Device Interrupts"
PRECISION 0;
```

```
SYSTEM_CALLS   = 402
LABEL   "System Calls"
PRECISION 0;

CONTEXT_SWITCHES = 403
LABEL   "Context Switches/Sec"
PRECISION 0;

USER_CPU     = 501
LABEL   "User CPU"
PRECISION 0;

SYSTEM_CPU    = 502
LABEL   "System CPU"
PRECISION 0;

IDLE_CPU     = 503
LABEL   "Idle CPU"
PRECISION 0;
```

4 DSI Program Reference

This chapter provides detailed reference information about:

- The `sdlcomp` compiler
- Configuration files
- DSI logging processes
- Exporting DSI data
- The `sdlutil` data source management utility

Sdlcomp Compiler

The DSI compiler, `sdlcomp`, checks the class specification file for errors. If it finds no errors, it adds the class and metric descriptions to the description file in the log file set you name. `Sdlcomp` also sets up the pointers in the log file set's root file to the log file to be used for data storage. If either the log file set or the log file does not exist, `sdlcomp` creates it.

You can put DSI log files anywhere on your system by specifying a full path in the compiler command. However, once the path has been specified, DSI log files *cannot* be moved to different directories. (SDL62 is the associated error.)

Compiler Syntax

```
sdlcomp [-maxclass value] specification_file logfile_set [log_ file]
[options]
```

Variable & Option	Definitions
-maxclass value	allows you to specify the maximum number of classes to be provided for when creating a new log file set. This option is ignored if it is used with the name of an existing log file set. Each additional class consumes about 500 bytes of disk space in overhead, whether the class is used or not. The default is 10 if -maxclass is not specified.
specification_file	is the name of the file that contains the class specification. If it is not in the current directory, it must be fully qualified.
logfile_set	is the name of the log file set this class should be added to. If the logfile_set does not exist, it will be created. If the logfile_set name is not fully qualified, it is assumed to be in the current directory. You can keep log file sets anywhere you choose. If no log file set is named, compilation errors are written to stderr and no log file is created. Compile without a log file set name first to check for compilation errors before actually creating the log file set. You can redirect stderr to a file for later viewing. Class and metric names and numeric IDs that have been previously used in the log file set will not cause compilation errors until you run sdlcomp with the log file set option.
log file	is the log file in the set that will contain the data for this class. If no log file is named, a new log file is created for the class and named according to class name. The default name is logfile_set_name.class_name.
-verbose	prints a detailed description of the compiler output to stdout.
-vers	displays version information.
-?	displays the syntax description.
-u	allows you to log more than one record per second. Use this option to log unsummarized data only.

Sample Compiler Output

Given the following command line:

```
->sdlcomp status.spec sdl_new
```

the following code is sample output for a successful compile. Note that status.spec is the sample specification file presented in the previous chapter.

```
sdlcomp
Check class specification syntax.

CLASS SYS_STATS = 10001
LABEL "STATUS data"
INDEX BY HOUR
MAX INDEXES 12
ROLL BY HOUR
RECORDS PER HOUR 120;
```

```
METRICS

RUN_Q_PROCS    = 106
LABEL   "Procs in run q"
PRECISION 0;

BLOCKED_PROCS   = 107
LABEL   "Blocked Processes"
PRECISION 0;

SWAPPED_PROCS   = 108
LABEL   "Swapped Processes"
PRECISION 0;

AVG_VIRT_PAGES  = 201
LABEL   "Avg Virt Mem Pages"
PRECISION 0;

FREE_LIST_SIZE  = 202
LABEL   "Mem Free List Size"
PRECISION 0;

PAGE_RECLAIMS   = 303
LABEL   "Page Reclaims"
PRECISION 0;

ADDR_TRANS_FAULTS = 304
LABEL   "Addr Trans Faults"
PRECISION 0;

PAGES_PAGED_IN  = 305
LABEL   "Pages Paged In"
PRECISION 0;

PAGES_PAGED_OUT  = 306
LABEL   "Pages Paged Out"
PRECISION 0;

PAGES_FREED    = 307
LABEL   "Pages Freed per Sec"
PRECISION 0;

MEM_SHORTFALL   = 308
LABEL   "Exp Mem Shortfall"
PRECISION 0;

CLOCKED_PAGES   = 309
LABEL   "Pgs Scanned per Sec"
PRECISION 0;

DEVICE_INTERRUPTS = 401
LABEL   "Device Interrupts"
PRECISION 0;

SYSTEM_CALLS   = 402
LABEL   "System Calls"
PRECISION 0;

CONTEXT_SWITCHES = 403
LABEL   "Context Switches/Sec"
PRECISION 0;

USER_CPU     = 501
LABEL   "User CPU"
PRECISION 0;
```

```
SYSTEM_CPU    = 502
LABEL   "System CPU"
PRECISION 0;

IDLE_CPU     = 503
LABEL   "Idle CPU"
PRECISION 0;
Note: Time stamp inserted as first metric by default.

  Syntax check successful.

Update SDL sdl_new.
Open SDL sdl_new
Add class SYS_STATS.
Check class SYS_STATS.

Class SYS_STATS successfully added to logfile set.
```

For explanations of error messages, see Chapter 6, Error Messages.

Configuration Files

Before you start logging data, you may need to update the following three Measure Ware Agent configuration files:

- `datasources - to` configure a DSI log file as a data source.
- `alarmdef.mwc - to` define alarm conditions on DSI metrics.
- `dsiconf.mwc` – to define continuous logging of DSI data.

These files are located in the `\rpmtools\data\` directory, unless you specified them to be in a different directory during installation.

See "Configuring Data Sources" in the *HP Performance Agentt Installation and Configuration Guide for Windows* for detailed information about using and updating the datasources configuration file.

➤ Make sure that the manual name & chapter/section name match with the Windows installation & configuration guide.

➤ Stopping the repository server processes results in any current connection in Performance Manager being lost. For example, if you are drawing a graph on a data source and try to draw another graph, you will need to reselect the data source in Performance Manager and re-establish the connection once the repository server is started again.

Examine the contents of the `status.rep_server` file to check if the repository was activated or for error messages.

Defining Alarms on DSI Metrics

You can use Performance Agent to define alarms on DSI metrics. These alarms notify you when DSI metrics meet conditions that you have defined. To define alarms, you specify conditions that, when met or exceeded, trigger an alert notification or action. You define alarms for data logged through DSI the same way as for other Performance Agent metrics — in the `alarmdef.mwc` file on the Performance Agent system. The `alarmdef.mwc` file is located in the `\rpmtools\data\` directory unless you specified a different directory during installation).

Whenever you specify a DSI metric name, it should be fully qualified; that is, preceded by the *datasource_name*, and the *class_name* as shown below:

`datasource_name:class_name:metric_name`

- **`datasource_name`** is the name you have used to configure the data source in the `datasources` file. See Configuration Files on page 47 for more information.
- **`class_name`** is the name you have used to identify the class in the class specification for the data source. You do not need to enter the **`class_name`** if the metric name is unique (not reused) in the class specification.
- **`metric_name`** is the data item from the class specification for the data source.

If you choose *not* to fully qualify a metric name, you need to include the USE statement in the `alarmdef.mwc` file to identify which data source to use. For more information about the USE statement, see Chapter 8 in the *MeasureWare Agent for Windows NT/2000: User's Manual.*

Before proceeding with another task, you *must* activate any changes you made to the `alarmdef.mwc` file.

1. Choose **Start/Stop** from the Agent menu on the Performance Agent main window to open the MeasureWare Services window.
2. Select the **Alarm Definitions** check box.
3. Click **Refresh**.
4. Click **Close** to return to the Performance Agent main window..

For details on the alarm definition syntax, how alarms are processed, and customizing alarm definitions, see Chapter 8 of the *MeasureWare Agent for Windows NT/2000: User's Manual*.

Alarm Processing

As data is logged by `dsilog` it is compared to the alarm definitions in the `alarmdef.mwc` file to determine if a condition is met or exceeded. When this occurs, an alert notification or action is triggered.

You can configure where you want alarm notifications sent and whether you want local actions performed. Alarm notifications can be sent to the central Performance Manager analysis system where you can draw graphs of metrics that characterize your system performance. SNMP traps can be sent to HP Network Node Manager. Local actions can be performed on the Performance Agent system. Alarm information can also be sent to Operations Manager.

For information about configuring where you want alarm notifications sent, see Chapter 8 of the *MeasureWare Agent for Windows NT/2000: User's Manual*.

Configuring Continuous Logging of DSI Data

The DSI service allows DSI to continuously log data for every data collection you have started with DSI, even after you have logged off your system. You can set the service to automatically start with Performance Agent by selecting **Services** in the Windows NT Control Panel (the Services applet is under Administrative Tools on Windows 2000) and choosing the **DSI service**.

Before you start using the DSI service, you must configure the DSI configuration file, `dsiconf.mwc`, to define which DSI data is to be logged. The `dsiconf.mwc` file is located in your `\rpmtools\data\` directory unless you specified a different directory during installation.

When you first open the `dsiconf.mwc` file, you see a series of settings and comments that help you supply all necessary information. Each line is preceded by the pound sign (#), which you remove when you supply the values for each entry.

The format for defining a continuous DSI logging process is:

```
DATAFEED=ping -t nnn.nnn.nnn.nnn

LOGFILE=c:\rpmtools\data\datafiles\PINGLOG

CLASS=SYS_RESPONSE

DSIPARMS=-f c:\rpmtools\data\datafiles\ping.fmt

;
```

Each DSI logging process you configure to run under the DSI service must have a `DATAFEED`, `LOGFILE`, and `CLASS` entry. The `DSIPARMS` entry is optional.

For each DATAFEED, LOGFILE, and CLASS entry, substitute a value within the < > brackets.

Keywords such as DATAFEED, LOGFILE, CLASS, and DSIPARMS file must be in uppercase.

Before proceeding with another task, you *must* activate any changes you made to the dsiconf.mwc file.

1. Choose **Start/Stop Services** from the Agent menu of the Performance Agent main window to display the MeasureWare Services window.
2. Select the **Persistent DSI Collections** check box.
3. Click **Refresh**.
4. Click **Close** to return to the Performance Agent main window.

You can also start and stop the DSI service and refresh (activate) the dsiconf.mwc file using the following commands in the Windows NT/2000 Command Prompt:

- To start the DSI service, type:

  OVPACMD START DSI

- To stop the DSI service, type:

  OVPACMD STOP DSI

- To activate changes in the dsiconf.mwc file, type:

  OVPACMD REFRESH DSI

DSI Logging Processes

DSI logging requires that you either devise your own program or use one already in existence (such as `ping`) for gaining access to data. You can then pipe this data into `dsilog`, which logs the data into the log file set. A separate logging process must be implemented for each class you define.

`Dsilog` expects to receive data from `stdin`. To start the logging process, you could pipe the output from the program you are using to collect data to `dsilog`, as shown in the example below. A complete example of piping `ping` data into DSI is given in Chapter 5.

```
ping <system name>| dsilog logfile_set class
```

Note that if stream I/O C runtime functions such as `fprintf( )` are used to produce output in the program you are using to collect data, the piping will not be continuous due to buffering of the data prior to piping. Although no data will be lost, it will not be processed in real time and could therefore make the data more difficult to interpret. For example, if the last portion of data that triggers an alarm is delayed, the alarm will not be processed (or received) until sometime after the actual alarm condition occurred. For this reason, turn off buffering if you use stream I/O functions or use Win32 functions when creating applications to be monitored with DSI.

An example of turning buffering off in the data stream to be piped to `dsilog` is given in the example program, `eschgdsi.c`, in Chapter 5.

Note that `dsilog` will not accept Unicode input.

Syntax

`dsilog logfile_set class [`*options*`]`

The `dsilog` parameters and options are described on the following pages.

Table 1 DSI Logging Parameters and Options

Variables & Options	Definitions
`logfile_set`	names the log file set where the data is to be stored. If it is not in the current directory, the name must be fully qualified.
`class`	names the class to be logged.
`-asyn`	specifies that the data will arrive asynchronously with the `RECORDS PER HOUR` rate. If no data arrives during a logging interval, the data for the last logging interval is repeated. However, if `dsilog` has logged no data yet, the metric value logged is treated as missing data. This causes a flat line to be drawn in a graphical display of the data and causes data to be repeated in each record if the data is exported.
`-c char`	uses the specified character as a string delimiter/separator. If embedded spaces occur in any text metrics, you must specify a unique separator using this option. You may not use the following characters as separators: decimal, minus sign, ^z, \n.

Table 1 DSI Logging Parameters and Options

Variables & Options	Definitions
`-f format_file`	names a file that describes the data that will be input to the logging process. If this option is not specified, `dsilog` derives the format of the input from the class specification with the following assumptions. See Creating a Format File on page 54 later in this chapter for more information. • Each data item in an input record corresponds to a metric that has been defined in the class specification. • The metrics are defined in the class specification in the order in which they appear as data items in the input record. • If there are more data items in an input record than there are metric definitions, `dsilog` ignores all additional data items. • If the class specification lists more metric definitions than there are input data items, the field will show “missing” data when the data is exported, and no data will be available for that metric when graphing data in the analysis software. The number of fields in the format file is limited to 100.
`-i input file`	indicates that the input should come from the file named. Although the most common use of dsilog is to receive live data directly, it can also read data from a static text file.
`-s seconds`	is the number of seconds by which to summarize the data. Zero turns off summarization, which means that all incoming data is logged. If this option is omitted, the summarization rate defaults to the `RECORDS PER HOUR` rate in the class specification. If present, this option overrides the value of `RECORDS PER HOUR`. A zero (0) turns off summarization, which means that all incoming data is logged. Caution should be used with the `-s 0` option because dsilog will timestamp the log ata at the time the point arrived. This can cause problems for Performance Manager and `perfalarm`, which work best with timestamps at regular intervals. If the log file will be accessed by Performance Manager or the OV Performance perfalarm program, use of the `-s 0` option is discouradged.
`-t`	prints everything that is logged to stdout in ASCII format.
`-timestamp`	indicates that the logging process should not provide the timestamp, but use the one already provided in the input data. The timestamp in the incoming data must be in UNIX timestamp format (seconds since 1/1/70 00:00:00) and represent the local time.
`-vi`	filters the input through `dsilog` and writes errors to `stdout` instead of the log file. It does not write the actual data logged to stdout (see the `-vo` option below). This can be used to check the validity of the input.

Table 1 DSI Logging Parameters and Options

Variables & Options	Definitions
`-vo`	filters the input through `dsilog` and writes the actual data logged and errors to `stdout` instead of the log file. This can be used to check the validity of the data summarization.
`-vers`	displays version information.
`-?`	displays the syntax description.

How Dsilog Processes Data

The `dsilog` program scans each input data string, parsing delimited fields into individual numeric or text metrics.

A key rule for predicting how the data will be processed is the validity of the input string. A valid input string requires that a delimiter be present between any specified metric types (numeric or text). A blank space is the default delimiter, but a different delimiter can be specified with the `dsilog -c char` command line option.

You *must* include a new line character at the end of any record fed to DSI in order for DSI to interpret it correctly.

Testing the Logging Process with Sdlgendata

Before you begin logging data, you can test the compiled log file set and the logging process using the `sdlgendata` program that is included with Performance Agent. `Sdlgendata` discovers metrics for a class (as described in the class specification) and creates data for each metric in a class.

Syntax

`sdlgendata logfile_set class` [*options*]

`Sdlgendata` parameters and options are explained below.

Variable & Options	Definitions
`logfile_set`	is the name of the log file set to generate data for
`class`	is the data class to generate data for.
`-timestamp` [*number*]	provides a timestamp with the data. If a negative number or no number is supplied, the current time is used for the timestamp. If a positive number is used, the time starts at 0 and is incremented by number for each new data record.
`-wait` *number*	causes a wait of number seconds between records generated.
`-cycle number`	recycles data after number cycles.
`-vers`	displays version information.
`-?`	displays the syntax description.

By piping `sdlgendata` output to the `dsilog` process with either the `-vi` or `-vo` options, you can verify the input (`vi`) and verify the output (`vo`) before you begin logging with your own process or program.

After you are finished testing, delete all log files created from the test. Otherwise, these files remain as part of the log file set

Use the following command to pipe data from `sdlgendata` to the logging process. Data and errors are written to `stdout`. Press **Ctrl+C** or other interrupt control character to stop data generation.

```
sdlgendata logfile_set class -wait 5 | dsilog logfile_set class -s 10
-vi
```

The previous command displays data created by `sdlgendata` and looks like this:

```
dsilog
I: 1.0000 2.0000 3.0000  4.0000  5.0000  6.0000  7.0000
I: 2.0000 3.0000 4.0000  5.0000  6.0000  7.0000  8.0000
I: 3.0000 4.0000 5.0000  6.0000  7.0000  8.0000  9.0000
I: 4.0000 5.0000 6.0000  7.0000  8.0000  9.0000  10.0000
I: 5.0000 6.0000 7.0000  8.0000  9.0000  10.0000  11.0000
I: 6.0000 7.0000 8.0000  9.0000  10.0000 11.0000  12.0000
I: 7.0000 8.0000 9.0000  10.000  11.000  12.0000  13.0000
I: 8.0000 9.0000 10.0000 11.0000 12.0000 13.0000  14.0000
```

You can also use the `-vo` option of `dsilog` to examine input and summarized output for your real data without actually logging it. The following command outputs `ping` data every second until interrupted, then pipes it to `dsilog` where it is summarized to five seconds. See Chapter 5, Examples of Data Source Integration for a complete example of piping `ping` data into DSI.

```
ping -t system name | dsilog logfile_set class -f ping.fmt -s 5 -vo
```

Note that a format file was used for directing how the final output should appear. This format file (`ping.fmt`) deletes specified columns so that in this case only the time-stamp, the IP-address, and the summarized output appear next to the "L:" after the "interval marker" line.

Sample output from this example `dsilog -vo` command is shown below:

```
I: 866017245 Reply-44109.2813 from-44110.3984 15.8.155.224:-44111.5195 32
10
I: 866017246 Reply-44109.2813 from-44110.3984 15.8.155.224:-44111.5195 32
12
I: 866017247 Reply-44109.2813 from-44110.3984 15.8.155.224:-44111.5195 32
15
I: 866017248 Reply-44109.2813 from-44110.3984 15.8.155.224:-44111.5195 32
18
I: 866017249 Reply-44109.2813 from-44110.3984 15.8.155.224:-44111.5195 32
20
interval marker
L: 866017245 15.8.155.224:     15
```

```
I: 866017250 Reply-44109.2813 from-44110.3984 15.8.155.224:-44111.5195 32
10

I: 866017251 Reply-44109.2813 from-44110.3984 15.8.155.224:-44111.5195 32
10

I: 866017252 Reply-44109.2813 from-44110.3984 15.8.155.224:-44111.5195 32
18

I: 866017253 Reply-44109.2813 from-44110.3984 15.8.155.224:-44111.5195 32
18

I: 866017254 Reply-44109.2813 from-44110.3984 15.8.155.224:-44111.5195 32
14

interval marker

L: 866017250 15.8.155.224:     14
```

You can also use the `dsilog -vo` option to use a file of old data for testing, as long as the data contains its own UNIX timestamp (seconds since 1/1/70 00:00:00). To use a file of old data, enter a command like this:

```
dsilog -timestamp -vo < oldfile >
```

Creating a Format File

Create a format file to map the data input to the class specification if:

- the data input contains data that is not included in the class specification.
- incoming data has metrics in a different order than you have specified in the class specification.

A format file is an ASCII text file that you can create with Notepad or any text editor. Use the `-f` option in `dsilog` to specify the fully qualified name of the format file.

Because the logging process works by searching for the first valid character after a delimiter (either a space by default or user-defined with the `-c` option to the `dsilog` process) to start the next metric, the format file simply tells the logging process which fields to skip and which metric names to associate with fields not skipped.

`$numeric` tells the logging process to skip one numeric metric field and go to the next. `$any` tells the logging process to skip one text metric field and go to the next. Note that the format file is limited to 100 fields.

For example, if the incoming data stream contains this information:

```
ABC 987 654 123 456
```

and you want to log only the first numeric field into a metric named `metric_1`, the format file would look like this:

```
$any metric_1
```

This tells the logging process to log only the information in the first numeric field and discard the rest of the data. To log only the information in the third numeric field, the format file would look like this:

```
$any $numeric $numeric metric_1
```

To log all four numeric data items, in reverse order, the format file would look like this:

```
$any metric_4 metric_3 metric_2 metric_1
```

If the incoming data stream contains the following information:

```
/users  15.9  3295  56.79%  xdisk1  /dev/dsk/c0d0s*
```

and you want to log only the first text metric and the first two numeric fields into metric fields you name text_1, num_1, and num_2, respectively, the format file would look like this:

```
text_1  num_1  num_2
```

This tells the logging process to log only the information in the first three fields and discard the rest of the data.

To log all of the data, but discard the "%" following the third metric, the format file would look like this:

```
text_1  num_1  num_2  num_3  $any  text_2  text_3
```

Because you are logging text fields and the "%" is considered to be a text field, you need to skip it to correctly log the text field that follows it.

To log the data items in a different order the format file would look like this:

```
text_3  num_2  num_1  num_3  $any  text_2  text_1
```

Note that this will result in only the first six characters of text_3 being logged if text_1 is declared to be six characters long in the class specification. To log all of text_3 as the first value, change the class specification and alter the data stream to allow extra space.

Another example of a format file is given in the DSI `ping` example in Chapter 5.

Changing a Class Specification

To change a class specification file, you must recreate the whole log file set as follows:

1. Stop the `dsilog` process.
2. Export the data from the existing log file using the UNIX time stamp option if you want to save it or integrate the old data with the new data you will be logging. For more information, see the next section, Exporting DSI Data .
3. Run `sdlutil` to remove the log file set. For more information, see Managing Data With Sdlutil on page 58 later in this chapter.
4. Update the class specification file.
5. Run `sdlcomp` to recompile the class specification.
6. Optionally, use the `-l` option in `dsilog` to integrate in the old data you exported in step 2.. You may need to manipulate the data to line up with the new data using the `-f format_file` option.
7. Run `dsilog` to start logging based on the new class specification.
8. Set up the DSI service to continue to log data, even after you have logged off your system. For more information, see Configuring Continuous Logging of DSI Data on page 48 earlier in this chapter. (You can start or stop all Performance Agent services from the Agent menu in the Performance Agent main window.)

As long as you have not changed the log file set name or location, you do not need to update the `datasources` file.

Exporting DSI Data

You can export data from a DSI log file using the **export** command from the Logfiles menu on the Performance Agent main window. For detailed instructions for exporting data, see online Help or the "Exporting Log File Data" section in Chapter 3 of the *MeasureWare Agent for Windows NT/2000: User's Manual*.

You can also export DSI log file data with the **export** command in Performance Agent's `extract` program in the Windows NT/2000 Command Prompt. For more information, see the `export` command description in `extract` online Help or the "Overview of the Export Function" section in Chapter 6 and the `export` command description in Chapter 7 of your *MeasureWare Agent for Windows NT/2000: User's Manual*.

There are several ways to find out which classes and metrics can be exported from a DSI log file. You can use `sdlutil` to list this information as described in Managing Data With Sdlutil on page 58 later in this chapter. Or you can use guided mode in Performance Agent's `extract` program in the Windows NT/2000 Command Prompt. Guided mode assists you in creating an export template file that lists the classes and metrics in the DSI log file. You can then use Notepad or WordPad to edit, name, and save the file. For more information, see the `guide` command description in `extract` online Help or the `guide` command description in Chapter 7 of your *MeasureWare Agent for Windows NT/2000: User's Manual*.

Viewing Data in Performance Manager

In order to display data from a DSI log file in Performance Manager, you need to configure the DSI log file as a Performance Agent data source. To configure the data source, add it to the `datasources` file on the Performance Agent system. See Configuration Files on page 47 for more information.

You can centrally view, monitor, analyze, compare, and forecast trends in DSI data using Performance Manager. Performance Manager helps you automatically identify current and potential problems. It provides the information you need to resolve problems *before* user productivity is affected.

For information about using Performance Manager, see Performance Manager online Help.

Managing Data With Sdlutil

To manage the data from a DSI log file, use the `sdlutil` program to do the following:

- List currently defined class and metric information.
- List complete statistics for classes.
- Show metric descriptions for all metrics listed.
- List the files in a log file set.
- Remove classes and data from a log file set.
- Recreate a class specification from the information in the log file set.
- Display version information.

Syntax

```
sdlutil logfile_set [options]
```

Table 2 Sdlutil Parameters and Options

Variables & Options	Definitions
`logfile_set`	is the name of the log file set created by compiling a class specification.
`-classes` *classlist*	provides a class desccription of all classes listed. If none are listed, all are provided. Separate the items in the classlist with spaces.
`-state` *classlis*t	provides complete statistics for all classes listed. If none are listed, all are provided. Separate the items in the classlist with spaces.
`-metrics` *metricslist*	provides metric descriptions for all metrics in the log file set. If none are listed, all are provided. Separate the items in the metricslist with spaces.
`-files`	lists all the files in the log file set.
`-rm all`	removes all classes and data as well as their data and shared memory ID from the log file.
`-decomp` *classlist*	recreates a class specification from the information in the log file set. The results are written to `stdout` and should be redirected to a file if you plan to make changes to the file and reuse it. Separate the items in the classlist with spaces.
`-vers`	displays version information.
`-?`	displays the syntax description.

5 Examples of Data Source Integration

DSI Examples

Data source integration is a powerful and flexible technology. Implementation of DSI can range from simple and straightforward to very complex. This chapter contains examples of using DSI with the following tasks:

- Monitoring Microsoft Exchange performance data
- Monitoring network response time using the `ping` command

Monitoring Microsoft Exchange Data

This example shows how to pull Microsoft Exchange performance data into DSI using a custom feeder program. You could also use the Extended Collection Builder for a simpler method of obtaining the data, but this example is meant to show you how you can access application data. This feeder program accesses the Windows NT/2000 system registry and can be modified to pull other data from the registry. The C source code for the program `exchgdsi` is shown below:

```
// THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
// EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES
// OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.
// Copyright 1997 Hewlett-Packard Corporation. All Rights Reserved.
// PROGRAM:  exchgdsi.c
// PURPOSE:  Harvest several Microsoft Exchange performance metrics and output // them in a
format suitable for input to Dsilog.
// PLATFORMS: Windows NT 4.0, Windows 2000
#ifdef UNICODE
#ifndef _UNICODE
#define _UNICODE 1
#endif
#define tmain  wmain
#else
#define tmain  main
#endif

#define WIN32_LEAN_AND_MEAN 1

#include <windows.h>
#include <winperf.h>
#include <malloc.h>
#include <stdio.h>
#include <conio.h>
#include <tchar.h>
#include <pdh.h>

#define DELIVERED_RATE_PRIVATE _T("\\MSExchangeIS Private\\Messages Delivered/min")
#define SENT_RATE_PRIVATE   _T("\\MSExchangeIS Private\\Messages Sent/min")
#define SUBMITTED_RATE_PRIVATE _T("\\MSExchangeIS Private\\Messages Submitted/min")
#define DELIVERED_RATE_PUBLIC _T("\\MSExchangeIS Public\\Messages Delivered/min")
#define SENT_RATE_PUBLIC    _T("\\MSExchangeIS Public\\Messages Sent/min")
continued
#define SUBMITTED_RATE_PUBLIC _T("\\MSExchangeIS Public\\Messages Submitted/min")
#define NUM_COUNTERS     6

typedef struct _COUNTER_INFO
{
  HCOUNTER       hCounter;
  PDH_FMT_COUNTERVALUE Value;
} COUNTER_INFO, *PCOUNTER_INFO;

int tmain()
{
  PCOUNTER_INFO pCounterInfo;
  HQUERY    hQuery;
  int      i;

  // this must be done before using the output stream
  // to prevent buffering of the output
  setvbuf( stdout, NULL, _IONBF, 0 );

  pCounterInfo = (PCOUNTER_INFO) malloc(NUM_COUNTERS * sizeof(COUNTER_INFO));
  PdhOpenQuery(NULL, 0, &hQuery);
  PdhAddCounter(hQuery, DELIVERED_RATE_PRIVATE, 0, &(pCounterInfo[0].hCounter));
```

```
  PdhAddCounter(hQuery, SENT_RATE_PRIVATE,    0, &(pCounterInfo[1].hCounter));
  PdhAddCounter(hQuery, SUBMITTED_RATE_PRIVATE, 0, &(pCounterInfo[2].hCounter));
  PdhAddCounter(hQuery, DELIVERED_RATE_PUBLIC, 0, &(pCounterInfo[3].hCounter));
  PdhAddCounter(hQuery, SENT_RATE_PUBLIC,     0, &(pCounterInfo[4].hCounter));
  PdhAddCounter(hQuery, SUBMITTED_RATE_PUBLIC, 0, &(pCounterInfo[5].hCounter));

  PdhCollectQueryData(hQuery);
  while ( !_kbhit() {
    Sleep(60000);
    PdhCollectQueryData(hQuery);
    for ( i = 0; i < NUM_COUNTERS; i++ {
      PdhGetFormattedCounterValue(pCounterInfo[i].hCounter,
                    PDH_FMT_LONG,
                    NULL,
                    &pCounterInfo[i].Value);
      _tprintf(_T("%ld "), pCounterInfo[i].Value.longValue);
    }
    _tprintf(_T("\n"));
  }
  _getch();
  PdhCloseQuery(hQuery);
  free(pCounterInfo);
  return 0;
}
```

This program uses the Performance Data Helper interface that is part of the Win32 SDK.

Creating a Class Specification File

The class specification file is an ASCII text file that you create to describe the "class" or set of incoming data, as well as each individual number you intend to log as a metric within the class. The file can be created with Notepad or any text editor or word processor that can save the file as ASCII text.

The following example is a class specification file (named `exchange.spec`) for use with the `exchgdsi` program shown in the previous section. The output of `exchgdsi` is the numeric values of six selected Microsoft Exchange performance counters, followed by a carriage return.

```
CLASS EXCHANGE_STATS = 10000
LABEL "MS Exchange Counters"
INDEX BY HOUR
MAX INDEXES 168
ROLL BY DAY
RECORDS PER HOUR 12
;

METRICS

Msgs_Delivered_Priv = 100
LABEL "Delivered/Min - priv"
PRECISION 0
;

Msgs_Sent_Priv    = 101
LABEL "Sent/Min - priv"
PRECISION 0
;

Msgs_Submitted_Priv = 102
LABEL "Submitted/Min - priv"
PRECISION 0
;

Msgs_Delivered_Pub  = 200
```

```
LABEL "Delivered/Min - pub"
PRECISION 0
;

Msgs_Sent_Pub    = 201
LABEL "Sent/Min - pub"
PRECISION 0
;

Msgs_Submitted_Pub  = 202
LABEL "Submitted/Min - pub"
PRECISION 0;
```

Compiling the Class Specification File

After creating the DSI class specification file as described in the previous section, you need to compile the class specification file to create a set of log files to hold the data for the class. If syntax errors are detected, they are reported and the log file set is *not* created.

Use the file name you gave the class specification file in the previous step, and choose a name for logfile_set that makes it easy to remember what kind of data the log file contains. In the following example of command and compiler output, `exchange.spec` is the class specification file name and `EXCHLOG` is the log file set to be created.

```
-> sdlcomp exchange.spec EXCHLOG


sdlcomp

Check class specification syntax.


CLASS EXCHANGE_STATS = 10000;


METRICS


Msgs_Delivered_Priv = 100;

Msgs_Sent_Priv      = 101;

Msgs_Submitted_Priv = 102;

Msgs_Delivered_Pub  = 200;

Msgs_Sent_Pub       = 201;

Msgs_Submitted_Pub  = 202;


NOTE: Time stamp inserted as first metric by default.


Syntax check successful.


Update SDL EXCHLOG.
```

```
Class EXCHANGE_STATS successfully added to logfile set.
```

This example creates a log file set called EXCHLOG in the current directory. The log file set includes a root file and description file in addition to the data file.

Creating a Format File

No format file is required for this example because the output of the exchgdsi program is already in the format expected by the DSI log file set created for this example.

Starting the DSI Logging Process

You can now begin logging data with the dsilog process by typing the following command at the Windows NT/2000 Command Prompt:

```
exchgdsi | dsilog EXCHLOG EXCHANGE_STATS
```

This command runs the example feeder program exchgdsi and pipes the output into dsilog. Dsilog then logs the Microsoft Exchange performance counters into the EXCHANGE_STATS class in the EXCHLOG log file set. The exchgdsi program delivers data for the six Microsoft Exchange performance counters every minute. dsilog summarizes these values and logs the data once every five minutes.

To collect data continuously, even after you have logged off, you need to configure the dsiconf.mwc file that is read by the DSI service. For example, the above feeder program, exchgdsi, would be included in this file. For details on how to configure the file, see Configuring the DSI Service.

Accessing the Data

You can use the sdlutil program to report on the contents of the class by typing the following command at the Windows NT/2000 Command Prompt:

```
sdlutil EXCHLOG -stats EXCHANGE_STATS
```

Exporting the Data

The data in the DSI log file EXCHLOG could be exported using the Performance Agent main window.

Briefly, here is how you would do the export:

1. Select **Export** from the Logfile menu in the Performance Agent main window.
2. Click the **Select Logfile** button and locate EXCHLOG, which, by default, is in the \rpmtools\data\datafiles directory.
3. Click **Open**.
4. Click the **Make Quick Template** button and choose the metrics to include in the template. (If you had already configured a template, could use it.)
5. Click the **Export Data** button.

A window appears that shows information about the exported data , including a list of the exported records and the amount of space the data occupies in the file.

Exporting Data From the Command Prompt

You also can use the following command line arguments in the Windows NT/2000 Command Prompt window to export data from the class.

```
extract -xp -l EXCHLOG -C EXCHANGE_STATS DETAIL -f output.txt
```

Monitoring Network Response Time

This example uses the `ping` command to show how to pipe data into DSI. The `ping` command sends a packet to a remote host on the network and requests that the remote host send a packet back. The output of the `ping` command is the round-trip time of the packet traveling between the two hosts as shown in the following example:

```
Reply from 15.8.155.224: bytes=32 time<10ms TTL=128
```

As this example shows, 10 milliseconds is the smallest round-trip time reported by `ping`.

Creating a Class Specification File

The class specification file is an ASCII text file that you create to describe the class, or set of incoming data, as well as each individual number to be logged as a metric within the class. The file can be created with Notepad or any text editor or word processor that allows you to save the file as a text file.

The following example is a class specification file (named `ping.spec`) for use with the `ping` command described above. The class specification file specifies 2 metrics—the IP address of the system that was pinged and the network response time.

```
CLASS SYS_RESPONSE = 20000

LABEL "System Response Data"

INDEX BY     HOUR

MAX INDEXES    12

RECORDS PER HOUR 60

ROLL BY      HOUR

;

METRICS


System_IP = 1001

LABEL "System IP"

TYPE TEXT LENGTH 20

;

Response_Time = 1002

LABEL "Response Time"

AVERAGED

PRECISION 0;
```

Compiling the Class Specification File

After creating the DSI class specification file, compile it with `sdlcomp` to create a set of log files to hold the data for the class. If syntax errors are detected, they are reported and the log file set is *not* created.

Use the file name you gave the class specification file in the previous step, and specify a name for logfile_set that makes it easy to remember what kind of data the log file contains. In the example of command and compiler output below, `ping.spec` is the class specification file name and `PINGLOG` is the log file set to be created.

```
-> sdlcomp ping.spec PINGLOG


sdlcomp
Check class specification syntax.


CLASS SYS_RESPONSE = 20000;


METRICS


System_IP    = 1001;
Response_Time  = 1002;


NOTE: Time stamp inserted as first metric by default.


Syntax check successful.


Update SDL PINGLOG.


Class SYS_RESPONSE successfully added to logfile set.
```

This example created the log file set called `PINGLOG` in the current directory. The log file set includes a root file and description file in addition to the data file.

Creating a Format File

The `ping` example requires a format file to work properly because the output of `ping` (shown below) includes more than just the two numeric values specified in the DSI class specification above.

```
Reply from 15.8.155.224: bytes=32 time<10ms TTL=128
```

Using the default input separator of a space, the output of `ping` would be seen as six items by DSI. The following format file, called `ping.fmt`, allows DSI to collect just the IP address and the response time from the `ping` output stream:

```
$any $any System_IP $numeric Response_Time
```

The format file works as follows on the output from `ping`:

Reply	from	15.8.155.224:	bytes=32	time<10ms TTL=128
$any	$any	System_IP	numeric	Response_Time
(skip)	(skip)	(log metric)	(skip)	log metric)

Note that when the input for a numeric field is a mixture of alpha characters and numbers (bytes=32 and time<10ms in the above example), the alpha characters are automatically ignored. The last item, TTL=128, is also ignored because `dsilog` will not look any further in the input stream once the two expected metrics (`System_IP` and `Response_Time`) have been collected.

A DSI format file is an ASCII text file that you can create with Notepad or any text editor or word processor that saves the file as a text file. The use of a format file is specified with the `-f` option to the `dsilog` command.

For more information, see Creating a Format File on page 54.

Starting the DSI Logging Process

You can now begin logging data with the `dsilog` process using the following command:

```
ping -t <system name> | dsilog PINGLOG SYS_RESPONSE -f ping.fmt
```

This command runs `ping` continuously (`-t` option) and pipes the output into `dsilog`, using the format file `ping.fmt`. The two metrics are logged into the `SYS_RESPONSE` class in the `PINGLOG` log file set. The `ping` command outputs data every second until interrupted. `Dsilog` summarizes these values and logs the data once per minute.

You can use either the host name or the IP address to identify the destination for the `ping` command. Use **Ctrl+C** to stop pinging.

Note that the following message will be generated at the start of the logging process:

```
Metric null has invalid data

Ignore to end of line, metric value exceeds maximum
```

This message is a result of the header line in the `ping` output that `dsilog` cannot log. Although the message appears on the screen, `dsilog` continues to run and begins logging data with the first valid input line.

Accessing the Data

You can use the `sdlutil` program to report on the contents of the class by typing:

```
sdlutil PINGLOG -stats SYS_RESPONSE
```

You can use the Performance Agent graphical interface to export data from the class. You can also export DSI data using command line arguments in the Windows NT/2000 Command Prompt as shown in the example below.

```
extract -xp -l PINGLOG -C SYS_RESPONSE DETAIL -f stdout
```

Here is an example of output that results from using these command line arguments.

```
DATE &                                Response
TIME                  System IP          Time
06/11/97 08:40:00 15.19.200.10:          611
06/11/97 08:41:00 15.19.200.10:          571
06/11/97 08:42:00 15.19.200.10:          481
06/11/97 08:43:00 15.19.200.10:          491
06/11/97 08:44:00 15.19.200.10:          210
06/11/97 08:45:00 15.19.200.10:          181
06/11/97 08:46:00 15.19.200.10:          190
06/11/97 08:47:00 15.19.200.10:          271
```

A value of 10 for the `Response_Time` metric means 10 milliseconds or less, since 10 milliseconds is the minimum value reported by `ping`.

Configuring Dsiconf.mwc for Collecting Ping Data

As a final task, you would add entries to the `dsiconf.mwc` file so that the DSI service would allow the continuous logging of DSI data, regardless of whether or not you were logged on.

To allow the DSI service to continuously collect data from the `ping` command, the `dsiconf.mwc` file would require the following entries:

```
DATAFEED=ping -t <system name>
LOGFILE=c:\hp openview\data\datafiles\PINGLOG
CLASS=SYS_RESPONSE
DSIPARMS=-f c:\rpmtools\data\ping.fmt
```

Note that the `DATAFEED` line requires the name of the pinged system and it is assumed that the correct paths are entered in the `LOGFILE` and `DSIPARMS` lines.

This file would be saved as `\hp openview\data\dsiconf.mwc`.

6 Error Messages

DSI Error Messages

DSI error messages fall into three groups: class specification, `dsilog` logging process, and general.

- Class specification error messages have the prefix `SDL` and the message number following the message text.
- The messages generated by the `dsilog` logging process have the prefix `DSILOG` and the message number following the message text.
- General error messages can be generated by either of the above as well as other tasks. These messages have a minus sign (–) prefix and the message number.

These error messages are listed in this chapter. SDL and DSILOG error messages are listed in numeric order, along with the actions you take to recover from the error condition. General error messages are self-explanatory so no recovery actions are given.

DSI messages regarding the DSI service are logged to the `status.dsi` file.

General Error Messages

Table 1 General Error Messages

Error	Explanation
-3	Attempt was made to add more classes than allowed by max-class.
-5	Could not open file containing class data.
-6	Could not read file.
-7	Could not write to file.
-9	Attempt was made to write to log file when write access was not requested.
-11	Could not find the pointer to the class.
-13	File or data structure not initialized.
-14	Class description file could not be read.
-15	Class description file could not be written to.
-16	Not all metrics needed to define a class were found in the metric description class.
-17	The path name of a file in the log file set is more than 1024 characters long.
-18	Class name is more than 20 characters long.
-19	File is not log file set root file.
-20	File is not part of a log file set.
-21	The current software cannot access the log file set.
-22	Could not get shared memory segment or id. Not applicable on Windows NT/ 2000.
-23	Could not attach to shared memory segment. Not applicable on Windows NT/ 2000.
-24	Unable to open log file set.
-25	Could not determine current working directory.
-26	Could not read class header from class data file.
-27	Open of file in log file set failed.
-28	Could not open data class.
-29	`lseek` failed.
-30	Could not read from log file.
-31	Could not write on log file.
-32	remove failed.

Table 1 General Error Messages

-33	`shmctl(REM_ID)` failed.
-34	Logfile set is incomplete; root or description file is missing.
-35	The target log file for adding a class is not in the current log file set.

SDL Error Messages

Message SDL1

```
ERROR: Expected equal sign, "=".
```

Probable cause: An "=" was expected here.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL2

```
ERROR: Expected semi-colon, ";".
```

Probable cause: A semi-colon (;) marks the end of the class specification and the end of each metric specification. You may also see this message if an incorrect or misspelled word is found where a semi-colon should have been.

For example, if you enter

```
class xxxxx = 10
   label "this is a test"
   metric 1000;


instead of
class xxxxx = 10
   label "this is a test"
   capacity 1000;
```

you would see this error message and it would point to the word "metric."

Corrective action*:* See Class Specification Syntax on page 25 for more information.

Message SDL3

```
ERROR: Precision must be one of {0, 1, 2, 3, 4, 5}.
```

Probable cause: Precision determines the number of decimal places used when converting numbers internally to integers and back to numeric representations of the metric value.

Corrective action: See PRECISION on page 38 for more information.

Message SDL4

```
ERROR: Expected quoted string.
```

Probable cause: A string of text was expected.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL5

```
ERROR: Unterminated string.
```

Probable cause: The string must end in double quotes.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL6

```
NOTE: Time stamp inserted as first metric by default.
```

Probable cause: A time stamp metric is automatically inserted as the first metric in each class.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL7

```
ERROR: Expected metric declarations.
```

Probable cause: The metrics section must start with the `METRICS` keyword before the first metric definition.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL8

```
ERROR: Expected data class specification.
```

Probable cause: The class section of the class specification must start with the `CLASS` keyword.

Corrective action: See Class Specification Syntax on page 25 for more formation.

Message SDL9

```
ERROR: Expected identifier.
```

Probable cause: An identifier for either the metric or class was expected. The identifier must start with an alphabetic character, can contain alphanumeric characters or underscores, and is not case-sensitive.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL10

```
ERROR: Expected positive integer.
```

Probably cause: Number form incorrect.

Corrective action: Enter numbers as positive integers only.

Message SDL13

```
ERROR: Expected specification for maximum number of indexes.
```

Probable cause: The maximum number of indexes is required to calculate class capacity.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL14

```
ERROR: Syntax Error.
```

Probable cause: The syntax you entered is incorrect.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL15

```
ERROR: Expected metric description.
```

Probable cause: Metric description is missing.

Corrective action: Enter a metric description to define the individual data items for the class. See METRIC Descriptions on page 35 for more information.

Message SDL16

```
ERROR: Expected metric type.
```

Probable cause: Each metric must have a *metric_name* and a numeric *metric_id*.

Corrective action: See METRIC Descriptions on page 35 for more information.

Message SDL17

```
ERROR: Time stamp metric attributes may not be changed.
```

Probable cause: A time stamp metric is automatically inserted as the first metric in each class. You can change the position of the time stamp, or eliminate it and use a UNIX time stamp.

Corrective action: See METRIC Descriptions on page 35 for more information.

Message SDL18

```
ERROR: Roll action limited to 199 characters.
```

Probable cause: The entry for upper limit for `ROLL BY` action exceeded 199 characters.

Corrective action: See INDEX BY, MAX INDEXES, AND ROLL BY on page 27 for more information about the `ROLL BY` entry.

Message SDL19

```
ERROR: Could not open specification file (file).
```

Probable cause: In the command line `sdlcomp` *specification_file*, the specification file could not be opened. The error follows in the next line as in:

```
$/usr/perf/bin/sdlcomp /xxx


ERROR: Could not open specification file /xxx.
```

Probable cause: File either does not exist or is not readable.

Corrective action: Verify the name of the file and that you are correctly entering it.

Message SDL20

```
ERROR: Metric declarations not found.
```

Probable cause: Metric description incorrectly formatted.

Corrective action: Start the metrics section with the `METRICS` keyword. See METRIC Descriptions on page 35 for more information.

Message SDL21

```
ERROR: Expected metric name to begin metric description.
```

Probable cause: Metric description incorrectly formatted.

Corrective action: Start metric descriptions with a *metric_name*. See METRIC Descriptions on page 35 for more information.

Message SDL24

```
ERROR: Expected MAX INDEXES specification.
```

Probable cause: A `MAX INDEXES` value is required when you specify `INDEX BY`.

Corrective action: Enter the required value. See INDEX BY, MAX INDEXES, AND ROLL BY on page 27 for more information.

Message SDL25

```
ERROR: Expected index SPAN specification.
```

Probable cause: Missing value for `INDEX BY`

Corrective action: Enter a qualifier when you specify `INDEX BY`. See INDEX BY, MAX INDEXES, AND ROLL BY on page 27 for more information.

Message SDL26

```
ERROR: Minimum must be zero.
```

Probable cause: The number must be zero or greater.

Message SDL27

```
ERROR: Expected positive integer.
```

Probable cause: Missing positive value.

Corrective action: Enter numbers as positive integers only.

Message SDL29

```
ERROR: Summarization metric does not exist.
```

Probable cause: You used `SUMMARIZED BY` for the summarization method, but did not specify a *metric_name*.

Corrective action: See METRIC Descriptions on page 35 for more information.

Message SDL30

```
ERROR: Expected 'HOUR', 'DAY', or 'MONTH'.
```

Probable cause: Missing qualifier for the entry.

Corrective action: You must enter one of these qualifiers. See INDEX BY, MAX INDEXES, AND ROLL BY on page 27 for more information.

Message SDL33

```
ERROR: Class id number must be between 1 and 999999.
```

Probable cause: The *class_id* must be numeric and can contain up to 6 digits.

Corrective action: Enter a class ID number for the class that does not exceed the 6-digit maximum. See Class Specification Syntax on page 25 for more information.

Message SDL35

```
ERROR: Found more than one index/capacity statement.
```

Probable cause: You can have only one `INDEX BY` or `CAPACITY` statement per `CLASS` section.

Corrective action: Complete the entries according to the formatting restrictions in Class Specification Syntax on page 25.

Message SDL36

```
ERROR: Found more than one metric type statement.
```

Probable cause: You can have only one `METRICS` keyword for each metric definition.

Correction action: See METRIC Descriptions on page 35 for formatting information.

Message SDL37

```
ERROR: Found more than one metric maximum statement.
```

Probable cause: You can have only one `MAXIMUM` statement for each metric definition.

Correction action: See METRIC Descriptions on page 35 for formatting information.

Message SDL39

```
ERROR: Found more than one metric summarization specification.
```

Probable cause: You can have only one summarization method (`TOTALED`, `AVERAGED`, or `SUMMARIZED BY`) for each metric definition.

Corrective action: See Summarization Method on page 37 for more information.

Message SDL40

```
ERROR: Found more than one label statement.
```

Probable cause: You can have only one `LABEL` for each metric or class definition.

Corrective action: See Class Specification Syntax on page 25 for more formation.

Message SDL42

```
ERROR: Found more than one metric precision statement.
```

Probable cause: The `PRECISION` statement limit was exceeded, which allows only one per metric definition.

Corrective action: See PRECISION on page 38 for more information.

Message SDL44

```
ERROR: SCALE, MINIMUM, MAXIMUM, (summarization) are inconsistent with text
metrics
```

Probable cause: These elements of the class specification syntax are valid only for numeric metrics.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL46

```
ERROR: Inappropriate summarization metric (!).
```

Probable cause: You cannot summarize by the `timestamp` metric.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL47

```
ERROR: Expected metric name.
```

Probable cause: Each METRICS statement must include a *metric_name*.

Corrective action: See METRIC Descriptions on page 35 for more information.

Message SDL48

```
ERROR: Expected positive integer.
```

Probable cause: The `CAPACITY` statement requires a positive integer.

Corrective action: See CAPACITY on page 33 for more information.

Message SDL49

```
ERROR: Expected metric specification statement.
```

Probable cause: The `METRICS` keyword must precede the first metric definition.

Corrective action: See METRIC Descriptions on page 35 for more information.

Message SDL50

```
ERROR: Object name too long.
```

Probable cause: The *metric_name* or *class_name* can only have up to 20 characters.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL51

```
ERROR: Label too long (max 20 chars).
```

Probable cause: The *class_label* or *metric_label* can only have up to 20 characters.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL53

```
ERROR: Metric must be between 1 and 999999.
```

Probable cause: The *metric_id* can have only 6 digits.

Corrective action: See METRIC Descriptions on page 35 in Chapter 3 for more information.

Message SDL54

```
ERROR: Found more than one collection rate statement.
```

Probable cause: You can have only one `RECORDS PER HOUR` statement.

Corrective action: See RECORDS PER HOUR on page 32 for more information.

Message SDL55

```
ERROR: Found more than one roll action statement.
```

Probable cause: You can have only one `ROLL BY` statement for each class specification.

Corrective action: See INDEX BY, MAX INDEXES, AND ROLL BY on page 27 for more information.

Message SDL56

```
ERROR: ROLL BY option cannot be specified without INDEX BY option.
```

Probable cause: The `ROLL BY` statement must be preceded by an `INDEX BY` statement.

Corrective action: See INDEX BY, MAX INDEXES, AND ROLL BY on page 27 for more information.

Message SDL57

```
ERROR: ROLL BY must specify time equal to or greater than INDEX BY.
```

Probable cause: Because the roll interval depends on the index interval to identify the data to discard, the `ROLL BY` time must be greater than or equal to the `INDEX BY` time.

Corrective action: See INDEX BY, MAX INDEXES, AND ROLL BY on page 27 for more information.

Message SDL58

```
ERROR: Metric cannot be used to summarize itself.
```

Probable cause: The `SUMMARIZED BY` metric cannot be the same as the *metric-name*.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL62

```
ERROR: Could not open SDL (name).
```

Probable cause: Explanatory messages will follow this error. It could be a file system error as in:

```
   $/usr/perf/bin/sdlutil xxxxx -classes
ERROR: Could not open SDL xxxxx.
ERROR: Could not open logfile set.
```

or it could be an internal error as in:

```
   $/usr/perf/bin/sdlutil xxxxx -classes
ERROR: Could not open SDL xxxxx.
ERROR: File is not SDL root file or the
description file is not accessible.
```

Probable cause: You may also see this error if the log file has been moved. Because the pathname information is stored in the DSI log files, the log files cannot be moved to different directories.

Corrective action: If the above description or the follow-up messages do not point to some obvious problem, use `sdlutil` to remove the log file set and rebuild it.

Message SDL63

```
ERROR: Some files in logfile set (name) are missing.
```

Probable cause: The list of files that make up the log file set was checked and one or more files needed for successful operation were not found.

Corrective action: Unless you know precisely what happened, the best action is to use `sdlutil` to remove the log file set and start over.

Message SDL66

```
ERROR: Could not open class (name).
```

Probable cause: An explanatory message will follow.

Corrective action: Unless it is obvious what the problem is, use `sdlutil` to remove the log file set and start over.

Message SDL67

```
ERROR: Add class failure.
```

Probable cause: The compiler could not add the new class to the log file set. Explanatory messages will follow.

Corrective action: If all the correct classes in the log file set are accessible, specify a new or different log file set. If they are not, use `sdlutil` to remove the log file set and start over.

Message SDL72

```
ERROR: Could not open export files (name).
```

Probable cause: The file to which the exported data was supposed to be written could not be opened.

Corrective action: Check to see if the export file path exists and what permissions it has.

Message SDL73

```
ERROR: Could not remove shared memory ID (name).
```

Not applicable on Windows NT/2000.

Message SDL74

```
ERROR: Not all files could be removed.
```

Probable cause: All the files in the log file set could not be removed.

Corrective action: Explanatory messages should follow. Delete the files using the Windows NT/2000 Command Prompt by entering:

```
sdlutil (logfile set) -files
```

or delete the files by selecting them and using Windows NT/2000 Explorer.

Message SDL80

```
ERROR: Summarization metric (metric) not found in class.
```

Probable cause: The `SUMMARIZED BY` metric was not previously defined in the `METRIC` section.

Corrective action: See METRIC Descriptions on page 35 for more information.

Message SDL81

```
ERROR: Metric id (id) already defined in SDL.
```

Probable cause: The *metric_id* only needs to be defined once. To reuse a metric definition that has already been defined in another class, specify just the *metric_name* without the *metric_id* or any other specifications.

Corrective action: See METRIC Descriptions on page 35 for more information.

Message SDL82

```
ERROR: Metric name (name) already defined in SDL.
```

Probable cause: The *metric_name* only needs to be defined once. To reuse a metric definition that has already been defined in another class, specify just the *metric_name* without the *metric_id* or any other specifications.

Corrective action: See METRIC Descriptions on page 35 for more information.

Message SDL83

```
ERROR: Class id (id) already defined in SDL.
```

Probable cause: The *class_id* only needs to be defined once. Check the spelling to be sure you have entered it correctly.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL84

```
ERROR: Class name (name) already defined in SDL.
```

Probable cause: The *class_name* only needs to be defined once. Check the spelling to be sure you have entered it correctly.

Corrective action: See Class Specification Syntax on page 25 for more information.

Message SDL85

```
ERROR: Must specify class to de-compile.
```

Probable cause: You must specify a *class list* when you use `-decomp`.

Corrective action: See Managing Data With Sdlutil on page 58 for more information.

Message SDL87

```
ERROR: You must specify maximum number of classes with -maxclass.
```

Probable cause: When you use the `-maxclass` option, you must specify the maximum number of classes to be provided for when creating a new log file set.

Corrective action: See Sdlcomp Compiler on page 43 for more information.

Message SDL88

```
ERROR: Option \"!\" is not valid.
```

Probable cause: The command line entry is not valid.

Corrective action: Check what you have entered to ensure that it follows the correct syntax.

Message SDL89

```
ERROR: Maximum number of classes (!) for -maxclass is not valid.
```

Probable cause: The `-maxclass` number must be greater than zero.

Corrective action: See Sdlcomp Compiler on page 43 for more information.

Message SDL90

```
ERROR: -f option but no result file specified.
```

Probable cause: You must specify a format file when using the `-f` option.

Corrective action: See DSI Logging Processes on page 50 for more information.

Message SDL91

```
ERROR: No specification file named.
```

Probable cause: No name assigned to class specification file.

Corrective action: You must enter a *specification_file* when using `sdlcomp`. See Sdlcomp Compiler on page 43 for more information.

Message SDL92

```
ERROR: No logfile set named.
```

Probable cause: No name entered for the log file set.

Corrective action: You must enter a *logfile_set* when using `sdlcomp`. See Sdlcomp Compiler on page 43 for more information.

Message SDL93

```
ERROR: Metric ID already defined in class.
```

Probable cause: The *metric_id* needs to be defined only once.

Corrective action: To reuse a metric definition that has already been defined in another class, specify just the *metric_name* without the *metric_id* or any other specifications. See METRIC Descriptions on page 35 for more information.

Message SDL94

```
ERROR: Metric name already defined in class.
```

Probable cause: The *metric_name* needs to be defined only once.

Corrective action: To reuse a metric definition that has already been defined in another class, specify just the *metric_name* without the *metric_id* or any other specifications. See METRIC Descriptions on page 35 for more information.

Message SDL95

```
ERROR: Text found after complete class specification.
```

Probable cause: The `sdlcomp` compiler found text it did not recognize as part of the class specification.

Corrective action: Reenter the specification and try again.

Message SDL96

```
ERROR: Collection rate statement not valid.
```

Probable cause: The proper format is `RECORDS PER HOUR` (number). The keywords must be present in this order and cannot be abbreviated.

Corrective action: Use the required format to correct the keyword.

Message SDL97

```
ERROR: Expecting integer between 1 and 2,147,483,647.
```

Probable cause: You must use a number in this range.

Corrective action: Enter a number that falls within the range.

Message SDL98

```
ERROR: Action requires preceding ROLL BY statement.
```

Probable cause: Entry out of order or missing in class specification file.

Corrective action: The action specifies what will happen when the log file rolls. It is important to first know when it should roll. `ROLL BY` must precede `ACTION`.

For example:

```
class xxxxx = 10
   index by month max indexes 12
   action "ll *";
```

should have been:

```
class xxxxx = 10
   index by month max indexes 12
   roll by month
   action "ll *";
```

Message SDL99

```
ERROR: MAX INDEXES must be preceded by INDEX BY statement.
```

Probable cause: Entry out of order or missing in class specification file.

Corrective Action: To specify a maximum number of indexes, the program needs to know where you are doing an indexing by. The `INDEX BY` statement must precede `MAX INDEXES`.

For example:

```
class xxxxx = 10
   max indexes 12
   label "this is a test";
```

should have been:

```
class xxxxx = 10
   index by month
   max indexes 12
   label "this is a test";
```

Message SDL100

```
WARNING: CAPACITY UNLIMITED not implemented, derived value used. (SDL-100)
```

Message SDL101

```
ERROR: Derived capacity too large. (SDL-101)
```

Message SDL102

```
ERROR: Text Length should not exceed 4096.
```

The maximum allowed length for the text metric is 4096.

Message SDL103

```
ERROR: RECORDS PER HOUR should not be greater than 3600 for logging summarized
data.
```

`Corrective Action:` The `RECORDS PER HOUR` can be greater than 3600 only for unsummarized data. Use the `-u` option to compile.

DSILOG Error Messages

Message DSILOG1

```
ERROR: Self describing log file not specified.
```

Probable cause: The log file name must be the first parameter passed to `dsilog`.

Corrective Action: Correct the command line and try again.

Message DSILOG2.

```
ERROR: Data class name not specified.
```

Probable cause: The data class must be the second parameter passed to `dsilog`.

Corrective Action: Correct the command line and try again.

Message DSILOG3

```
ERROR: Could not open data input file (name).
```

Probable cause: The file specified in the command line could not be opened.

Message DSILOG4

```
ERROR: OpenClass (\"name\") failed.
```

Probable cause: The class specified could not be opened. It may not be in the log file set specified, or its data file is not accessible.

Corrective Action: Explanatory messages will follow giving either an internal error description or a file system error.

Message DSILOG5

```
ERROR: Open of root log file (name) failed.
```

Probable cause: The log file set root file could not be opened. The reason is shown in the explanatory messages.

Message DSILOG6

```
ERROR: Time stamp not defined in data class.
```

Probable cause: The class was built and no time stamp was included.

Correction Action: Use `sdlutil` to remove the log file set and start over.

Message DSILOG7

```
ERROR: (Internal error) AddPoint ( ) failed.
```

Probable cause: `dsilog` tried to write a record to the data file and could not.

Correction Action: Explanatory messages will follow.

Message DSILOG8

```
ERROR: Invalid command line parameter(name).
```

Probable cause: The parameter shown was either not recognized as a valid command line option, or it was out of place in the command line.

Corrective Action: Correct the command line and try again.

Message DSILOG9

```
ERROR: Could not open format file(name).
```

Probable cause: The file directing the match of incoming metrics to those in the data class could not be found or was inaccessible.

Corrective Action: Check the class specification file to verify that it is present.

Message DSILOG10

```
ERROR: Illegal metric name (name).
```

Probable cause: The format file contained a metric name that was longer than the maximum metric name size, or the metric name did not look like a metric name.

Corrective Action: Make the correction and try again.

Message DSILOG11

```
ERROR: Too many input metrics defined. Max 100.
```

Probable cause: Only 100 metrics can be specified in the format file.

Corrective Action: The input should be reformatted externally to `dsilog`, or the data source should be split into two or more data sources.

Message DSILOG12

```
ERROR: Could not find metric (name)in class.
```

Probable cause: The metric name found in the format file could not be found in the data class.

Corrective action: Make corrections and try again.

Message DSILOG13

```
ERROR: Required time stamp not found in input specification.
```

Probable cause: The `-timestamp` command line option was used, but the format file did not specify where the time stamp could be found in the incoming data.

Corrective Action: Specify where the time stamp can be found.

Message DSILOG14

```
ERROR: (number) errors, collection aborted.
```

Probable cause: Serious errors were detected when setting up for collection.

Corrective Action: Correct the errors and retry. The `-vi` and `-vo` options can also be used to verify the data as it comes in and as it would be logged.

Message DSILOG15

```
ERROR: Self describing log file and data class not specified.
```

Probable cause: The command line must specify the log file set and the data class to log data to.

Corrective action: Make corrections to the command line and try again.

Message DSILOG16

```
ERROR: Self describing logfile set root file (name) could not be accessed.
error=(number).
```

Probable cause: Could not open the log file set root file.

Corrective action: Check the explanatory messages that follow the error text for the problem.

Message (unnumbered)

```
Metric null has invalid data
```

```
Ignore to end of line, metric value exceeds maximum
```

Probable cause: This warning message occurs when `dsilog` does not log any data for a particular line of input. This situation occurs when the input does not fit the format expected by the DSI log files, such as when blank or header lines are present in the input or when a metric value exceeds the specified precision. In this case, the offending lines will be skipped (*not* logged). `dsilog` will resume logging data for the next valid input line.

Corrective action: None; message is informational.

Message DSILOG17

```
ERROR: Logfile set is created to log unsummarized data, could not log
summarized data.
```

Corrective action: If the set of log files are created using the `-u` option during compilation, use `-s 0` option to log using `dsilog`. Using the option indicates that the data logged is unsummarized.

Index

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W