# VSI OpenVMS

# PERFDAT V4.8

## DQL$ Reference Manual

**February 2019**

| | |
|---|---|
| **Revision/Update Information** | New manual. |
| **Software Version** | VSI PERFDAT V4.8 |
| **Operating System Version** | OpenVMS Alpha V7.3-2 & higher<br>OpenVMS I64 V8.2 & higher |

**February 2019**

# Contents

# Preface

This manual includes:

- Description of the PERFDAT distributed database
- Introduction to the VSI PERFDAT Query Interface (DQL) and description of the components.
- Introduction to the DQL$ command line utility. This section provides an overview of to the command set available.
- Detailed description of the DQL$ command set alphabetically ordered.

## Audience

This manual provides a detailed description of the DQL$ command set. The reader should be familiar with

- VSI PERFDAT – Architecture and Technical Description

## Document Structure

- Chapter 1 VSI PERFDAT distributed performance database
- Chapter 2 VSI PERFDAT Query Interface (DQL)
- Chapter 3 Introduction to DQL$ command line utility
- Chapter 4 DQL$ reference section

## Conventions Used in this Manual

| | |
|---|---|
| Special | in examples indicates text that the system displays or user type input. |
| UPCASE | in a command represents text that you have to enter as shown. |
| *Lowercase Italics* | indicates variable information that a user supplies. |
| [ ] | in a command definition, enclose parts of the command that a user can omit. |
| Key | indicates a named key on the keyboard; for example, RETURN |
| CTRL/x | is the symbol used to represent the pressing of a control key. It indicates that the user holds down the key marked Ctrl and press the appropriate key. |

# VSI PERFDAT distributed performance database

## *Database Organization*

All data collected by the VSI PERFDAT OpenVMS data collector and the VSI PERFDAT SNMP extension are stored in index sequential RMS files. Each data collector (OpenVMS data collector, SNMP extension) creates a new file whenever a collection is started (restarted), or on date change. Thus, 1 to n data files can exist per day and collection. A single data file is called a physical storage area. All physical storage areas that are created on the same day and that belong to the same collection (collection profile & node) is called a logical storage area. All logical storage areas created by the same data collection make up a collection database. The sum of all collection databases available within your environment is called the VSI PERFDAT distributed performance database (see Fig.1.1).

The database alias for each collection database is automatically assigned and cannot be changed by the user. The format of the alias is:

*Nodename_collection-profile*

For example, an OpenVMS performance collection using a collection profile DEFAULT running on node BCSXTC creates and accesses the collection database BCSXTC_DEFAULT.

Trend and capacity report data files contain data of a particular time period – called a report period (day, week, month, quarter, year – for more information seethe manual VSI PERFDAT – Architecture and Technical Description) - defined by the report profile used to create these reports. At the end of such a predefined time period a new report data file is created, or whenever the statistics defined in the report profile changes. A single report data file is also called a physical storage area. A logical storage is the sum of all physical storage areas that are created during a report period. For example, if the report period is WEEK, all report data files created during a week make up the logical storage area for this report.

The VSI PERFDAT OpenVMS data collector and the VSI PERFDAT SNMP extension write the data directly via RMS calls to the appropriate data files. The trend engine does not directly access the data file, but inserts data via the DQL interface. Thus, as long as data files are write accessed by the VSI PERFDAT OpenVMS data collector and the VSI PERFDAT SNMP extension these files have to be stored locally. Data files, that are write accessed by the auto-trend engine or by the DQL$ utility can be stored on any member of the community (logical PERFDAT$COMMUNITY) the local node belongs to.

The VSI PERFDAT OpenVMS data collector as well as the VSI PERFDAT SNMP extension create the data files in the directory pointed to by the logical PERFDAT$DB_LOCAL on the local node.

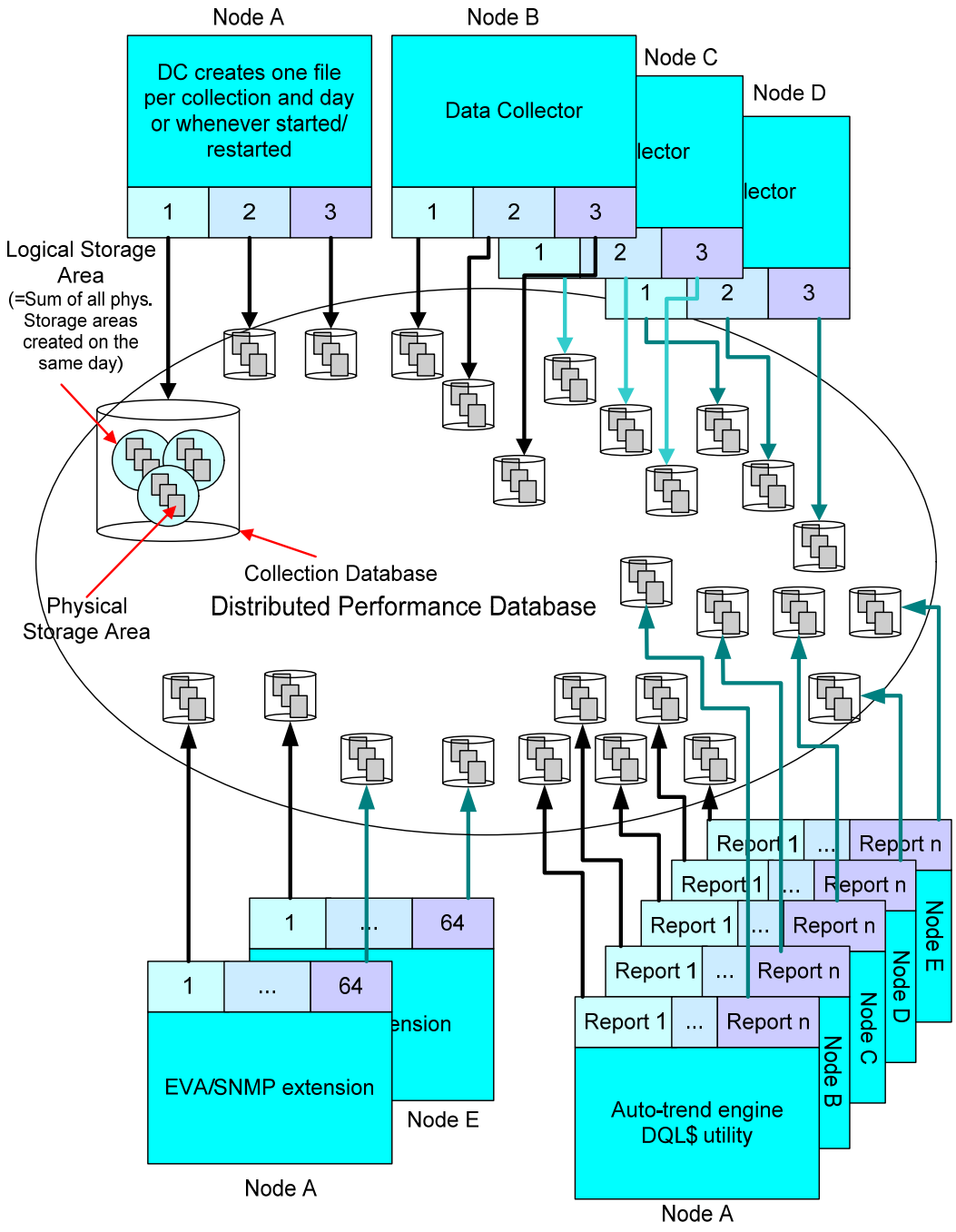



Fig. 1.1 Database organization of the PERFDAT distributed database.

Report data files created by the auto-trend engine as well as the data files (physical storage areas) created by the DQL$ utility are stored initially on the local node too. However in contrast to the collection data files created by the VSI PERFDAT OpenVMS data collector and the VSI PERFDAT SNMP extension these files can be moved to another node within the PERFDAT community (e.g. archive node) after write access has been released by the auto-trend

engine or the DQL$ utility, even if these files are reopened later on for data insert, since the data is always accessed via the DQL interface.

Thus, report data files can be relocated between subsequent report processing runs to another node of the PERFDAT community even if the auto-trend engine inserts data into the (relocated) data files again the next time.

The query interface to the distributed performance database is similar to SQL. All basic query statements such as SELECT, INSERT, CREATE, DROP etc. except UPDATE and DELETE to prevent after image data manipulation are supported. The main difference between a relational database such as Oracle, Informix, mySQL etc. is that no root file exists for each database. All meta-data (field and record descriptors, data link descriptors, index reference table descriptor …) necessary to access the data in the file is stored in the header of each physical storage area.

The advantage is that even if a physical storage area is moved to any other OpenVMS node (target node) that is not member of the community that physical storage stays read accessible to DQL without any additional actions such as data conversion, un-load and load operations. Thus, any data file can be copied to any OpenVMS node (target node) for offline analysis. The only prerequisite is that the DQL environment is installed on that OpenVMS node and it is configured as an archive node, or you simply define one of these logicals:

- PERFDAT$NODEDATA_HOSTED
- PERFDAT$NO_NODE_FILTER

For more information about these logicals see the manual VSI PERFDAT – Architecture and Technical Description. Version compatibility (= same versions) of OpenVMS and PERFDAT between where the node data is collected and where the node data is analysed is NOT required.

Data within physical storage areas are organized in METRICES, ELEMENTS and STATISTICS. A METRIC consists of 1 to n ELEMENTS, and each ELEMENT consists of 1 to n STATISTICS.

Comparing this structure to a classic database organization the following comparisons are valid

- A METRIC is comparable to a TABLE.
- An ELEMENT is comparable to an INDEX of a TABLE.
- A STATISTIC is comparable to a FIELD within a TABLE.

## *Directory structure*

Since no root file is involved for accessing the data files, directories are not freely definable. They have to be stored in one of the directories listed below.

- PERFDAT$DB_LOCAL
- PERFDAT$DB_ARCHIVE
- PERFDAT$DB_TREND

- PERFDAT$DB_SAVE
- PERFDAT$DB

These logical directories have to exist on any node within the PERFDAT environment. Otherwise some or all PERFDAT components may fail.

**PERFDAT$DB_LOCAL**

This is the default directory for creating performance collection data files. The VSI PERFDAT OpenVMS data collector and the SNMP extension create files in this directory. If the logical does not exist or does not reference a valid physical directory, the data collectors fail.

**PERFDAT$DB_ARCHIVE**

All closed data files created by the VSI PERFDAT OpenVMS data collector and the VSI PERFDAT SNMP extension are periodically moved to this directory by the archiving process. The data files in this directory are managed by the archiving process. It guarantees that all files are kept for a predefined period of time (default 30 days – for more information see the manuals VSI PERFDAT – Architecture and Technical Description and VSI PERFDAT– PERFDAT_MGR Reference Manual). Physical storage areas that are older than the defined keep time are automatically and unconditionally deleted. If this logical does not exist, or the logical does not refer a valid physical directory the archiving process fails.

**PERFDAT$DB_TREND**

Report data files are created in this directory. Between two subsequent report runs, the report data file can be moved to the same directory on any other node within the same community (e.g. archive node) and the report data file stay write accessible to the auto-trend engine as explained in the previous sections. If the logical does not exist or the logical does not refer to a valid physical directory the auto-trend engine as well any report extraction manually done via the DQL$ utility will fail.

**PERFDAT$DB_SAVE**

This directory is used for defining baseline performance data. A baseline is a set of logical storage areas (performance collection data) that covers a full week. The baseline represents a week where the system performance is deemed normal based on the customer's knowledge and experience. The baseline has to be defined by the system manager by moving the appropriate logical storage areas of a collection database to this directory. If the logical directory does not exist or the logical does not reference a valid physical directory a performance baseline cannot be defined. Thus, baseline deviation reports will fail.

**PERFDAT$DB**

When accessing the distributed performance database the DQL interface scans this directory on each member of the community for collection databases of the community members. PERFDAT$DB is a directory search list. Per default PERFDAT$DB refers to:

- PERFDAT$DB_LOCAL
- PERFDAT$DB_ARCHIVE
- PERFDAT$DB_TREND
- PERFDAT$DB_SAVE

This search list can be extended at any time. If this logical is not defined or it does not refer to valid logical or physical directories the whole database or parts of it will not be accessible via the DQL interface.

# VSI PERFDAT Query Interface (DQL)

The VSI PERFDAT query interface DQL (Data Query Language) is the common data access layer to the distributed performance database. It provides a data abstraction and a network abstraction layer.

As described in chapter VSI PERFDAT distributed performance database, any data file within the distributed performance database stores all meta-data necessary to access the data in the file header (field and record descriptors, data link descriptors, index reference table descriptor etc.). Due to the fact that the query interface needs no implicit knowledge about the internal structure of the data files, there exists no version dependency when accessing the data. This data abstraction layer guarantees version independency for read access.

The network abstraction layer grants transparent access to any data within the defined community. A community is a logical partition of the whole environment and defines the database view when accessing the data via one system within a community regardless of where the data files are actually stored within the community. Systems of particular interest to a PERFDAT user can be configured in the context of a community. The systems that belong to a particular community is freely definable e.g. all members of a cluster might be part of a particular community, or standalone systems running the same application may be part of another community. The community definition is not cluster bound.

The command syntax of DQL is similar to SQL and makes data query easy.

## *Prerequisites*

A supported TCP/IP stack for OpenVMS has to be installed and configured in order to use VSI PERFDAT and the DQL query interface.

## *Features*

- Query interface (DQL) similar to SQL
- Transparent single point access via network abstraction layer
- Up- and downward data compatibility via data abstraction layer
- Dynamic CSV file mapping capability for accessing and analysing data from different data sources
- Multi file version support
- CSV load capability
- CSV file import capability (data is not only inserted but also normalized)

- CSV export capability
- Statistic package fully integrated in data query interface
- Stored procedures (= user-defined statistics).
- Data Clustering (= ability to define cluster views. This feature enables the user to run cluster wide data analysis without any change in the workflow. All methods and features to analyse performance data of single nodes are available for cluster views too).

## *Components*

The query interface is not a monolithic layer but consists of six components as shown in Fig. 2.1.

- DQL$SRV (DQL server)
- Cluster view engine
- Stored procedure engine
- Statistics Package
- DQL$ command line utility
- PDBC$SRV (Performance database connectivity server)

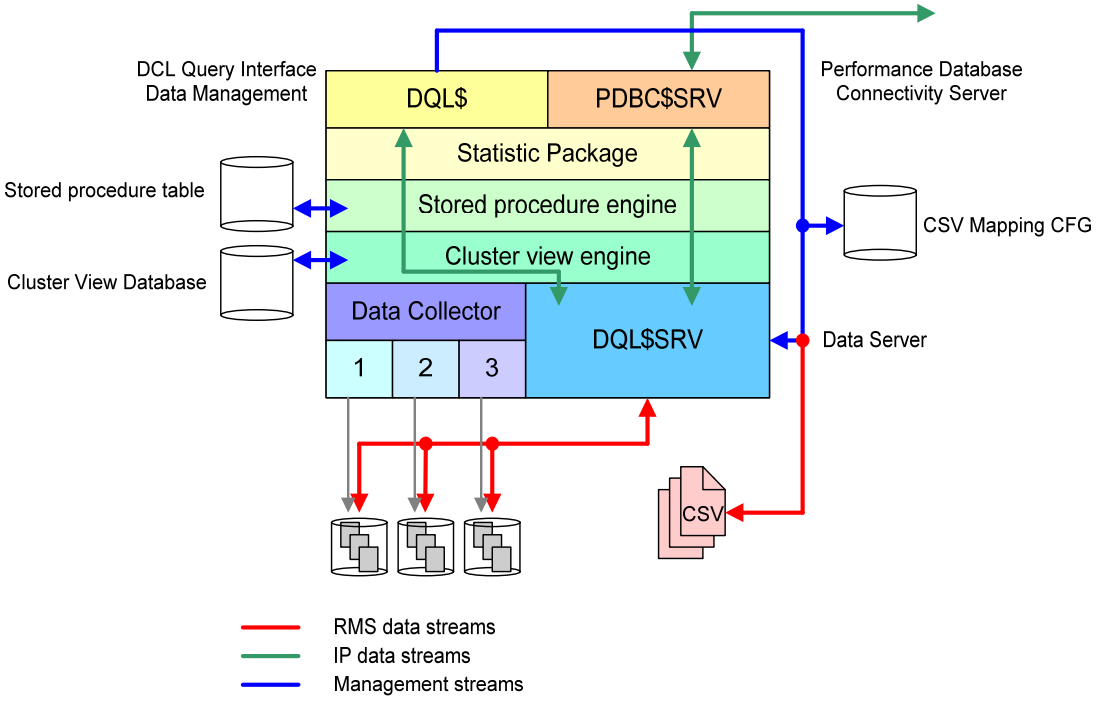



Fig. 2.1 Components of the DQL interface.

**DQL$SRV (DQL server)**

The DQL$SRV (DQL server) represents the data abstraction layer of the DQL interface. This component directly accesses the data of the performance data stored locally according to the definitions in the header of the data files. Its main task is to map the data query command received from the cluster view engine to RMS calls. Data read from the data files are converted into type

independent format and returned compressed to the calling layer. It handles data of the collection databases stored locally as well as CSV files mapped locally. In order to access a CSV file DQL$SRV reads the CSV descriptors from the CSV mapping database (see Fig. 2.1) that defines the layout of the CSV file. The CSV mapping (inserting the CSV descriptors into the CSV mapping database) has to be done manually by the user using the MAP command of the DQL$ command line utility (see the DQL$ command line utility reference section).

The DQL server is implemented as an IP service. Thus, any node within your environment can request data from the DQL$SRV. Up to 99 DQL$SRV connections are allowed per node. Each DQL$SRV process can access up to 2048 data files concurrently.

The default listener port number of the DQL$SRV service is 3879. It can be redefined by setting the system-wide logical DQL$SRV_PORT. Whenever this logical has been modified the DQL$SRV service has to be restarted using the LAUNCH DQL$SRV command of the PERFDAT_MGR utility.

---

**Note**

Once the DQL$SRV_PORT logical has been modified and the DQL$SRV service has been restarted, you have to modify the following logicals on all nodes where these logicals refer to the node you have redefined as the DQL$SRV listener port:

- PERFDAT$COMMUNITY
- PERFDAT$ARCHIVE_NODE

To signal all VSI PERFDAT components that the DQL$SRV listener port on a remote node differs from the default (3879), you have to add the new DQL$SRV listener port to the node name string separated by a semicolon when you define these system-wide logicals.

---

All informational, warning and error messages during the runtime of the DQL$SRV processes are posted to OPCOM and stored in log-files. The log-files are located in the PERFDAT$LOG directory. The filename has the following format

DQL$SRV_*nodename*.LOG

The DQL$SRV component can be explicitly started by invoking either of the commands listed below, since there exists no dependency to any other PERFDAT SW-component:

- $ MCR PERFDAT_MGR LAUNCH DQL$SRV
- $ @SYS$STARTUP:DQL$SRV_STARTUP.COM

Starting DQL$SRV in stand-alone mode may be important if the local node hosts the distributed performance database or parts of it, but the local node will not be used as an access server for the GUI.

**Cluster view engine**

The cluster view engine provides the feature to map performance data of different nodes for cluster wide performance analysis. Once a cluster view is created a virtual collection database is accessible that refers and maps the data of the cluster view members. The advantage is that such a virtual cluster view collection database can be accessed in the same way by the DQL$ utility and the PERFDAT GUI as if it is a collection database created by the OpenVMS data collector or the SNMP extension. Thus, all methods and features to analyse performance data of single nodes are available for cluster views too. Consequently the workflow to analyse cluster view performance data does not differ from the workflow to analyse single node performance data.

Although in most cases cluster views will be created for cluster wide performance data analysis of OpenVMS clusters there exists no restriction that performance collection databases of OpenVMS cluster members only can be members of a cluster view. Any collection database of any node available can be added to a cluster view. The only restriction is that all collection databases of a cluster view were created with the same sample interval.

Any data query is passed to the cluster view engine. If the data query requests cluster view data, appropriate data queries are created for all members (= collection databases) of the cluster view. These queries are sent to DQL$SRV. The data streams received from DQL$SRV are merged (stacked) and the merged (stacked) data stream is returned to the calling layer. If the data query received contains no cluster view data requests the query is directly bypassed to DQL$SRV.

Cluster views can be configured using the DQL$ utility or the GUI. Cluster view definitions are node specific. Thus, a cluster view can be accessed by those users only that are connected to the distributed PERFDAT performance data via the same node on which the cluster view was configured. Cluster view definitions are stored in node specific cluster view databases. The file names of the cluster view databases have the format:

PERFDAT$CFG:DQL_VIEW_*nodename*.CFG

The cluster view databases are accessed by the cluster view engine only. Any cluster view configuration request from the DQL$ utility or the GUI is passed to the cluster view engine. It verifies:

- If all collection databases addressed by the cluster view exist and if there exists at least one matching logical storage area within each collection database. With other words, for at least one day performance data have to exist in all collection databases.
- If all collection databases addressed by the cluster view were created with the same sample interval

If one of these checks fail the configuration request is rejected. Otherwise the view definition is stored in the node specific cluster view database and the newly created cluster view is immediately accessible by the user.

**Stored procedure engine**

The stored procedure engine enables the user to define side (community, node) specific measures (statistics). Such user-defined statistics are calculated values and they are created using the DEFINE PROCEDURE command of the DQL$ utility by assigning a function (procedure) to a freely definable statistics name (For more information about defining stored procedures / user-defined statistics please refer to section DQL$ command line utility reference section of this manual). Statistics collected by the OpenVMS data collector or the SNMP extension, existing user-defined statistics and constant values can be used within the function (procedure) assigned. The supported operators are +, -, * and /.

Any data query is passed to the stored procedure engine. If the data query requests user-defined statistics, the data query is modified to request all base statistics necessary to calculate the user-defined statistics. The modified query is passed to the cluster view engine. Once the stored procedure engine receives data from the cluster view engine the user-defined statistics is calculated according to the assigned function (procedure) and the result is returned to the caller. In case of stacked requests (SELECT and CALCULATE queries) the input statistics are stacked before calculating the user-defined statistics.

User-defined statistics are marked with a dollar ($) sign in front to indicate that they are calculated statistics. The user can, but doesn't, have to enter the dollar ($) sign when defining the stored procedure. If the dollar sign is omitted it is automatically assigned.

User-defined statistics and the assigned functions (procedures) are stored in the stored procedure table of the PERFDAT configuration database. Thus, once a user-defined statistics has been successfully defined it is immediately accessible by all users accessing data via one of the nodes that share the same PERFDAT configuration database.

The stored procedure table of the PERFDAT configuration database is accessed by the stored procedure engine only. Any stored procedure (user-defined statistics) configuration request is passed to the stored procedure engine. It performs several checks before it inserts the user-defined statistics into the stored procedure table of the PERFDAT configuration database:

- It checks if all statistics defined within the function (procedure) assigned to the user-defined statistics exist.
- It checks the syntax of the function (procedure) assigned
    - It checks if all brackets are present
    - It checks if supported operators (+, -, *, /) are applied only

If one of these checks fails the configuration request is rejected.

There are several reasons to use this feature. Here are some examples:

- This feature is important in case you want to normalize data. E.g. the statistics for the system wide CPU load collected by the OpenVMS data collector ranges from 0 … 100% * number of CPUs. Thus, if you are monitoring a system with 8 CPUs the statistics for the system wide

CPU load collected by the OpenVMS data collector ranges from 0 … 800 %. In order to fetch normalized data of the system wide CPU load ranging from 0 … 100 % a user-defined statistics can be created (in this example the user-defined statistics is named $iCpuNorm, but you can choose any other name)

$iCpuNorm = iCpuLoad / iCpuCnt

where

| | |
|---|---|
| iCpuLoad | system wide CPU load collected by the OpenVMS data collector |
| iCpuCnt | number of CPUs. |

- You can use this feature to create special statistics you are interested in if these statistics are not directly collected by the OpenVMS data collector or the SNMP extension but all input parameters to compute them are available. E.g. you are interested in the average I/O size of disk I/Os. The average I/O size is not collected by the OpenVMS data collector but the number of I/Os to the device (iIOs) and the throughput (iMBs) is collected. If you request the data of the user-defined statistics

  $iIOSize = iMbs / iIOs

  for a disk device the average I/O size values are returned.

**Statistics package**

Any query is passed to the statistics layer. The query is analysed if it contains a statistics request. If this is the case appropriate data queries are sent to stored procedure engine. The data received from the stored procedure engine are decompressed, locally cached, processed according to the statistics request and the final result is returned to the caller. If the query is a data query the query is bypassed directly to the stored procedure engine. (For more information about the statistics package please refer to the manual VSI PERFDAT – Architecture and Technical Description).

**DQL$ command line utility**

The DQL$ command line utility services interactive data and statistics query from the DCL command line. The DQL$ command line utility and the performance data connectivity server (PDBC$SRV) represent the network abstraction layer of the DQL interface. Its main tasks are:

- Creating a virtual root file (memory resident) whenever a user connects to the distributed performance database via the DQL$ command line utility. This is done by checking the community definition (PERFDAT$COMMNUITY), establishing connections to DQL$SRV on the nodes listed in the logical PERFDAT$COMMNUITY, the archive node if defined, plus the local node. Once the connection is established DQL$ asks the DQL server to return all known data files. The DQL$ command line utility filters these files that belong to the community (data files that are created by the members of the community) and caches the data file links. Thus, the DQL$ keeps the knowledge were the data files are located and how to access.

- Passing the data and statistics queries to the appropriate nodes that host the data files. If the query refers to data files that are stored on different nodes, the DQL$ command line utility de-assembles the query, forwards appropriate queries to the nodes, consolidates the data received and returns the result to the caller.

**PDBC$SRV (Performance data connectivity server)**

The performance data connectivity server is like the DQL$ command line utility responsibly for transparent access to the data files within the defined community (network abstraction). As with DQL$ services interactive requests from the DCL command line PDBC$SRV services GUI requests.

The performance data connectivity server is implemented in a similar manner to the DQL$SRV as an IP service. Up to 99 concurrent PDBC$SRV (PC-client) connections are allowed per node.

The default listener port number of the PDBC$SRV service is 5254. It can be redefined by setting the system-wide logical PDBC$SRV_PORT. Whenever this logical has been modified the PDBC$SRV service has to be restarted using the LAUNCH PDBC$SRV command of the PERFDAT_MGR utility.

All informational, warning and error messages during the runtime of PDBC$SRV processes are posted to OPCOM and stored in log-files. The log-files are located in the directory PERFDAT$LOG. The filename has the following format

PDBC$SRV_*nodename*.LOG

The PDBC$SRV component can be explicitly started by invoking either of the commands listed below, since there exists no dependency to any other PERFDAT SW-component:

- *$ MCR PERFDAT_MGR LAUNCH DQL$SRV*
- *$ @SYS$STARTUPDQL$SRV_STARTUP.COM*

Starting PDBC$SRV in stand-alone mode may be important if the local node is be used as an access server (GUI) only, and no collection data files are stored on the node.

If you have not installed TCPIP for OpenVMS on your system but you are using another product such as MultiNet or TCPware then please modify DQL$SRV_STARTUP.COM accordingly to add the DQL$SRV IP service.

# Introduction to DQL$ command line utility

As described in chapter VSI PERFDAT Query Interface (DQL) the DQL$ command line utility is the DCL interface to handle interactive database requests.

This section provides an overview of to the command set available. The DQL$ command set consists of four main groups

- Data query commands
- Statistics query commands
- Report extraction command
- Data content management commands

You can execute DQL$ command scripts using the @ command. A command script can be any text file that contains valid DQL$ commands.

## *Data query commands*

Table 3.1 summarizes the data query commands available. For more detailed information about the available commands please refer to chapter DQL$ command line utility reference section.

Table 3.1 Data query command summary table

| Command | Description |
|---|---|
| ATTACH | Attach a physical or logical storage area or a whole collection database of the distributed performance database. |
| DEATTACH | Disconnect from a physical or logical storage area or a collection database of the distributed performance database. |
| DEFINE HEADER | This command can be applied in advance of the EXPORT and the CREATE GRAPH command in order to enter a user-defined header line for the CSV file or a user-defined caption for the graph created. |
| INSERT | Insert fields of a record or the whole record into an existing metric of a physical storage area |
| SET TRANSACTION | Set the transaction access for a physical or logical storage area or a whole collection database of the distributed performance database. The transaction access can be<br>• READ ONLY (default)<br>• READ WRITE |
| EXPORT | Exports 1…n statistics from a metric of attached physical or logical storage areas or whole collection databases to a CSV file. |

| | |
|---|---|
| SELECT | Reads data fields from a metric of attached physical or logical storage areas or whole collection databases and displays the data on screen. |
| CREATE GRAPH | Reads data fields from a metric of attached physical or logical storage areas or whole collection databases, creates graphs from the data, converts these graphs into PNG format and stores them either in a user-defined directory or in the directory PERFDAT$GRAPH. |

## *Statistics query commands*

Table 3.3 summarizes the statistics query commands available. For more detailed information about the available commands please refer to chapter DQL$ command line utility reference section.

Table 3.3 Statistics query command summary table

| Command | Description |
|---|---|
| DEFINE ELEMENT | This command can be applied in advance of the stacked form of CALCULATE command in order to assign a user-defined element name to the stacked element of a stacked calculation or deviation report. |
| CALCULATE | This command is used for two different types of calculations<br>• Depending on the parameters applied, this command calculates the arithmetic mean value, integral mean value, max value, standard deviation or all of this for 1...n statistics of a metric.<br>• Deviation analysis<br>1...n statistics of a metric are read from two different logical storage areas (= data of different days). These data are averaged and compared to each other. The percentage the source data average differs from the reference data average is displayed for each statistics and element. The deviation analysis can be done integral or arithmetic based. |
| CORRELATE | Calculates the correlation between different statistics of different metrics within a logical storage area. |

## *Report extraction command*

Table 3.3 describes the report extraction command in brief. For more detailed information about the available commands please refer to chapter DQL$ command line utility reference section.

Table 3.3 Report extraction command summary table

| Command | Description |
|---|---|
| EXTRACT | The EXTRACT command is used to create reports (trend, capacity and baseline reports) according to predefined report profiles. With the EXTRACT command you can apply any predefined report profile to any collection database. The only restriction is that the report is of the same type as the collection database (OpenVMS reports can only be applied to OpenVMS collection databases, Tru64 reports can only be applied to Tru64 collection databases and so on). |

## *Data content management commands*

Table 3.4 summarizes the data content management commands available. For more detailed information about the available commands please refer to chapter DQL$ command line utility.

Table 3.4 Data content management command summary table

| Command | Description |
|---|---|
| CHECK | This command checks if the CSV files addressed by the CSV mapping entries in the CSV mapping database are valid. |
| CREATE | Depending on the parameters applied the CREATE command is used for creating a new physical storage area or a new metric (table) within an existing physical storage area. |
| CONVERT | Converts the header(s) of physical storage areas created by an older VSI PERFDAT version than actually used to new format. Collection databases have to be converted to actual format if data shall be inserted. For read access there is no need to convert the collection databases. |
| DEFINE | • DATA HOST<br><br>The DEFINE DATA HOST command defines the node (data host) were new data files created during the current DQL$ session will be stored. The user can define any community member or the archive node as the new data host.<br><br>• ELEMENT<br><br>When a stacked element report or a stacked deviation report is created these reports contain the calculated values for a single (stacked) element. This stacked element name can be (re)defined by applying the DEFINE ELEMENT command in advance of the CALCULATE statement. |

- HEDAER

  If data are exported to a CSV file the header line (comment) of that file and the caption of a graph created by applying the CREATE GRAPH command can be user-defined. This is done by applying this command in advance of the EXPORT and CREATE GRAPH command.

- PROCEDURE

  Defines side specific, calculated statistics (measures) that can, once defined, be accessed as if they are part of the associated metrics of the collection databases available.

- REGION

  Used to define regional settings. The feature to define regional settings increases the flexibility to import, load and mapping CSV files of different format and to export data to a CSV file formatted as expected by the system the CSV will transferred to.

- VIEW

  Creates a cluster view. A cluster view maps performance data of different nodes for cluster wide performance analysis.

DROP

Depending on the parameters applied the DROP command deletes a metric within a physical storage area, a physical storage area, a logical storage area or a whole collection database.

IMPORT

Imports data of a CSV file into an existing collection database. Prerequisite for importing data of a CSV file is a valid descriptor file for the CSV file, and the CSV file has to contain a time column. For more information about creating a CSV descriptor file please refer to the MAP command description in the command reference section in this manual.

Using the IMPORT command CSV data are normalized before they are inserted into the collection database. It is very likely that the timestamps in the time column do not match the timestamps in the collection database. This is a prerequisite when correlating data. Any correlation based on data that does not match in time (timestamp of a sample, sample interval) will return wrong results. Normalizing means that based on the CSV data expectancy values are calculated for the timestamps of the collection database. An integral based algorithm is used to normalize the data.

LIST

- METRIX

  This command displays all the performance metrics (tables) stored in the record descriptor table of the VSI PERFDAT configuration database.

- STATISTICS

  The LIST STATISTICS command displays the statistics stored in the record description table of the VSI PERFDAT configuration database and the user-defined statistics (stored procedures) of a particular performance metric (table). The field name, data type, field length and the field description is displayed.

LOAD

Loads data of a CSV file into an existing collection database. Prerequisite for importing data of a CSV file is a valid descriptor file for the CSV file, and the CSV file has to contain a time column. For more information about creating a CSV descriptor file please refer to the MAP command description in the command reference section in this manual.

Using the LOAD command CSV data are not pre-processed (normalized) before they are inserted into the collection database. The LOAD command should only be used if the timestamps in the CSV file match exactly the timestamps in the collection database. Otherwise it is recommended to use the IMPORT command. The LOAD command consumes less system resources and is faster than the IMPORT command.

MAP

Maps CSV files to the distributed performance database. Mapping a CSV files means that the data of the CSV file can be accessed as if they are part of a collection database. Prerequisite for mapping CSV files are

- In the first row of the CSV file the nodes the data shall be visible to are inserted as a comma separated list.
- The second row has to contain the column headers. Max length of a header item is 64 characters. Max number of columns is 200.
- One column header item has to be named 'Time' and the data format of that column has to be OpenVMS date/time format.
- The data rows of the CSV file have to be ordered descending by the 'Time' column.
- A CSV descriptor file that contains a valid CSV record descriptor. For more information about creating a CSV descriptors and CSV descriptor files please refer to to the MAP command description in the command reference section in this manual.

It does not matter if the CSV file includes data of different days. DQL splits the file virtually in order to map the CSV file content to the database scheme.

There may co-exist several rows with the same time-stamp. In that case 1 to max 3 columns can be defined as index fields in the CSV descriptor file. It is not allowed to have duplicates in the CSV file. A duplicate record contains the same the timestamp and the index fields contain the same data as another record.

Mapped CSV files can be accessed read only.

CSV data content cannot be correlated to other CSV file content or to data of a collection database.

CSV file mappings are only valid on the node where the mapping is done but the CSV content can be accessed by any member of the community the node that hosts the CSV file belongs to in case the node(s) listed in the first line of the CSV file are also member(s) of the community.

Mapped CSV files are not managed by the VSI PERFDAT environment. They have to be managed by the system manager.

In order to map CSV file content a record descriptor is required to define the record layout of the CSV file(s). CSV record descriptors are stored in so called CSV descriptor files. The CSV record descriptor file is inserted into the CSV

mapping database. PERFDAT$CFG:CSV_PROFILES.CFG when the MAP command is executed. Thus, regardless if the CSV descriptor file is deleted afterwards, the CSV mapping will be valid till the CSV mapping is manually removed from the CSV mapping database.

REMOVE

- CSV File Mapping

  Removes a valid CSV mapping from the CSV mapping database.

- PROCEDURE

  Removes particular user-defined statistics from the stored procedure table of the VSI PERFDAT configuration database.

- REGION

  Removes existing regional settings from the regional setting table of the VSI PERFDAT configuration database.

- VIEW

  Removes an existing cluster view.

SET

- REGION

  Changes the default regional setting for the current DQL$ session and all subsequent DQL$ sessions started on any node that share the same VSI PERFDAT configuration database.

SHOW

Depending on the parameters applied one receives the information listed below

- DATABASE

  Shows all collection databases accessible within the community.

- LOGICAL STORAGE AREA

  Shows all logical storage areas within a particular collection database.

- PHYSICAL STORAGE AREA

  Depending on the parameter applied, the physical storage areas within a logical storage area or within a whole collection database are displayed.

- HEADER

  Shows the headers of all attached physical storage areas.

- METRIC

  Shows the metrics (tables) available from each attached physical storage area.

- STATISTICS

  Shows the statistics available of a particular metric.

- ELEMENT

  Shows all elements of a particular metric stored in all physical storage areas attached. Elements can be sorted by any field of the metric to get hot element statistics easily.

- CSV File Mapping

  Shows the CSV mappings configured on the local

node.

- PROCEDURE

  Displays the user-defined statistics stored in the stored procedure table of the VSI PERFDAT configuration database.

- REGION

  Displays the default regional setting of the current DQL$ session and all regional settings defined in the regional setting table of the VSI PERFDAT configuration database.

- VERSION

  The SHOW VSERION command displays the version of the DQL$ utility and DQL$SRV of the node the current DQL$ session is connected to

- VIEW

  Displays the cluster views configured on the local node (content of the local cluster view database).

# DQL$ command line utility reference section

This section alphabetically describes the available commands of the DQL$ command set in detail.

The DQL$ command line utility grants transparent access to the data files within the defined community (network abstraction). The DQL$ utility services interactive requests from DCL command line. To invoke the DQL$ command line utility enter the following command at the DCL prompt:

*$ MCR DQL$ [/REGION=regional setting]*

The /REGION command qualifier can be applied. It defines the default regional setting of the DQL$ session. If this command qualifier is omitted the default regional setting stored in the regional setting table of the PERFDAT configuration database is used (for more information about regional settings please refer to the DEFINE REGION command description).

**Note**

Enter the /REGION command qualifier blank separated right after the image activation MCR DQL$. Otherwise the qualifier is not passed to the DQL$ image and the regional setting will not be changed.

All statements of the DQL$ command line utility but the @, EXIT and HELP command have to terminated by a semicolon (;). The maximum length of the command input supported by DQL$ is 2048 characters.

# @

Executes valid DQL$ command script.

## Format

@file_name

## Description

Executes valid DQL$ command script. DQL$ is requested to execute subsequent DQL$ commands from a specific file specified by file_name. To add a comment place an exclamation mark (!) at the first position of the line that contains the comment.

## Example

The DQL$ command script SYS$LOGIN:EXPORT.DQL contains statements to export data to a CSV file:

```
!
! DEFINE HEADER
!
DEFINE HEADER "CLUSTER WIDE CPU LOAD / KERNEL MODE OF PERFDAT*, DQL*";
!
! EXPORT TO CSV FILE
!
EXPORT STACKED TIME, ICPULOAD, IKERNEL FROM PROCESS
ALIAS BCSXTC_DEFAULT,HOBEL_DEFAULT DATE 30-AUG-2005
ELEMENT PERFDAT*,DQL*
INTO SYS$LOGIN:PERFDAT_30052005.CSV;
!
! DISCONNECT FROM LOGICAL STORAGE AREAS
!
DEATTACH ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005;
DEATTACH ALIAS HOBEL_DEFAULT DATE 30-AUG-2005;
```

Execute the DQL$ command script:

```
DQL>@SYS$LOGIN:EXPORT.DQL
DQL-I-ATTACH, successfully attached file /BCSXTC_DEFAULT_2005-08-30:00:03:00:1/
DQL-I-ATTACH, successfully attached file /HOBEL_DEFAULT_2005-08-30:00:03:00:1/
DQL-EXPORT, start export data to /SYS$LOGIN:PERFDAT_30052005.CSV/
DQL-I-ATTACH, successfully deatached file /BCSXTC_DEFAULT_2005-08-30:00:03:00:1/
DQL-I-ATTACH, successfully deattached file /HOBEL_DEFAULT_2005-08-30:00:03:00:1/
```

For detailed information about the commands in the DQL$ command script please see the appropriate command descriptions.

## *ATTACH ALIAS*

This command opens a collection database or logical storage area for Read/Write access.

### Format

ATTACH ALIAS alias_name [DATE date] [EXCLUSIVE];

### Description

This command opens a collection database or a logical storage area according to the access mode (Read/Write – Read Only) defined by the SET TRANSACTION command. The default access mode is Read Only. Once the command returns successfully the collection database/logical storage area stays accessible for the runtime of the DQL$ session until you explicitly disconnect from the collection database/logical storage area using the DEATTACH command.

The ALIAS clause specifies the alias of the collection database to open. That database alias can't be user-defined. DQL$ assigns these aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

If you want to access a logical storage area (all data files that have been created on the same day) of a collection database, the DATE clause is mandatory. Use OpenVMS date format to define the day of interest. If you omit the DATE clause all data files (physical storage areas) of the collection database defined by the ALIAS clause are opened.

If you omit the EXCLUSIVE keyword the collection database/logical storage area is opened shareable. Using the EXCLUSIVE keyword the collection database/logical storage area is opened non-shareable.

**Note**

If you apply the EXCLUSIVE keyword the collection database/logical storage area is always opened for Read/Write access independently of the transaction type defined by the SET TRANSACTION command.

For further information about the distributed collection database please refer to chapter VSI PERFDAT distributed performance database.

**Examples**

Example 1

This is example shows how to attach a collection database.

```
DQL>ATTACH ALIAS VNOABS_2MIN;
DQL-I-ATTACH, successfully attached file /VNOABS_2MIN_2003-SEP-19:00:03:00/
.
.
.
DQL-I-ATTACH, successfully attached file /VNOABS_2MIN_2003-SEP-25:00:03:00/
```

In this example the collection database referenced by the alias VNOABS_2MIN (= sum of all data files that have been created by performance collections started with the collection profile 2MIN on node VNOABS) is opened for shareable Read/Write access.

Example 2

This is an example how to attach a logical storage area.

```
DQL>ATTACH ALIAS VNOABS_2MIN DATE 19-SEP-2003;
DQL-I-ATTACH, successfully attached file /VNOABS_2MIN_2003-SEP-19:00:03:00/
DQL-I-ATTACH, successfully attached file /VNOABS_2MIN_2003-SEP-19:15:03:00/
```

In this example the logical storage of the collection database referenced by the alias VNOABS_2MIN containing the data files of 19-SEP-2003is opened for shareable Read/Write access. The logical storage area consists of two physical storage areas (data files)(VNOABS_2MIN_2003-SEP-19:00:03:00 and VNOABS_2MIN_2003-SEP-19:15:03:00).

## *ATTACH FILE*

This command opens a specific database file (physical storage area) for Read/Write access.

### Format

ATTACH 'FILE filename_alias' [EXCLUSIVE];

### Description

This command opens a specific database file (physical storage area) according to the access mode (Read/Write – Read Only) defined by the SET TRANSACTION command. The default access mode is Read Only. Once the command returns successfully the data file stays accessible for the runtime of the DQL$ session until you explicitly disconnect from the data file using the DEATTACH command.

The filename_alias parameter specifies the alias of the physical storage area. That filename alias can't be user-defined. DQL$ assigns these aliases when it starts up automatically. The file name aliases available are displayed when you apply the SHOW PHYSICAL STORAGE AREA command. The filename alias includes information about the database and logical storage area the physical storage area belongs to.

If you omit the EXCLUSIVE keyword the physical storage area is opened shareable. Using the EXCLUSIVE keyword the physical storage area is opened non-shareable.

**Note**

If you apply the EXCLUSIVE keyword the physical storage area is always opened for Read/Write access independently of the transaction type defined by the SET TRANSACTION command.

For further information about the distributed collection database please refer to chapter VSI PERFDAT distributed performance database.

### Example

```
DQL>ATTACH 'FILE VNOABS_2MIN_2003-SEP-18:20:43:00';
DQL-I-ATTACH, successfully attached file /VNOABS_2MIN_2003-SEP-18:20:43:00/
```

In this example the physical storage area of the collection database VNOABS_2MIN (node: VNOABS, collection profile: 2MIN) created on 18-SEP-200320:43:00 is opened for shareable Read/Write access.

## *CALCULATE (base form)*

This command selects element data for all statistics (data fields) of a metric defined by the query and calculates depending on the calculation option specified based on these data the arithmetic mean value, integral mean value, max value, standard deviation or all of them.

### Format

```
CALCULATE [STACKED] calculation_option
        OF statistics_itemlist
                FROM metrix_name
                        ALIAS alias_name [DATE] date
                                [ELEMENT] element_itemlist
                                        [WHERE] filter_itemlist
                                                [INTO] filename;
```

### Description

This command selects element data for all statistics (data fields) of a metric defined by the query and calculates depending on the calculation option specified based on these data the arithmetic mean value, integral mean value, max value, standard deviation or all of them.

DQL$ evaluates the clauses of the CALCULATE statement in the following order:

1. OF
2. FROM
3. ALIAS
4. DATE
5. ELEMENT
6. WHERE
7. INTO

The CALCULATE command first selects element data by executing a SELECT statement, inserts the result into a temporary result table and executes the calculation defined by the calculation option for each element and statistics stored in that result table. The SELECT statement has the format:

```
SELECT [STACKED] statistics_itemlist FROM metric_name
        ALIAS alias_name [DATE] date
                [ELEMENT] element_name [WHERE] filter_list;
```

The arguments and clauses of the SELECT statement are copied from the appropriate arguments and clauses in the CALCULATE command. For detailed information about the following arguments and clauses please refer to the SELECT command description:

- STACKED
- FROM
- ALIAS
- DATE
- ELEMENT
- WHERE

The CALCULATE statement does not support direct address mode to address the data fields (statistics_itemlist) to calculate (for information about supported data field (statistics) address modes please refer to the SELECT command description).

Depending if the STACKED argument is applied or not the CALCULATION query is stacked or un-stacked.

Since the stacked form of the CALCULATION statement selects element data using the stacked form of the SELECT statement the result table consists of one element per statistics that contains the stacked data of all elements that match the CALCULATE query filter defined by the ELEMENT and WHERE clause. The un-stacked form of the command selects the data using the un-stacked form of the SELECT statement. Thus, the result table contains as many elements per statistics as defined by the ELEMENT and WHERE clause.

With other words the un-stacked form of the CALCULATE statement creates an element selective report that contains the calculated values for each element and statistics defined. Since the input table of the stacked form of the command contains stacked element data, only one element that addresses these data exists. Thus, a stacked element report is created containing the calculated values for that single (stacked) element. A user-defined element name is assigned to that element. The stacked element name can be redefined by applying the DEFINE ELEMENT command in advance of the CALCULATE statement. The default name of that stacked element is STACKED.

Prerequisite:
The data files of the collection database / logical storage areas defined by the ALIAS and DATE clause have to be attached in advance using the ATTACH command.

Supported keywords for the calculation option are:

- ARITHMETIC_MEANVALUE
  The arithmetic mean value for each statistics defined by the OF clause is calculated ( $x_{avg} = \sum x_n / n$ ).
- INTEGRAL_MEANVALUE
  The integral mean value for each statistics defined by the OF clause is calculated ( $x_{avg} = \int x dt / t$ ).
- MAXIUMUM

The maximum value for each statistics defined by the OF clause is evaluated.

- STD
  The standard deviation for each statistics defined by the OF clause is calculated.
- ALL
  The arithmetic mean value, integral mean value, maximum value and the standard deviation for each statistics defined by the OF clause is calculated.

The calculation option keyword is mandatory.

The OF clause specifies the statistics to be selected from the source databases. Enter the statistics as a comma separated list. All statistics specified in the OF clause must exist in the metric defined by the FROM clause. Otherwise the command fails. To verify if the statistics are valid use the SHOW STATISTICS command. The OF clause is mandatory.

You can redirect the output of the query to a user definable CSV file if you apply the optional INTO clause. If you omit the INTO clause the result of the query is only displayed on screen.

## Examples

Example 1

```
DQL>CALC ALL OF ICPULOAD FROM PROCESS
cont>ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005 ELEMENT PERFDAT*;
Done: 0%...20%...100%

ALL STAT. CALCULATION,          METRIX: PROCESS, COLLECTION(S):
                                        ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005
                                PARAMETER: N/A

Statistics                |     Type          iCpuLoad
Elements                  |
--------------------------|-----------------------------------
PERFDAT                   |     (arithmetic)  0.303
PERFDAT                   |     (integral)    0.303
PERFDAT                   |     (max)         1.808
PERFDAT                   |     (std)         0.116
PERFDAT_ARCHIVE           |     (arithmetic)  0.000
PERFDAT_ARCHIVE           |     (integral)    0.000
PERFDAT_ARCHIVE           |     (max)         0.058
PERFDAT_ARCHIVE           |     (std)         0.002
PERFDAT_REPORT            |     (arithmetic)  27.100
PERFDAT_REPORT            |     (integral)    0.264
PERFDAT_REPORT            |     (max)         39.466
PERFDAT_REPORT            |     (std)         28.265
PERFDAT_SNMP              |     (arithmetic)  0.000
PERFDAT_SNMP              |     (integral)    0.000
PERFDAT_SNMP              |     (max)         0.008
PERFDAT_SNMP              |     (std)         0.001
PERFDAT_SNMP_0            |     (arithmetic)  0.014
```

```
PERFDAT_SNMP_0      |   (integral)    0.014
PERFDAT_SNMP_0      |   (max)         0.058
PERFDAT_SNMP_0      |   (std)         0.011
```

In this example the un-stacked form of the CALCULATE command is used to calculate the arithmetic mean value, integral mean value, standard deviation and maximum value of the statistics iCpuLoad (CPU load) for each process that match the element filter criteria PERFDAT*.Data source is the logical storage area that contains the performance data of node HOBEL from 30-AUG-2005. Since no time filter is defined by the WHERE clause the time range for the calculation is the whole day.

Example 2

```
DQL>CALC ALL OF ICPULOAD FROM PROCESS
cont>ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005 ELEMENT PERFDAT*
cont> WHERE TIME > 30-AUG-2005 10:00, TIME < 30-AUG-2005 12:00;
Done: 0%...20%...100%

ALL STAT. CALCULATION,      METRIX: PROCESS, COLLECTION(S):
                                ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005
                            PARAMETER:
                                TIME > 30-AUG-2005 10:00, TIME < 30-AUG-2005 12:00

Statistics Elements     |   Type          iCpuLoad
------------------------|-----------------------------------
PERFDAT                 |   (arithmetic)  0.305
PERFDAT                 |   (integral)    0.305
PERFDAT                 |   (max)         0.500
PERFDAT                 |   (std)         0.057
PERFDAT_ARCHIVE         |   (arithmetic)  0.000
PERFDAT_ARCHIVE         |   (integral)    0.000
PERFDAT_ARCHIVE         |   (max)         0.008
PERFDAT_ARCHIVE         |   (std)         0.001
PERFDAT_SNMP            |   (arithmetic)  0.000
PERFDAT_SNMP            |   (integral)    0.000
PERFDAT_SNMP            |   (max)         0.008
PERFDAT_SNMP            |   (std)         0.001
PERFDAT_SNMP_0          |   (arithmetic)  0.012
PERFDAT_SNMP_0          |   (integral)    0.013
PERFDAT_SNMP_0          |   (max)         0.033
PERFDAT_SNMP_0          |   (std)         0.009
```

This example demonstrates the use of the WHERE clause. The CALCULATE statement is almost identical to the statement in Example 1. The only difference is that the WHERE clause is present, that defines the time range for the calculation. In this case the time range is 30-AUG-200510:00 to 30-AUG-200512:00. If you compare the report created with the report in example 1 you can see that the process PERFDAT_REPORT is missing since this process did not exist during the specified time range.

Example 3

```
DQL>CALC STACKED ALL OF ICPULOAD FROM PROCESS
```

```
cont>ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005 ELEMENT PERFDAT*;
Done: 0%...20%...100%

ALL STAT. CALCULATION,      METRIX: PROCESS, COLLECTION(S):
                                    ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005
                            PARAMETER: N/A

        Statistics          |   Type          iCpuLoad
        Elements            |
        --------------------|-------------------------------
        STACKED             |   (arithmetic)  0.582
        STACKED             |   (integral)    0.581
        STACKED             |   (max)         37.604
        STACKED             |   (std)         2.791
```

In this example the stacked form of the CALCULATE command is used to calculate the arithmetic mean value, integral mean value, standard deviation and maximum value (ALL calculation option) of the stacked iCpuLoad (CPU load) statistics for a group of processes (all PERFDAT processes). The processes of interested are defined by the element filter criteria PERFDAT*. Data source is the logical storage area that contains the performance data of node BCSXTC from 30-AUG-2005. Since the data of all processes that match the element filter criterion are stacked before the calculation is done only one value per calculation type (arithmetic mean value, integral mean value, standard deviation, maximum value) is displayed. Since the data of several processes are stacked the values calculated can't be assigned to a single process (element name). Thus, a user-defined element name is displayed in the element column of the report. The stacked element name can be redefined by applying the DEFINE ELEMENT command in advance of the CALCULATE statement. If no stacked element name has been defined before the default stacked element name STACKED is displayed.

Example 4

```
DQL>DEFINE ELEMENT "PERF_CLUE";

DQL>CALC STACKED ALL OF ICPULOAD FROM PROCESS
cont>ALIAS BCSXTC_DEFAULT, HOBEL_DEFAULT DATE 30-AUG-2005
cont>ELEMENT PERFDAT*;
Done: 0%...20%...100%

ALL STAT. CALCULATION,      METRIX: PROCESS, COLLECTION(S):
                            ALIAS BCSXTC_DEFAULT, HOBEL_DEFAULTDATE 30-AUG-2005
                            PARAMETER: N/A

        Statistics          |   Type          iCpuLoad
        Elements            |
        --------------------|-------------------------------
        PERF_CLUE           |   (arithmetic)  0.676
        PERF_CLUE           |   (integral)    0.675
        PERF_CLUE           |   (max)         42.850
        PERF_CLUE           |   (std)         3.139
```

In this example the stacked form of the CALCULATE command is used to calculate the arithmetic mean value, integral mean value, standard deviation and maximum value (ALL calculation option) of the stacked iCpuLoad (CPU load) statistics for all processes that where active on the nodes BCSXTC and HOBEL on 30-AUG-2005 and match the filter criterion PERFDAT*. Since the data of all these processes are stacked before the calculation is done only one value per calculation type (arithmetic mean value, integral mean value, standard deviation, maximum value) is displayed. Since the data of several processes are stacked the values calculated can't be assigned to a single process (element name). Thus, a user-defined element name is displayed in the element column of the report. The stacked element name can be redefined by applying the DEFINE ELEMENT command in advance of the CALCULATE statement. In this example the DEFINE ELEMENT command has been used to assign the name PERF_CLUE.

## *CALCULATE (deviation report)*

This command creates a deviation analysis report for 1…n statistics and 1…m elements of a metric. The element data of1...n statistics of a metric are read from two different logical storage areas (= data of different days). These data are averaged and compared to each other. The percentage the source data average differs from the reference data average is displayed for each statistics. The deviation analysis can be done integral or arithmetic based.

### Format

```
CALCULATE [STACKED] calculation_option DEVIATION
        OF statistics_itemlist
                FROM metrix_name
                        [ELEMENT] element_itemlist
SOURCE (ALIAS src_alias_name DATE src_date [WHERE] src_fiter_itemlist)
REFERENCE (ALIAS ref_alias_name DATE ref_date [WHERE] ref_fiter_itemlist)
                                [INTO] filename;
```

### Description

This command creates a deviation analysis report for 1…n statistics and 1…m elements of a metric. Element data of 1...n statistics of a metric are read from two different logical storage areas (= data of different days). These data are averaged and compared to each other. The percentage the source data average differs from the reference data average is displayed for each statistics. The deviation analysis can be done integral or arithmetic based.

DQL$ evaluates the clauses of the CALCULATE statement in the following order:

1. OF
2. FROM
3. ELEMENT
4. SOURCE
   - ALIAS
   - DATE
   - WHERE
5. REFERENCE
   - ALIAS
   - DATE
   - WHERE
6. INTO

The CALCULATE command first selects element data from the source logical storage areas defined by the SOURCE clause and element data from the reference logical storage areas defined by the REFERENCE clause by executing appropriate SELECT statement. The result of both SELECT queries are inserted

into temporary result tables and according to the calculation option the arithmetic or integral mean value is calculated for each statistics and element stored in these result tables. The ratio between the source average values and the reference average value for each statistics and element available in both result tables are displayed. The source SELECT statement has the format:

SELECT [STACKED] statistics_itemlist FROM metric_name
ALIAS src_alias_name DATEsrc_date
[ELEMENT] element_name [WHERE] src_filter_list

The reference SELECT statement has the format:

SELECT [STACKED] statistics_itemlist FROM metric_name
ALIAS ref_alias_name DATEref_date
[ELEMENT] element_name [WHERE] ref_filter_list

The arguments and clauses of the SELECT statement are copied from the appropriate arguments and clauses in the CALCULATE command. For detailed information about the following arguments and clauses please refer to the SELECT command description

- STACKED
- FROM
- ALIAS
- DATE
- ELEMENT
- WHERE

The CALCULATE statement does not support direct address mode to address the data fields (statistics_itemlist) to calculate (for information about supported data field (statistics) address modes please refer to the SELECT command description).
Depending if the STACKED argument is applied or not the CALCULATION query is stacked or un-stacked.

Since the stacked form of the CALCULATION statement selects element data using the stacked form of the SELECT statement the result tables consist of one element per statistics that contains the stacked data of all elements that match the CALCULATE query filters defined by the ELEMENT and (source or reference) WHERE clause. The un-stacked form of the command selects the data using the un-stacked form of the SELECT statement. Thus, the result tables contain as many elements per statistics as defined by the ELEMENT and (source or reference) WHERE clause.

With other words the un-stacked form of the CALCULATE statement creates an element deviation report that contains the calculated values for each element and statistics defined. Since the input tables of the stacked form of the command contains stacked element data, only one element that addresses these data exists. Thus, a stacked element report is created containing the

calculated values for that single (stacked) element. A user-defined element name is assigned to that element. The stacked element name can be redefined by applying the DEFINE ELEMENT command in advance of the CALCULATE statement. The default name of that stacked element is STACKED.

Prerequisite:
The data files of the logical storage areas defined by the ALIAS and DATE clause within the SOURCE and REFERENCE clause have to be attached in advance using the ATTACH command.

Supported keywords for the calculation option are:

- ARITHMETIC_MEANVALUE
  The average values to compare are calculated arithmetical ( $x_{avg} = \sum x_n / n$ ).
- INTEGRAL_MEANVALUE
  The average values to compare are calculated integral ( $x_{avg} = \int x dt / t$ ).

The calculation option keyword is mandatory.

The OF clause specifies the statistics to be selected from the source and reference logical storage areas defined by the SOURCE and REFERENCE clause. Enter the statistics as a comma separated list. All statistics specified in the OF clause must exist in the metric defined by the FROM clause. Otherwise the command fails. To verify if the statistics are valid use the SHOW STATISTICS command. The OF clause is mandatory.

The SOURCE clause specifies the source logical storage areas and optional filter criteria that apply to the source SELECT query. The REFERENCE clause specifies the reference logical storage areas and optional filter criteria that apply to the reference SELECT query. Both clauses start with an open bracket (“(“) and ends with a close bracket (“)”).

In contrast to the SELECT statement the DATE clause is mandatory within the SOURCE and the REFERNCE clause. Enter all the days of interest within the DATE clause as a comma separated date list. Use OpenVMS date format to define the days of interest.

You can redirect the output of the query to a user definable CSV file if you apply the optional INTO clause. If you omit the INTO clause the result of the query is only displayed on screen.

## Examples

Example 1

```
DQL> CALC INTEG DEVIATION OF ICPULOAD FROM PROCESS ELEMENT PERFDAT*
```

```
cont> SOURCE (ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005)
cont> REFERENCE (ALIAS BCSXTC_DEFAULT DATE 20-APR-2005);

 Integral deviation,      Metrix: PROCESS
                          Source Parameter: ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005
                          Reference Parameter: ALIAS BCSXTC_DEFAULT DATE 20-APR-2005
                          Element list: PERFDAT*

Done: 0%...20%...100%

Done: 0%...33%...100%

          Statistics                |          iCpuLoad
          Elements                  |
          ----------------------------------|----------------------
          PERFDAT                   |          -0.937
          PERFDAT_ARCHIVE           |          -26.666
          PERFDAT_REPORT            |          -11.723
```

This example demonstrates the use of the un-stacked form of the statement. The deviation of the average CPU load caused by all PERFDAT processes (see ELEMENT clause PERFDAT*) active on node BCSXTC on two different days is calculated. The source date is 30-AUG-2005. The reference date is 20-APR-2005. Since element data are selected un-stacked the calculation is done un-stacked for each statistics and process that were active on node BCSXTC during both days. Processes that match the source element filter criteria but were not active on one the reference day and vice versa are not listed in the report. As you can see the average CPU load caused by the process PERFDAT_REPORT was 26.67 % less on 30-AUG-2005 than on 20-APR-2005.

Example 2

```
DQL> CALC STACKED INTEG DEVIATION OF ICPULOAD FROM PROCESS
cont> ELEMENT PERFDAT*
cont> SOURCE (ALIAS BCSXTC_DEFAULT, HOBEL_DEFAULT DATE 30-AUG-2005)
cont> REFERENCE (ALIAS BCSXTC_DEFAULT, HOBEL_DEFAULT DATE 20-APR-2005);

 Integral deviation,      Metrix: PROCESS
                          Source Parameter: ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005
                          Reference Parameter: ALIAS BCSXTC_DEFAULT DATE 20-APR-2005
                          Element list: PERFDAT*

Done: 0%...100%

Done: 0%...100%

          Statistics                |          iCpuLoad
          Elements                  |
          ----------------------------------|----------------------
          STACKED                   |          -3.960
```

This example demonstrates the use of the stacked form of the statement. The deviation of the stacked average CPU load caused by all PERFDAT processes (see ELEMENT clause PERFDAT*) active on the nodes BCSXTC and HOBEL on two different days is calculated. The source day is 30-AUG-2005. The reference day is 20-APR-2005. Since the data of all these processes are stacked before the

calculation is done only one value per calculation type (arithmetic mean value, integral mean value, standard deviation, maximum value) is displayed. Since the data of several processes are stacked the deviation calculated can't be assigned to a single process. Thus, a user-defined element name is displayed in the element column of the report. Since the DEFINE ELEMENT command has not been applied in advance of the CALCULATE statement the default stacked element name STACKED is displayed. As you can see the average CPU load caused by all PERFDAT processes active on both nodes BCSXTC and HOBEL consumed 3.96 % less CPU power on 30-AUG-2005 than on 20-APR-2005.

## *CHECK FILE MAP*

This command checks if the CSV files addressed by the CSV mapping entries in the CSV mapping database are valid.

### Format

CHECK FILE MAP;

### Description

This command checks if the CSV files addressed by the entries in the CSV mapping database are valid. This is done for each CSV mapping entry in the CSV mapping database. If an invalid record is found in a CSV file mapped the file name and the line number of the invalid record is displayed.

For detailed information about CSV file mapping please see the MAP command description.

## *CONVERT ALIAS*

Converts the headers of all performance collection data files (physical storage areas) that are member of a collection database / logical storage area created by an older PERFDAT version than actually used to new format.

### Format

CONVERT ALIAS alias_name [DATE date];

### Description

Converts the headers of all performance collection data files (physical storage areas) that are member of a collection database created by an older PERFDAT version than actually used to new format.

The ALIAS clause specifies the alias of the collection database to convert. That database alias can't be user-defined. DQL$ assigns these aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. The aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

If you want to convert all physical storage areas of a logical storage area (all data files that have been created on the same day) the DATE clause is mandatory. Use OpenVMS date format to define the day of interest. If you omit the DATE clause all data files (physical storage areas) of the collection database are converted.

You have to convert data files to new format if you want to insert new records into exiting data files after upgrading PERFDAT to new version manually. Read only data files do not have to be converted, since data files are read accessible independently of the PERFDAT version in use. Trend and capacity report data files are converted automatically by the upgrade and installation procedure of PERFDAT.

In order to convert collection databases/logical storage areas they don't have to be attached in advance. This is done implicitly.

### Example

```
DQL> CONVERT ALIAS HOBEL_DEFAULT DATE 30-AUG-2005;
DQL-I-Convert, successfully converted physical storage area /HOBEL_DEFAULT_2005-08-30:00:03:00:1/
DQL-I-Convert, successfully converted physical storage area /HOBEL_DEFAULT_2005-08-30:10:27:00:1/
```

This example shows how to convert the logical storage area 30-AUG-2005 of the collection database HOBEL_DEFAULT to new format. The logical storage area consists of two physical storage areas – one data file was created at 30-AUG-200500:03 and the other on 30-AUG-2005 0:27.

# *CONVERT FILE*

Converts the header of a performance collection data files (physical storage areas) created by an older PERFDAT version than actually used to new format.

## Format

CONVERT 'FILE filename_alias';

## Description

Converts the header of a performance collection data files (physical storage areas) created by an older PERFDAT version than actually used to new format.

The filename_alias parameter specifies the alias of the physical storage area. The filename alias can't be user-defined. DQL$ assigns these aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW PHYSICAL STORAGE AREA command. The filename alias includes information about the database and logical storage area that physical storage area belongs to.

You have to convert a data file to new format if you want to insert new records into the metrics of the data files after upgrading PERFDAT to new version manually. Read only data files do not have to be converted, since data files are read accessible independently of the PERFDAT version in use.

## Example

```
DQL> CONVERT 'FILE HOBEL_DEFAULT_2005-08-30:00:03:00:1';
DQL-I-Convert, successfully converted physical storage area /HOBEL_DEFAULT_2005-08-30:00:03:00:1/
```

This example shows how to convert the physical storage area (data file)

HOBEL_DEFAULT_2005-08-30:00:03:00:1

to new format.

## *CORRELATE*

This command calculates the correlation of the data of 1...n elements of a source statistics of a metric with the data of 1…n elements of a reference statistics of another (or the same) metric within a logical storage area.

### Format

```
CORRELATE
    (statistics_name FROM metrix_name [ELEMENT] element_itemlist)
        WITH
            (statistics_name FROM metrix_name [ELEMENT] element_itemlist)
                ALIAS alias_name DATE date
                        [WHERE] fiter_itemlist
                                [INTO] filename;
```

### Description

This command calculates the correlation of the data of 1...n elements of a source statistics of a metric with the data of 1…n elements of a reference statistics of another (or the same) metric within a logical storage area.

Prerequisite:
The data files of the logical storage areas defined by the ALIAS and DATE clause within the CORRELATE and WITH clause have to be attached in advance using the ATTACH command.

DQL$ evaluates the clauses of the CORRELATE statement in the following order:

1. CORRELATE
    - FROM
    - ELEMENT
2. WITH
    - FROM
    - ELEMENT
3. ALIAS
4. DATE
5. WHERE
6. INTO

The CORRELATE clause specifies the statistics and the elements of the metric that shall be correlated with the statistics and elements of the metric defined in the WITH clause. The source statistics of all elements defined by the CORRELATE clause is correlated with the reference statistics of all element in the WITH clause. Both clauses start with an open bracket (“(“) and ends with a close bracket (“)”).

A source statistics for an element is valid if the statistics is member of the metric defined by the FROM clause within the CORRELATE clause. A reference statistics is valid if the statistics is member of the metric defined by the FROM clause within the WITH clause. If either the source or a reference statistics does not exist the command fails. To verify if the statistics are valid use the SHOW STATISTICS command.

You can enter only one statistics (statistics_name) within the CORRELATE and WITH clause.

The FROM clause in the CORRELATE and WITH clause specify the metric the statistics defined belong to. The metrics defined must exist in the logical storage area defined by the ALIAS and DATE clause. The FROM clause is mandatory.

The optional ELEMENT clause in the CORRELATE and WITH clause can be used to filter the elements to be included in the correlation report. Enter the element that shall be included in the correlation report as a comma (,), or OR sign (|) separated list. Elements that should be excluded from the CORRELATE query have to be preceded with the '!=' or '<>' tag in the comma separated list of the ELEMENT clause. VSI PERFDAT V3.0and higher versions provide full wildcard support. Asterisk (*) and percent sign (%) wildcard characters can be placed anywhere within each string of the comma separated element list. If you enter quotation marks at the beginning and the end of an element string the string is taken literally (no wildcard operation performed on that string even if it contains wildcard characters).

The ALIAS and DATE clause specifies the logical storage area the correlation query applies to. The ALIAS clause contains the collection database alias and the DATE clause the day of interest.

The database alias can't be user-defined. DQL$ assigns the aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

The optional WHERE clause can be applied to define additional filter criteria. Enter the filter criteria as a comma separated list. A single filter criterion consists of a valid statistics name, an operator and a comparison value. A statistics name is valid if it is member of both metrics defined (FROM clause) within the CORRELATE and the WITH clause. Since the TIME field (statistics) is the only one that is common to all metrics, the TIME statistics is the only one that can be entered in the WHERE clause to define the time period for correlation processing. Valid operators are

- < less than

- <= less than or equal
- = equal
- >= greater than or equal
- > greater than
- <> not equal
- != not equal

If the operator applied is <, <=, => or >only a single comparison value can be entered. If the operator applied is =, != or <> you can enter a comparison value list.

Example:
If you want to limit the query to the time period 25-AUG-200301:00 to 25-AUG-200304:00 enter:

```
WHERE TIME >= 25-AUG-200301:00, TIME <= 25-AUG-200304:00
```

You can redirect the output of the query to a user definable CSV file if you apply the optional INTO clause. If you omit the INTO clause the result of the query is only displayed on screen.

## Example

```
DQL> CORRELATE (iCpuLoad FROM PROCESS ELEMENT *)
cont> WITH (iCpuLoad FROM SYSTEM ELEMENT *)
cont> ALIAS HOBEL_DEFAULT DATE 30-AUG-2005
cont> WHERE TIME >= 30-AUG-200502:00, TIME <= 30-AUG-200502:30;
```

| Elements | OPENVMS |
|---|---|
| ACME_SERVER | 0.00 ( 100.00) |
| AUDIT_SERVER | 0.00 ( 100.00) |
| DNS$ADVER | 0.00 ( 100.00) |
| DQL$SRV_BG2148 | 98.70 ( 100.00) |
| DTSS$SERVICE | 0.00 ( 100.00) |
| ERRFMT | 3.77 ( 100.00) |
| FASTPATH_SERVER | 0.00 ( 100.00) |
| IPCACP | 0.00 ( 100.00) |
| JOB_CONTROL | 6.64 ( 100.00) |
| LANACP | 0.00 ( 100.00) |
| LATACP | 47.59 ( 100.00) |
| LES$ACP_V30 | 0.00 ( 100.00) |
| NET$ACP | 0.00 ( 100.00) |
| NET$EVD | 0.00 ( 100.00) |
| NET$MOP | 0.00 ( 100.00) |
| OPCOM | -20.33 ( 100.00) |
| PERFDAT | 99.29 ( 100.00) |
| PERFDAT_ARCHIVE | 6.64 ( 100.00) |
| PERFDAT_REPORT | 82.54 ( 100.00) |
| QUEUE_MANAGER | 0.00 ( 100.00) |
| REMACP | 6.64 ( 100.00) |
| SECURITY_SERVER | 80.95 ( 100.00) |
| SHADOW_SERVER | 0.00 ( 100.00) |
| SWAPPER | 0.00 ( 100.00) |
| SYSTEM | 0.00 ( 100.00) |
| TCPIP$INETACP | 0.00 ( 100.00) |
| TCPIP$PWIP_ACP | 0.00 ( 100.00) |

```
TP_SERVER                |                11.81 (  100.00)
```

In this example the CPU load (statistics iCpuLoad of metric PROCESS) data of all processes (see ELEMENT clause within the CORRELATE clause) is correlated with the system CPU load (statistics iCpuLoad of metric SYSTEM – the SYSTEM metric contains only on element). The source logical storage area is 30-AUG-2005 of the collection database HOBEL_DEFAULT (data collected on 30-AUG-2005 on node HOBEL by a data collection stated with the DEFAULT profile). The time range for correlating the data was limited to 30-AUG-200502:00 to 30-AUG-200502:30 using the WHERE clause.

You can see from the output above that the processes PERFDAT and DQL$SRV_BG2148 correlates the most with the overall CPU load on node HOBEL between 30-AUG-200502:00 and 30-AUG-200502:30. The numbers in the brackets displays the percentage of source element data used for the correlation. E.g. LATACP – 100% of the CPU load data of the LATACP process are used for correlating with the system CPU load data.

Be careful interpreting the result of the CORRELATION query. The correlation coefficient is a measure for the equality of the curve shapes but no measure of the amplitude of the curves. With other words you cannot conclude that if e.g. the CPU load caused by process PERFDAT correlates the most with the overall system CPU load that PERFDAT was a top consumer of the CPU resources. This interpretation is wrong. To get the top consumers you have to apply the SHOW ELEMENT command:

```
DQL> SHOW ELEMENT * FROM PROCESS
cont> ORDERD BY ICPULOAD DESCENDING
cont> ALIAS HOBEL_DEFAULT DATE 30-AUG-2005
cont> WHERE TIME >= 30-AUG-200502:00, TIME <= 30-AUG-200502:30;

Done:
0%...7%...14%...21%...28%...35%...42%...50%...57%...64%...71%...78%...85%...92%...100%

 ELEMENT LIST of storage area

Element                                  referes to                    Ratio
-------------------------------------------------------------------------------------------------------
DQL$SRV_BG2148                                                         75.6
PERFDAT_REPORT                                                         20.8
PERFDAT                                                                3.0
TP_SERVER                                                              0.2
.
.
```

As you can see PERFDAT caused only 3% of the overall CPU load although the curve shape of the CPU load data of process PERFDAT correlates the most with the system CPU load.

## *CREATE GRAPH*

This command selects data from collection databases and creates PNG formatted line graphs that can be viewed directly with your WEB browser.

### Format

```
CREATE GRAPH [STACKED] statistics_itemlist
        FROM metric_name
                ALIAS alias_name [DATE] date
                        [ELEMENT] element_name
                                [WHERE] filter_list
                                        [INTO] directory
                                        [NAME] graph_name
                                  [STACKED_OVERLAY | SINGLE_SCALED];
```

### Description

This command selects data from collection databases and creates PNG formatted line graphs that can be viewed directly with your WEB browser. This command facilitates automated WEB based graphing and data analysis.

The CREATE GRAPH command first selects data by executing a SELECT statement extracted from the command input. This SELECT statement has the format:

```
SELECT [STACKED] statistics_itemlist [FROM] metric_name
        ALIAS alias_name [DATE] date
                [ELEMENT] element_name [WHERE] filter_list;
```

The arguments used in the clauses of the SELECT statement are copied from the appropriate arguments and clauses in the CREATE GRAPH command. For detailed information about the following arguments and clauses please refer to the SELECT command description:

- STACKED
- FROM
- ALIAS
- DATE
- ELEMENT
- WHERE

Both address modes (base address and direct address mode) to address data fields (statistics) are supported (for more information about address mode please refer to the SELECT command description). As with the EXPORT statement statistics of different metrics can be addressed at once (see EXPORT command description for more information).

The directory to store the graphs can be defined by use of the DIRECTORY clause. If the DIRECTORY clause is omitted the PNG files are stored in PERFDAT$GRAPH.

The data of all statistics defined within the CREATE GRAPH query are plotted onto one graph automatically. If the user wants to plot the data of just one statistic the CREATE GRAPH query has to contain this statistic only.

If the user applies the STACKED_OVERLAY keyword the selected data (statistics) are plotted in stacked mode. If the Keyword is omitted the data is plotted in un-stacked mode.

If the SINGLE_SCALED keyword is applied the selected data (statistics) plotted onto a single graph can be individually scaled, so that data of different orders of magnitude can be plotted on the same graph.

The SINGLE_SCALED and STACKED_OVERLAY keywords are mutual exclusive.

The maximum number of statistics that can be plotted into one graph is 16.

The NAME clause defines the name of the PNG file. If the NAME clause is omitted a default name is used.

The caption of the graph can be redefined by applying the DEFINE HEADER command in advance of this statement.

The user can customize the layout of the resulting graph via a configuration file. The default configuration file is:

- PERFDAT$CFG:PERFDAT_CSV2PNG.CFG.

Table 4.1 PNG graph customization parameters

| Parameter | Values | Description |
|---|---|---|
| AutoScaleLandscape | TRUE / FALSE | TRUE: the PNG file is auto-sized to A4 Landscape. AutoScaleLandscape takes precedence over AutoScalePortrait |
| AutoScalePortrait | TRUE / FALSE | TRUE: the PNG file is auto-sized to A4 Portrait |
| GraphWidth | Integer value [pixel] | Width of the Graph drawing area. If either AutoScaleLandscape or AutoScalePortrait is set this parameter is ignored. |
| GraphHeight | Integer value [pixel] | Height of the Graph drawing area. If either AutoScaleLandscape or AutoScalePortrait is set this parameter is ignored |
| XMajorTicks | • Auto | Major ticks count of the X (time) axis |

| | | |
|---|---|---|
| | • Integer Value | Auto: The time range displayed and the major X axis ticks count is calculated based on the selected time range in the CREATE GRAPH query. Due to the algorithm used it may happen that the displayed time range is greater than the time range defined in the query.<br>Integer Value: Manual selected major X axis ticks count. In addition the time range displayed is exactly the same as defined by the CREATE GRAPH query. |
| FillGraphArea | • FALSE<br>• TRUE | FALSE: line graphs are created<br>TRUE: the area under the lines plotted for each statistics selected by the CREATE GRAPH command are filled with the selected line colors |
| YScaleMin | • Auto<br>• Integer Value | Min. value of the Y-axis<br>Auto: Ymin is calculated based on the data to display.<br>Integer Value: Ymin is set to the value applied |
| YScaleMax | • Auto<br>• Integer Value | Max. value of the Y-axis<br>Auto: Ymax is calculated based on the data to display.<br>Integer Value: Ymax is set to the value applied |
| YxMin (x = 1..16) | • Auto<br>• Integer Value | Min. value of the Y-axis for the statistics listed at position x in the CREATE GRAPH command if the SINGLE_SCALED key word is applied and the configuration paramters YscaleMin and YScaleMax both have the keyword Auto assigned.<br>Auto: Value Ymin for the statistic listed at position x in the CREATE GRAPH query is calculated based on the data to display.<br>Integer Value: Value Ymin for the statistics listed at position x in the CREATE GRAPH query is set to the value applied |
| YxMax (x = 1..16) | • Auto<br>• Integer Value | Max. value of the Y-axis for the statistics listed at position x in the CREATE GRAPH command if the SINGLE_SCALED key word is applied and the configuration parameters YscaleMin and YScaleMax both have the keyword Auto assigned.<br>Auto: Value Ymax for the statistic listed at position x in the CREATE GRAPH query is calculated based on the data to display. |

| | | |
|---|---|---|
| | | Integer Value: Value Ymax for the statistics listed at position x in the CREATE GRAPH query is set to the value applied |
| LegendSorted | • FALSE<br>• TRUE | FALSE: statistics in the legend of the graph are listed in the same order as they are listed in the CREATE GRAPH command<br>TRUE: statistics in the legend of the graph are listed in order of their average values in the selected range. |
| LabelFont | • gdTiny<br>• gdSmall<br>• gdMediumBold<br>• gdLarge<br>• gdGiant | Font of the X-, and Y-labels of the PNG formatted graph:<br>• gdTiny approx. 8 Pkt<br>• gdSmall approx. 12 Pkt<br>• gdMediumBold approx. 13 Pkt, Bold<br>• gdLarge approx. 16 Pkt<br>• gdGiant approx. 16 Pkt, Bold |
| LegendFont | • gdTiny<br>• gdSmall<br>• gdMediumBold<br>• gdLarge<br>• gdGiant | Font of the legend of the PNG formatted graph:<br>• gdTiny approx. 8 Pkt<br>• gdSmall approx. 12 Pkt<br>• gdMediumBold approx. 13 Pkt, Bold<br>• gdLarge approx. 16 Pkt<br>• gdGiant approx. 16 Pkt, Bold |
| HeaderFont | • gdTiny<br>• gdSmall<br>• gdMediumBold<br>• gdLarge<br>• gdGiant | Font of the header of the PNG formatted graph:<br>• gdTiny approx. 8 Pkt<br>• gdSmall approx. 12 Pkt<br>• gdMediumBold approx. 13 Pkt, Bold<br>• gdLarge approx. 16 Pkt<br>• gdGiant approx. 16 Pkt, Bold |
| GridEnable | TRUE / FALSE | TRUE: Grid will be displayed<br>FALSE: Now Grid will be displayed |
| GridColor | • gdWhite<br>• gdBlack<br>• gdRed | Grid Color. You can select one of the predefined colors listed below: |

| | | |
|---|---|---|
| | • gdBlue<br>• gdGreen<br>• gdYellow<br>• gdGray | • gdWhite = white<br>• gdBlack = black<br>• gdRed = red<br>• gdBlue = blue<br>• gdGreen = green<br>• gdYellow = yellow<br>• gdGray = gray |
| ImageBackgroundColor | • gdWhite<br>• gdBlack<br>• gdRed<br>• gdBlue<br>• gdGreen<br>• gdYellow<br>• gdGray | Background Color of the PNG image. You can select one of the predefined colors listed below:<br>• gdWhite = white<br>• gdBlack = black<br>• gdRed = red<br>• gdBlue = blue<br>• gdGreen = green<br>• gdYellow = yellow<br>• gdGray = gray |
| GraphBorderColor | • gdWhite<br>• gdBlack<br>• gdRed<br>• gdBlue<br>• gdGreen<br>• gdYellow<br>• gdGray | Graph border color. You can select one of the predefined colors listed below:<br>• gdWhite = white<br>• gdBlack = black<br>• gdRed = red<br>• gdBlue = blue<br>• gdGreen = green<br>• gdYellow = yellow<br>• gdGray = gray |
| HeaderColor | • gdWhite<br>• gdBlack<br>• gdRed<br>• gdBlue<br>• gdGreen<br>• gdYellow<br>• gdGray | Color of the header string. You can select one of the predefined colors listed below:<br>• gdWhite = white<br>• gdBlack = black<br>• gdRed = red<br>• gdBlue = blue<br>• gdGreen = green<br>• gdYellow = yellow<br>• gdGray = gray |
| LineColorX (X…1 to 16) | Any valid RGB triple | Up to 16 line graphs can be displayed within a single PNG image. The color of these line graphs can be freely defined by assigning the appropriate RGB-triple to these parameters. LineColor1 refers the color of the 1. line graph, LineColor2 refers the color of the 2. line graph and so on. |

The recommended way to customize the graphs created by the CREATE GRAPH command is to copy the default configuration file, modify the configuration parameters therein and apply the DEFINE GRAPH_CFG command in advance of the CREATE GRAPH command to advise the DQL$ utility to use the configuration file addressed by the DEFINE GRAPH_CFG command.

Do not change the default configuration file as the default configuration file may be replaced by future VSI PERFDAT releases. Hence, any changes in the parameters in the default configuration file may be lost after upgrading VSI PERFDAT.

## Control Logical

**PERFDAT$SCRATCH**

During command execution temporary CSV files are created. The default directory to store these temporary CSV files is PERFDAT$GRAPH. If the user who executes the CREATE GRAPH command is not owner of the PERFDAT$GRAPH directory audit alerts are triggered.

To avoid such audit alerts the directory to store the temporary CSV files can be user-defined. If the logical PERFDAT$SCRATCH exists and if it refers a valid directory all temporary CSV files created by the CREATE GRAPH command are stored in this directory.

The logical PERFDAT$SCRATCH has to be defined system-wide:

```
$ DEFINE/SYSTEM PERFDAT$SCRATCH directory
```

## Example

Example 1

Attach the logical storage area:

```
DQL> ATTACHALIAS WSSPQ_DEFAULT DATE 29-APR-2006;
DQL-I-ATTACH, successfully attached file /WSSPQ_DEFAULT_2006-04-29:00:00:30:1/
```

Define the graph caption:

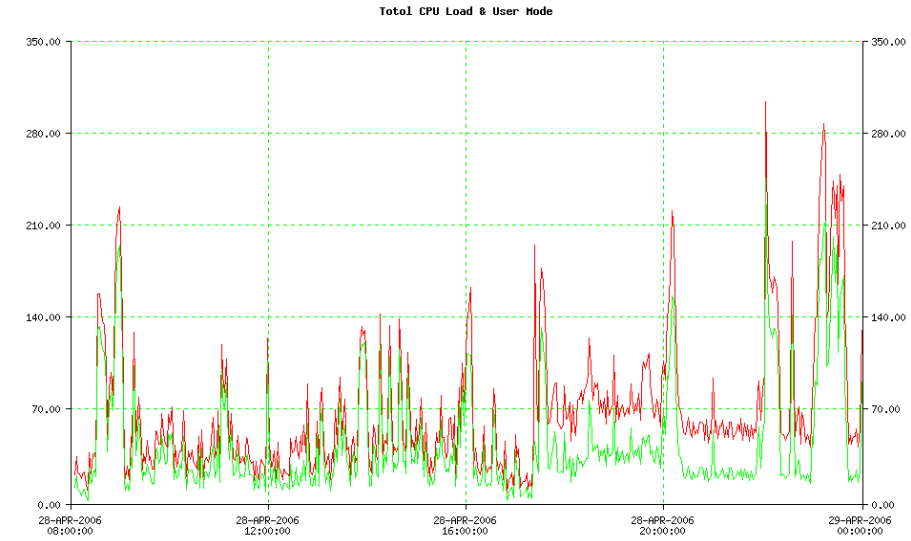
```
DQL> DEFINE HEADER “Total CPU Load & User Mode”;
```

Create Graph:

```
DQL> CREATE GRAPH iCpuLoad, iUser FROM SYSTEM
cont> ALIAS WSSPQ_DEFAULT DATE 29-APR-2006
cont> INTO $1$DKB100:[TEST_GRAPH] NAME CPUMODE;

DQL-I-GRAPH, Graphs created
        PNG file(s):                        CPUMODE.PNG
        Files are moved to directory:       $1$DKB100:[TEST_GRAPH]
```

In this example line graphs for the overall CPU load und the user mode CPU load are plotted onto one graph. The PNG file is created in the directory $1$DKB100:[TEST_GRAPH]. The name of the PNG file is user-defined since the

NAME clause is present. To view the graph access the CPUMODE.PNG file using your web-browser.




| Avg. | Min. | Max. | Stats | Unit | Metrix | System | Element |
|---|---|---|---|---|---|---|---|
| 69.98 | 8.60 | 303.86 | iCpuLoad | [%] | SYSTEM | WSSPQ | OPENVMS |
| 45.19 | 2.52 | 245.98 | iUser | [%] | SYSTEM | WSSPQ | OPENVMS |

Example 2

This example demonstrates the use of the un-stacked form of the CREATE GRAPH query using the direct address mode (see EXPORT command description) to select data from different sources (collection databases / logical storage areas defined by the ALIAS and DATE clause) and the selected statistics and metrics do not exist in all collection data files.

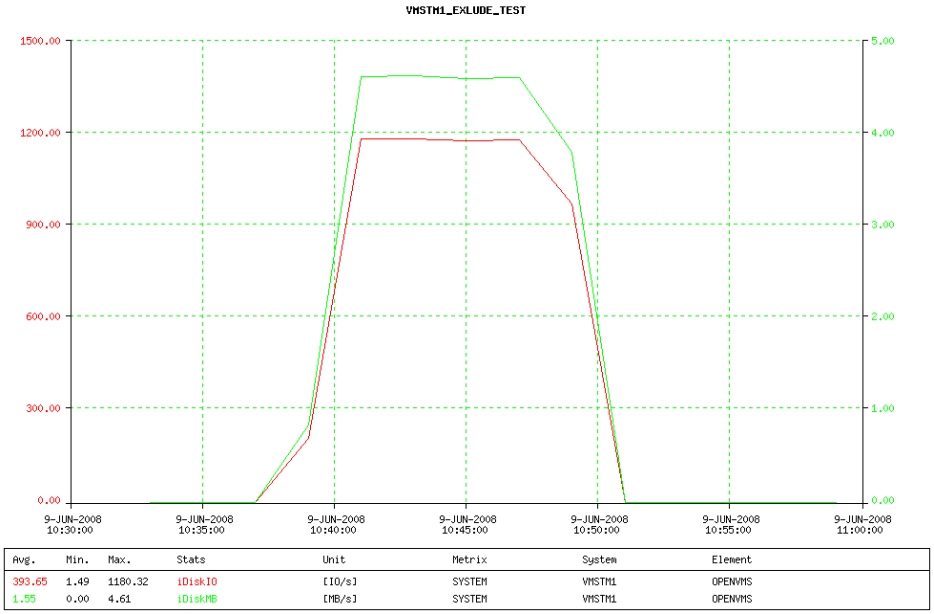
We want to create a graph that displays the I/O rate of all FC-adapters of the system VMSTM1 and the throughput of the FC-switches these adapters are connected. VMSTM1 has two FC- adapters FGA and FGB. FGA is connected to Port_1. The user-defined name of the graph is IOPATH.PNG.

```
DQL> ATTACH ALIAS VMSTM1_DEFAULT DATE 26-NOV-2007;
DQL> ATTACH ALIAS FC-SWITCH1_DEFAULT DATE 26-NOV-2007;
DQL> ATTACH ALIAS FC-SWITCH2_DEFAULT DATE 26-NOV-2007;

DQL> DEFINE HEADER “VMSTM1 FC-I/O rate & Switch throughput”

DQL> CREATE GRAPH IOPATHES.iOpCnt, PORT.TotWordsALIAS
cont> VMSTM1_DEFAULT, FC-SWITCH1_DEFAULT, FC-SWITCH2_DEFAULT
cont> DATE 26-NOV-2007 ELEMENT FG* | PORT_0
cont> NAME IOPATH;
```

Example 3:

In the following example the disk throughput (iDiskMB) and the disk I/O rate (iDiskIO) of the SYSTEM metric are plotted onto the same graph once with the SINGLE_SCALED applied.

```
DQL> ATTACH ALIAS VMSTM1_DEFAULT DATE 9-JUN-2008;
DQL> CREATE GRAPH iDiskIO, iDiskMB FROM SYSTEM ALIAS
cont> VMSTM1_DEFAULTDATE 9-JUN-2008
cont> WHERE TIME > 9-JUN-2008 10:30, TIME < 9-JUN-2008 11:00
cont> NAME SAME_SCALE SINGLE_SCALED;

DQL-I-GRAPH, Graphs created
        PNG file(s):                    SINGLE_SCALED.PNG
        Files are moved to directory:   PERFDAT$GRAPH
```




The disk throughput ranges from 0 to 4.6 Mbytes/sec. The disk I/O rate ranges from 0 to 1180 I/Os per second. Thus, without applying the SINGLE_SCALE keyword the disk throughput would be almost invisible.

## *CREATE METRIX*

This command creates a metric (table) in a physical storage area.

### Format

```
CREATE METRIX metric_name
                IN STORAGE AREA filename_alias
                                [FROM] descriptor_file ;
```

### Description

This command creates a metric (table) in a physical storage area.

The metric_name specifies the name of the new metric (table) to create in the physical storage area defined by the IN STORAGE AREA clause. If the metric already exists in the physical storage area defined the command fails.

Each metric (table) contains a record descriptor of the records stored in the metric. The optional DESCRIBED BY clause specifies the descriptor file containing the record descriptor valid for the newly created metric. This clause can be omitted if the metric name applied matches one of the OpenVMS metrics

- System
- CPU
- Process
- User
- Image
- Account
- Device
- Device.IOSize
- Device.File
- Device.Process
- Device.Process.File
- Device.Capacity
- Device.Path
- IOPathes
- XFCVolume
- XFCVolume.IOSize
- XFCVolume.File
- XFCVolume.File.IOSize
- LANAdapter
- LANAdapter.Device
- LANProtocol
- SCSPort

- SCSPort.VC
- SCSPort.VC.Channel

In this case the record descriptor is fetched from the PERFDAT configuration database.

Any valid record descriptor has to contain at least one statistics (data field) called TIME that contains the timestamp of the records stored in the metric. The maximum number of statistics (data fields) defined by a record descriptor is 200. If record descriptor exceeds the maximum number of statistics or it contains no TIME data field the command fails.

For detailed information about record descriptors and descriptor files please see the descriptor file section of the MAP command description.

## Example

```
DQL> CREATE METRIX SPHINX
cont> IN STORAGE AREA SPHINX_DEFAULT_2005-08-25:00:00:00:1
cont> FROM PERFDAT$CFG:SPHINX_DSC.CFG;
DQL-I-CREATE, successfully created Metrix /SPHINX/
```

This example shows how to add a new metric (table) to the physical storage area referred by the filename alias SPHINX_DEFAULT_2005-08-25:00:00:00:1 (see the CREATE STORAGE AREA command example how to create this physical storage area). The name of the new metric is SPHINX. The descriptor file of the metric is PERFDAT$CFG:SPHINX_DSC.CFG.

Content of the descriptor file PERFDAT$CFG:SPHINX_DSC.CFG:

```
METRIX_SPHINX:

     Time:                FIELD$_DATETIME:8:    Time      :                [s]:
PART9 Ges:                FIELD$_FLOAT:  4:     Partion 9 Gesamt TA:       [1/s]:
PART9 ACK:                FIELD$_FLOAT:  4:     Partion 9 Ack TA:          [1/s]:
     PART9 NAK:           FIELD$_FLOAT:  4:     Partion 9 Nack TA:         [1/s]:
PART10 Ges:               FIELD$_FLOAT:  4:     Partion 10 Gesamt TA:      [1/s]:
PART10 ACK:     FIELD$_FLOAT:  4:     Partion 10 Ack TA:         [1/s]:
     PART10 NAK:          FIELD$_FLOAT:  4:     Partion 10 Nack TA:        [1/s]:
PART11 Ges:               FIELD$_FLOAT:  4:     Partion 11 Gesamt TA:      [1/s]:
PART11 ACK:     FIELD$_FLOAT:  4:     Partion 11 Ack TA:         [1/s]:
     PART11 NAK:          FIELD$_FLOAT:  4:     Partion 11 Nack TA:        [1/s]:
PART12 Ges:               FIELD$_FLOAT:  4:     Partion 12 Gesamt TA:      [1/s]:
PART12 ACK:     FIELD$_FLOAT:  4:     Partion 12 Ack TA:         [1/s]:
     PART12 NAK:          FIELD$_FLOAT:  4:     Partion 12 Nack TA:        [1/s]:
     Request Ges:         FIELD$_FLOAT:  4:     Gesamt TA:                 [1/s]:
     GES ACK:             FIELD$_FLOAT:  4:     Gesamt Ack TA:             [1/s]:
     GES NAK:             FIELD$_FLOAT:  4:     Gesamt Nack TA:            [1/s]:

METRIX_SPHINX_END:
```

## *CREATE STORAGE AREA*

This command creates a new physical storage area (data file).

### Format

```
CREATE STORAGE AREA alias_name [OSTYPE] os_name
        FILENAME file_name TIMERANGE FROM start_time TO stop_time
                SAMPLE INTERVAL time_in_sec
                        ALLOCATION allocsize_in_blks BLOCKS
                                [TYPE] { COLLECTION | TREND };
```

### Description

This command creates a new physical storage area (data file).

To create a physical storage some basic parameters have to be applied in order the newly created data file becomes part of the distributed collection database.

The alias_name parameter specifies the collection database alias the newly created data file belongs to. A collection database alias has the format:

NodeName_CollectionProfile

No restriction exists for the input. Neither the NodeName string has to match with an existing node in your environment nor has the CollectioProfile string to match with an existing collection profile in the collection profile table of the PERFDAT configuration database. Thus, the alias_name is freely definable.

The optional OSTYPE clause defines the type of data that will be stored in the newly created physical storage area. Since performance data collected are operating system dependent typically an operating system name (OpenVMS, Tru64, HP-UX …) is entered. However you can enter any string that does not exceed 12 characters that characterizes the data stored the most. Since this value is internally treated as informational data it has no effect but on the output of the SHOW HEADER command. If the OSTYPE string is not “OpenVMS” the SHOW HEADER command displays no summary information about the metrics stored in the physical storage area. In this case you have to apply the SHOW METRIX command explicitly to display the metrics stored in the physical storage area. The default is OpenVMS.

The FILENAME clause specifies the file name of the newly created physical storage area. Enter the file name without directory information. The CREATE command automatically creates the new data file in the directory PERFDAT$DB_LOCAL or PERFDAT$DB_TREND. It depends on the keyword used in the TYPE clause:

- TYPE clause missing
  The new data file will be stored in PERFDAT$DB_LOCAL.

- COLLECTION keyword used
  The new data file will be stored in PERFDAT$DB_LOCAL.

- TREND keyword used
  The new data file will be stored in PERFDAT$DB_TREND.

The TIMERANGE clause defines the time range data the physical storage area is valid for. If the timestamp of a data record is covered by into the time range of a physical storage area it can be inserted in case the metric the record belongs to exists. If the timestamp of the record is outside the time range of the physical storage area the insert operation is blocked.

One of the key control parameters for the OpenVMS data collector and the SNMP extension is the sample interval. In order to create a PERFDAT compliant physical storage area manually a sample interval in seconds has to be defined too. The SAMPLE INTERVAL clause is used to enter that sample interval in seconds. The sample interval is also important if you want to load CSV data into the newly created, empty physical storage area. The sample interval of the physical storage area should match the sample intervals defined by the timestamps in the CSV file. No duplicate inserts are allowed for data files of the PERFDAT distributed database. If the sample interval of the data records in a load CSV file differs from the sample interval defined in the physical storage area significantly data insert may fail due to duplicates.

For detailed information about CSV data load see the LOAD command description.

The ALLOCATION clause terminated by the BLOCKS keyword defines the initial file size of the new data file in blocks.

The optional TYPE clause defines the directory where to create the new physical storage area. If the TYPE clause is omitted or the COLLECTION keyword is entered, the new data file is created in the directory PERFDAT$DB_LOCAL. If the TREND keyword is applied the new data file is created in the directory PERFDAT$DB_TREND.

## Example

```
DQL> CREATE STORAGE AREA SPHINX_DEFAULT
cont> FILENAME SPHINX_25082005.DAT
cont> TIMERANGE FROM 25-AUG-200500:00 TO 26-AUG-200500:00
cont> SAMPLE INTERVAL 300 ALLOCATION 500 BLOCKS
cont> TYPE COLLECTION;
DQL-I-CREATE, successfully created storage area /PERFDAT$SPECIFIC:[DB]SPHINX_25082005.DAT_69238;1/
 alias /SPHINX_DEFAULT_2005-08-25:00:00:00:1/
```

This example shows how to create a new physical storage area. The newly create physical storage area is member of the collection database SPHINX_DEFAULT. The sample interval is 300 sec. The initial file size is 500 blocks. The physical storage area is valid to store data records containing timestamps between 25-AUG-200500:00and26-AUG-200500:00. Since the data file type is COLLECTION the physical storage area is created in the directory PERFDAT$DB_LOCAL (PERFDAT$SPECIFIC:[DB]);

## *DEATTACH ALIAS*

This command disconnects (closes) a previously attached (opened) collection database or logical storage area.

### Format

DEATTACH ALIAS alias_name [DATE date];

### Description

This command disconnects (closes) a previously attached (opened) collection database or logical storage area. The command syntax is equivalent to the ATTACH ALIAS command.

For detailed information about the command syntax please see the ATTACH ALIAS command description.

### Examples

Example 1

This is example shows how to disconnect from a collection database.

```
DQL> DEATTACH ALIAS VNOABS_2MIN;
DQL-I-ATTACH, successfully deattached file /VNOABS_2MIN_2003-SEP-19:00:03:00/
.
.
.
DQL-I-ATTACH, successfully deattached file /VNOABS_2MIN_2003-SEP-25:00:03:00/
```

In this example the collection database referenced by the alias VNOABS_2MIN (= sum of all data files that have been created by performance collections started with the collection profile 2MIN on node VNOABS) is disconnected (closed).

Example 2

This is an example how to disconnect (close) from a logical storage area.

```
DQL> DEATTACH ALIAS VNOABS_2MIN DATE 19-SEP-2003;
DQL-I-ATTACH, successfully deattached file /VNOABS_2MIN_2003-SEP-19:00:03:00/
DQL-I-ATTACH, successfully deattached file /VNOABS_2MIN_2003-SEP-19:15:03:00/
```

In this example the connection to the logical storage of the collection database referenced by the alias VNOABS_2MIN containing the data files of 19-SEP-2003is closed. The logical storage area consists of two physical storage areas

(data files)(VNOABS_2MIN_2003-SEP-19:00:03:00 and VNOABS_2MIN_2003-SEP-19:15:03:00).

## *DEATTACH FILE*

This command disconnects (closes) a previously attached (opened) physical storage area.

### Format

DEATTACH 'FILE filename_alias';

### Description

This command disconnects (closes) a previously attached (opened) physical storage area. The command syntax is equivalent to the ATTACH FILE command.

For detailed information about the command syntax please see the ATTACH FILE command description.

### Example

```
DQL> DEATTACH 'FILE VNOABS_2MIN_2003-SEP-18:20:43:00';
DQL-I-ATTACH, successfully deattached file /VNOABS_2MIN_2003-SEP-18:20:43:00/
```

In this example the connection (channel) to the physical storage area of the collection database VNOABS_2MIN (node: VNOABS, collection profile: 2MIN) created on 18-SEP-200320:43:00 is closed.

## *DEFINE DATA HOST*

The DEFINE DATA HOST command defines the node that shall host the data files created during the current DQL$ session.

### Format

DEFINE DATA HOST *node_name*;

### Description

The DEFINE DATA HOST command defines the node that shall host the data files created during the current DQL$ session. The user can define any member of the community the local node is member of and the archive node if configured on the local node.

It is a prerequisite that the node defined by the *node_name* parameter has VSI PERFDAT V3.3 or any higher version installed. If this is not the case the command fails.

Once the command has been successfully executed any subsequent EXTRACT and CREATE STORAGE AREA command of the DQL$ utility creates data files on the target node defined by the *node_name* parameter.

Prior to VSI PERFDAT V3.3 data files created by the EXTRACT and CREATE STORAGE AREA command of the DQL$ utility were always created on the local node. This is the default when you start a new DQL$ session.

### Example

Assume you are running a DQL$ session on node VMSTM1. The PERFDAT community defined on VMSTM1 consists of VMSTM1 and HOBEL. The archive node for VMSTM1 is VMSTM4. In order to redirect the file creation of data files to the archive node, enter the command sequence as listed below.

```
DQL> DEFINE DATA HOST VMSTM4;
DQL-I-DATAHOST, data file creation has been successfully redirected to node /VMSTM4/

DQL> CREATE STORAGE AREA SPHINX_DEFAULT
cont> FILENAME SPHINX_25082005.DAT
cont> TIMERANGE FROM 25-AUG-200500:00 TO 26-AUG-200500:00
cont> SAMPLE INTERVAL 300 ALLOCATION 500 BLOCKS
cont> TYPE COLLECTION;
DQL-I-CREATE, successfully created storage area
                                /PERFDAT$SPECIFIC:[DB]SPHINX_25082005.DAT_51993;1/
                        alias /SPHINX_DEFAULT_2005-08-25:00:00:00:1/ on node VMSTM4.
```

## *DEFINE ELEMENT*

When a stacked element report or a stacked deviation report is created these reports contain the calculated values for a single (stacked) element. This stacked element name can be (re)defined by applying the DEFINE ELEMENT command in advance of the CALCULATE statement.

### Format

DEFINE ELEMENT element_name;

### Description

When a stacked element report or a stacked deviation report is created these reports contain the calculated values for a single (stacked) element. This stacked element name can be redefined by applying the DEFINE ELEMENT command in advance of the CALCULATE statement.

The string length of the element name is limited to 64 characters.

For more information please see the CALCULATE command descriptions.

### Examples

See CALCULATE command description and examples.

## *DEFINE GRAPH_CFG*

This DEFINE GRAPH_CFG command defines the configuration file to be used for subsequent CREATE GRAPH commands.

**Format**

DEFINE GRAPH_CFG *cfg_filename*;

**Description**

The CREATE GRAPH command facilitates automated WEB based graphing and data analysis from the DQL$ command line. The layout of the graphs created by the CREATE GRAPH command can be customized by several parameters stored in a configuration file. You can define several configuration files on your own.

This DEFINE GRAPH_CFG command defines the configuration file to be used for subsequent CREATE GRAPH commands. The *cfg_filename* has to address an existing graph configuration file. If the file name defined by the *cfg_filename* parameter does not exist the command fails and the default configuration file:

PERFDAT$CFG:PERFDAT_CSV2PNG.CFG

will be used for subsequent CREATE GRAPH commands.

If you want to create graphs from the DQL$ utility with user-defined settings apply the DEFINE GRAPH_CFG command in advance of the CREATE GRAPH command.

For detailed information about the configuration parameters available please refer to the CREATE GRAPH command description.

**Examples**

See the CREATE GRAPH command description and examples.

## *DEFINE HEADER*

If data are exported to a CSV file the header line (comment) of that file and the caption of a graph created by applying the CREATE GRAPH command can be user-defined. This is done by applying this command in advance of the EXPORT and CREATE GRAPH command.

### Format

DEFINE HEADER “header string”;

### Description

If data are exported to a CSV file the header line (comment) of that file and the caption of a graph created by applying the CREATE GRAPH command can be user-defined. This is done by applying this command in advance of the EXPORT and CREATE GRAPH command.

Use quotation marks to terminate the header (caption) string. The maximum length of the header string is 2048 characters.

For more information please see the CREATE GRAPH and EXPORT command description.

### Examples

See the CREATE GRAPH and EXPORT command description and examples.

## *DEFINE PROCEDURE*

This command is used to define side specific, calculated statistics (measures) that can, once defined, be accessed as if they are part of the associated metrics of the collection databases available.

### Format

```
DEFINE PROCEDURE equation
        METRIX metric_name
                OSTYPE OS_name
                        DESCRIPTION description_text
                                UNIT unit_text
                                        [NODE node_name];
```

### Description

This command is used to define side specific, calculated statistics (measures) that can, once defined, be accessed as if they are part of the associated metrics of the collection databases available.

The equation parameter defines the user-defined statistics. Enter an equation. The name of the user-defined statistics to be created has to be entered and on the left and a valid function (procedure) that is used to calculate it on the right of the equal sign. E.g.:

```
$iCpuNorm = iCpuLoad / iCpuCnt
```

User-defined statistics are marked with a dollar ($) sign in front to indicate that they are calculated statistics. The user can, but doesn't have to, enter the dollar ($) sign when defining the stored procedure. If the dollar sign is omitted it is automatically assigned.

Statistics collected by the OpenVMS data collector or the SNMP extension, existing user-defined statistics and constant values can be used within the function (procedure) assigned. The supported operators are +, -, * and /. All variables used within the function assigned (statistics collected by the OpenVMS data collector or the SNMP extension, existing user-defined statistics) have to be member of the same metric as defined by the METRIX clause. Otherwise the command fails.

The metric_name parameter defined the METRIX the newly created user-defined statistics is member of.

The OS_name parameter in the OSTYPE clause defines the operating system the metric defined by the metric_name parameter is valid for.

The DESCRIPTION clause is used to briefly describe the new user-defined statistics. The string entered will be displayed when applying the SHOW STATISTICS command. Maximum length of the string is 64 characters. If you enter the string with parenthesis the string is stored case sensitive. Otherwise the string will be converted to upper case.

The UNIT clause is used to enter the unit of the user-defined statistics. Maximum length of the string is 16 characters. If you enter the string with parenthesis the string is stored case sensitive. Otherwise the string will be converted to upper case.

If the keyword applied to the OS_name parameter is one of the systems supported by any of the components of VSI PERFDAT (OpenVMS data collector, SNMP extension, EVA extension, RDB performance data import utility, CACHE data performance data import utility, VSI PERFDAT API):

- OpenVMS
- TRU64
- EVA
- Brocade
- Solaris
- Linux
- RDB
- CACHE
- The name of any application that uses the VSI PERFDAT API to insert data into the distributed VSI PERFDAT performance data base

the DQL$ utility performs several checks before creating the user-defined statistics:

- Checks if all statistics defined within the function (procedure) assigned to the user-defined statistics exist
- Checks the syntax of the function (procedure) assigned
    - Checks if supported operators (+, -, *, /) are applied only
    - Checks if all brackets are present

If one of these checks fails the user-defined statistics will not be created.

If an unknown keyword is assigned to the OS_name parameter the user is prompted to confirm the creation of the user-defined statistics. If the user confirms the creation the user-defined statistics is created unconditionally.

All clauses and parameters of the command except the NODE clause are mandatory.

Stored procedures can be defined generic or node specific. If you omit the NODE clause the user-defined statistics is generic. If you apply a node name using the NODE clause the user-defined statistics is node specific.

A generic user-defined statistics is valid for all performance databases accessible within the PERFDAT community the local node is member of and that contains the metric defined by the metric_name parameter and that contains performance data of the system defined by the OS_name parameter.

Node specific user-defined statistics are only valid for the specified metric stored in performance databases created on/for a specific node. If a node specific user statistics is defined this statistics can be accessed only if the user requests data from a performance database the node name field in the database header matches the node name string of the user-defined statistics (stored procedure).

Due to the generic/node specific statistics concept user-defined statistics with the same name can be defined that refer different formulas. If you define statistics with the same name with different formulas assigned as a generic and as a node specific user-defined statistics the node specific definition is used to calculate the statistics if you request data from a performance database that match the node criteria. If the node name field in the header of the performance database does not match the node name the node specific user statistics refers to the generic definition is used to calculate the user-defined statistics.

Once the command succeeded the statistics will be immediately accessible by all users accessing the distributed PERFDAT performance database via one of the nodes that share the same PERFDAT configuration database, since user-defined statistics and their parameter are stored in the stored procedure table of the PERFDAT configuration database. The only reason for having no access to a user-defined statistics is that some statistics defined within the function assigned to the user-defined statistics does not exist in the collection database the users are attached to.

There are several reasons to use this feature. Here are some examples:

- This feature is important in case you want to normalize data.
- You can use this feature to create special statistics you are interested in if these statistics are not directly collected by the OpenVMS data collector or the SNMP extension but all input parameters to compute it are available.

## Examples

Example 1:

This example shows how to normalize data by defining a user-defined statistics.

The statistics for the system wide CPU load collected by the OpenVMS data collector ranges from 0 to 100% * number of CPUs. Thus, if you are monitoring a system with 8 CPUs the statistics for the system wide CPU load collected by the

OpenVMS data collector ranges from 0 … 800 %. In order to fetch normalized data of the system wide CPU load ranging from 0 … 100 % create a user-defined statistics. In this example the user-defined statistics is named $iCpuNorm, but you can choose any other name.

```
DQL> DEFINE PROCEDURE $iCpuNorm = iCpuLoad / iCpuCnt METRIX SYSTEM
cont> OSTYPE OpenVMSDESCRIPTION "CPU load normalized" UNIT "[%]";
DQL-I-PROC, generic stored procedure /$iCpuNorm/ for metric /SYSTEM/,
                                                    OS Type /OPENVMS/ defined
```

Once defined the user-defined statistics $iCpuNorm can be fetched from any attached collection database created by the OpenVMS data collector in case the data of the statistics iCpuLoad and iCpuCnt exist are stored in these collection databases.

Since the NODE clause is omitted this user-defined statistics is generic (valid for the system metric of any OpenVMS performance database)

Example 2:

This example demonstrates how to create special statistics you are interested in if these statistics are not directly collected by the OpenVMS data collector or the SNMP extension but all input parameters to compute them are available.

The average I/O size of disk I/Os is not collected by the OpenVMS data collector but the number of I/Os to the device (iIOs) and the throughput (iMBs) is collected. Thus you can define the average I/O size statistics on your own:

```
DQL> DEFINE PROCEDURE $iIOSize = iMbs / iIOs * 1024 METRIX DEVICE
cont> OSTYPE OpenVMS DESCRIPTION "Average I/O size" UNIT "[kB]"
cont> NODE VMSTM1;
DQL-I-PROC, node specific stored procedure /$iIOSize/ for metric /DEVICE/,
                                                    OS Type /OPENVMS/,
                                                    Node /VMSTM1/ defined
```

Since the unit of the throughput is MB/s it is not simply divided by the number of I/Os but also multiplied by 1024 in order to have the user-defined statistics scaled to kByte.

The NODE name clause is applied. Thus, this user-defined statistics is only accessible via the system metric if the user requests data from an OpenVMS performance database that were created by the OpenVMS data collector on node VMSTM1.

## *DEFINE REGION*

This command invokes the regional options wizard to define regional settings.

### Format

DEFINE REGION reg_name;

### Description

This command invokes the regional options wizard to define regional settings.

Regional settings define the list separator, the format of numbers, date and time of the CSV files that are mapped, loaded or imported to the distributed PERFDAT performance database as well as how the DQL$ utility formats numbers, date, time and the list separator when exporting performance data to CSV files.

Defining regional settings increases the flexibility to map, load or import CSV files from different sources without any format preprocessing. In addition data can be exported to CSV files using the format expected by the target system to transfer the CSV file.

The reg_name parameter defines the name of the regional settings to be defined by the regional options wizard. You can enter any string. Blanks are allowed. Maximum length of the string is 56 characters. If you enter the parameter with parenthesis the string is stored case sensitive. Otherwise the string will be converted to upper case.

The newly created regional setting is stored in the regional setting table of the PERFDAT configuration database. Thus, once a regional setting is defined it is valid for all nodes sharing the same PERFDAT configuration database.

**Note**

This command just creates a new regional setting but does not affect the default regional setting of the current DQL$ session. To switch the default regional setting of the current DQL$ session use the SET REGION command.

### Example

In this example regional settings “German” is defined. Since the regional name is entered with parenthesis it is stored case sensitive.

```
DQL> DEFINE REGION "German";

 Name: German
```

Enter Decimal Symbol [.]: , ↵
Enter List Seperator [,]:;↵

Enter the day format:
d defines the day format (minimum number of day digits)
m defines the month format. If you assign mmm the month is displayed textual (JAN, FEB, MAR ...). Otherwise the month is displayed numerical (e.q. m -> 1, 9, 10; mm -> 01, 09, 10)
y defines the year format (number of year digits)

Enter Date Format [dd-mmm-yyyy]:
Enter month desriptors as a comma seperated list [e.g. JAN, FEB, ...]
Enter List: [JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC]:
JAN,FEB,MAR,APR,MAI,JUN,JUL,AUG,SEP,OKT,NOV,DEZ↵

DQL-I-CFGSUCCESS, successfully defined region setting /German/
To use it as default for the session use the SET REGION command.

## *DEFINE VIEW*

This command creates a cluster view. A cluster view maps performance data of different nodes for cluster wide performance analysis.

### Format

DEFINE VIEW view_name ALIAS alias_namelist;

### Description

This command creates a cluster view. A cluster view maps performance data of different nodes for cluster wide performance analysis.

The view_name parameter defines the name of the cluster view to be created. Enter a name that characterizes the cluster view the most. Once a cluster view is created a virtual collection database is accessible that maps the data of the cluster view members. The alias assigned to that virtual collection database is:

view_name_VIEW

The newly created virtual collection database of the cluster view can be accessed using the database alias shown above in the same way as any collection database created by the OpenVMS data collector or SNMP extension. For more information about the distributed performance database and database organization please see the chapters VSI PERFDAT distributed performance database and VSI PERFDAT Query Interface (DQL)or the manual VSI PERFDAT - Architecture and Technical Description.

The alias_namelist parameter lists the members of the newly created cluster view. Any existing collection databases created by the OpenVMS data collector, the SNMP extension or the auto-trend engine can be added to a cluster view. Enter the collection databases as a comma separated list.

Prerequisites for defining cluster views are:

- All collection databases addressed by the alias_namelist parameter exist and there exists at least one matching logical storage area within each of these collection databases. With other words, for at least one day performance data have to exist in all collection databases addressed.
- All collection databases addressed by the alias_namelist were created with the same sample interval.

If one of these checks fail the configuration request is rejected.

Cluster view definitions are node specific. Thus, a cluster view is visible to those users only that are connected to the distributed PERFDAT performance data via the same node the cluster view was configured.

Once the command succeeds the newly created cluster view is immediately accessible by all users connected to the same node this command was applied.

The advantage of defining cluster views is that the virtual collection database of a cluster view can be accessed in the same way by the DQL$ utility and the PERFDAT GUI as if it is a collection database created by the OpenVMS data collector or the SNMP extension. Thus, all methods and features to analyse performance data of single nodes are available for cluster views too. Consequently the workflow for cluster analysis does not differ from the workflow to analyse single node performance data.

Although in most cases cluster views will be created for cluster wide performance data analysis of OpenVMS clusters there exists no restriction that performance collection databases of OpenVMS cluster members only can be members of a cluster view. Any collection database of any node available can be added to a cluster view as long as these collection databases fulfil the criteria mentioned above.

## Example

This example shows how to create a cluster view for an OpenVMS cluster. The OpenVMS cluster consists of three nodes:

- BCSXTC
- VMSTM1
- VMSTM4

PERFDAT is up and running on all three nodes and a performance collection started with the DEFAULT collection profile is active on each node. The cluster name of the OpenVMS cluster is VIECLU.

To create a cluster view for the cluster members enter:

```
DQL> DEFINE VIEW VIECLU
cont> ALIAS BCSXTC_DEFAULT, VMSTM1_DEFAULT, VMSTM2_DEFAULT;
DQL-I-VIEW, view /VIECLU/ defined
```

Once the command succeeded the virtual collection database that maps the assigned collection databases can be accessed immediately for cluster analysis. The alias VIECLU_VIEW addresses the virtual collection database of the new cluster view.

## *DROP ALIAS*

This command drops (deletes) a logical storage area or a whole collection database.

### Format

DROP ALIAS alias_name [DATE date];

### Description

This command drops (deletes) a logical storage area or a whole collection database depending if the optional DATE clause is specified or not.

The ALIAS clause specifies the alias of the collection database to delete. That database alias can't be user-defined. DQL$ assigns these aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

If you want to delete a logical storage area (all data files that have been created on the same day) of a collection database, the DATE clause is mandatory. Use OpenVMS date format to define the day of interest. If you omit the DATE clause all data files (physical storage areas) of the collection database defined by the ALIAS clause are deleted.

Logical storage areas/collection databases can be deleted if they are not connected by any DQL$ or PDBC$SRV session. Otherwise the command fails.

### Example

This example shows how to drop (delete) a logical storage area from the collection database SPHINX_DEFAULT.

```
DQL> DROP ALIAS SPHINX_DEFAULT DATE 25-AUG-2005;
DQL-I-DELNAME, removing /SPHINX_DEFAULT_2005-08-25:00:00:00:1/ from DQL name
cache on node VMSTM1
DQL-I-DELNAME, removing /SPHINX_DEFAULT_2005-08-25:00:00:00:1/ from DQL name
cache on node VMSTM4
DQL-I-DROP, dropping physical storage area /SPHINX_DEFAULT_2005-08-25:00:00:00:1/
successfully completed
DQL-I-DELNAME, removing /SPHINX_DEFAULT_2005-08-25:08:00:00:1/ from DQL name
cache on node VMSTM1
DQL-I-DELNAME, removing /SPHINX_DEFAULT_2005-08-25:08:00:00:1/ from DQL name
cache on node VMSTM4
```

```
DQL-I-DROP, dropping physical storage area /SPHINX_DEFAULT_2005-08-25:08:00:00:1/
successfully completed
11-SEP-2006 14:33:30: DQL-I-REBUILD, rebuild database information from BCSXTC
11-SEP-2006 14:33:30: DQL-I-REBUILD, rebuild database information from VMSTM1
11-SEP-2006 14:33:30: DQL-I-REBUILD, rebuild database information from VMSTM4
DQL-I-REMDB, physical storage area /SPHINX_DEFAULT_2005-08-25:00:00:00:1/ removed
from database view
DQL-I-REMDB, physical storage area /SPHINX_DEFAULT_2005-08-25:08:00:00:1/ removed
from database view
```

Since the specified logical storage area (= all physical storage area create on 25-AUG-2005) consists of two physical storage areas both physical storage areas are deleted.

## *DROP METRIX*

Drops (deletes) the metric (table) defined by the metric_name parameter from all attached physical storage areas that metric exist.

### Format

DROP METRIX metric_name;

### Description

Drops (deletes) the metric (table) defined with the metric_name parameter from all attached physical storage areas that metric exist.

**Note**

Before you apply the DROP METRIX command, make sure that you are attached to these physical storage areas only you actually want to drop (delete) the metric. Use the SHOW PHYSICAL STORAGE AREA command to view which physical storage areas are attached (opened) by the actual DQL$ session.

### Example

This example shows how to drop the metric SPHINX from a logical storage area. The logical storage area consists of a single physical storage area (see also CREATE METRIX command example). Attach the logical storage area:

```
DQL> ATTACH ALIAS SPHINX_DEFAULT DATE 25-AUG-2005;
```

Show available metrics:

```
DQL> SHOW METRIX;

METRIX DEFINITION of storage area SPHINX_DEFAULT_2005-08-25:00:00:00:1

            Metrics enabled         Element Count
------------------------------------------------------------
                     SPHINX                     1
```

Drop metric SPHINX:

```
DQL> DROP METRIX SPHINX;
DQL-I-DROP, metrix /SPHINX/ of physical storage area
                                /SPHINX_DEFAULT_2005-08-25:00:00:00:1/ successfully dropped
DQL-I-DROP, dropping metrix /SPHINX/ successfully completed
```

## *DROP PHYSICAL STORAGE AREA*

Drops (deletes) the physical storage area defined by the filename_alias parameter.

### Format

DROP PHYSICAL STORAGE AREA filename_alias;

### Description

Drops (deletes) the physical storage area defined by the filename_alias parameter. The filename alias is not the file name of the physical storage area but an alias DQL$ defines when it starts up. These filename aliases are displayed when you issue the SHOW PHYSICAL STORAGE AREA command. The filename alias includes information about the database and logical storage area that physical storage area belongs to.

This command can be applied to physical storage areas only that are not attached by any DQL$ or PDBC$SRV session. If any DQL$ or PDBC$SRV session is attached to the physical storage area the command fails.

### Example

This example shows how to drop (delete) a physical storage area. The collection database SPHINX_DEFAULT contains the physical storage area SPHINX_DEFAULT_2005-08-25:00:00:00:1 (see CREATE STORAGE AREA command example).

```
DQL> DROP PHYSICAL STORAGE AREA SPHINX_DEFAULT_2005-08-25:00:00:00:1;
DQL-I-DELNAME, removing /SPHINX_DEFAULT_2005-08-25:00:00:00:1/ from DQL name
cache on node VMSTM1
DQL-I-DELNAME, removing /SPHINX_DEFAULT_2005-08-25:00:00:00:1/ from DQL name
cache on node VMSTM4
DQL-I-DROP, dropping physical storage area /SPHINX_DEFAULT_2005-08-25:00:00:00:1/
successfully completed
11-SEP-2005 14:33:30: DQL-I-REBUILD, rebuild database information from VMSTM1
11-SEP-2005 14:33:30: DQL-I-REBUILD, rebuild database information from VMSTM4
DQL-I-REMDB, physical storage area /SPHINX_DEFAULT_2005-08-25:00:00:00:1/ removed
from database view
```

## *EXIT*

This command terminates the DQL$ session.

### Format

EXIT

### Description

This command disconnects from any collection database actually attached and terminates the DQL$ session.

## *EXPORT*

The EXPORT command is used to select data from PERFDAT collection databases and/or logical storage areas and to store these data in a CSV file timely ordered.

### Format

```
EXPORT [STACKED] statistics_itemlist
        FROM metrix_name
                ALIAS alias_namelist [DATE] date_list
                        [ELEMENT] element_list
                        [WHERE] filter_list
                        [LIMIT] number
                        [INTO] file_name
                        [FORMAT] {SINGLE_LINE | MULTI_LINE | T4};
```

### Description

The EXPORT command is used to select data from PERFDAT collection databases and/or logical storage areas and to store these data in a CSV file timely ordered.

Depending if the STACKED argument is applied or not the EXPORT query is stacked or un-stacked.

The stacked form of the EXPORT query stacks the element data of each statistics fetched from the source collection databases before inserting to the output CSV file. Using the un-stacked form the data are not pre-processed.

As said before the stacked form of the EXPORT statement stacks the element data of each statistics selected. Thus, the stacked form of the query can be used:

- If you are interested in the stacked values of a group of elements stored in a single collection database (e.g. overall CPU load caused by a group of processes on a single node)
- If you are interested in the stacked values of a particular element stored in collection databases that refer different nodes (e.g. total I/O requests from all cluster members on a cluster wide mounted disk)
- If you are interested in the stacked values of a group of elements stored in collection databases that refer different nodes (e.g. overall I/O requests from all cluster members on all cluster wide mounted disks, cluster wide CPU load caused by a number of processes).

If the clauses of the EXPORT statement are defined in a way that only one element of the metric specified is selected and the data source is a single collection database (data collected for one node) the stacked and the un-stacked form of the query returns the same result.

Prerequisite:
The data files of the collection database / logical storage areas defined by the ALIAS and DATE clause have to be attached in advance using the ATTACH command.

DQL$ evaluates the clauses of the EXPORT statement in the following order:

1. FROM
2. ALIAS
3. DATE
4. ELEMENT
5. WHERE
6. LIMIT
7. INTO
8. FORMAT

There exist two modes to address the data fields (statistics) to export:

- Base address mode
  In this case the statistics to fetch from the source database and the metric the selected statistics are member of are defined separately. The statistics_itemlist specifies the names of the statistics (field names) to fetch from the metric defined by the FROM clause. Enter the statistics as a comma separated list. In this case the FROM clause is mandatory. E.g. using the base address mode to export the system wide CPU load (field name: iCpuLoad) and kernel mode (field name: iKernel) from metric SYSTEM of an OpenVMS collection database the EXPORT statement has the form:

  DQL$> EXPORT iCpuLoad, iKernel FROM SYSTEM ALIAS…;

- Direct address mode
  In this case the statistics to fetch from the source database are entered using the full data field address. Enter the full data field addresses of all statistics to fetch (statistics_itemlist parameter) as a comma separated list. The full data field address of a statistics consists of the metric the statistics belongs to and its name (field name). The format is:

  MetricName.StatisticsName

  Since the metric the statistics belongs to is part of the full data field address the FROM clause must not be defined. If you enter the FROM clause the EXPORT statement fails. Using the direct address mode to export the same data from a source database to a CSV file as shown in the example above the EXPORT statement has the form:

  DQL$> EXPORT SYSTEM.iCpuLoad, SYSTEM.iKernel ALIAS…;

**Note**

In contrast to the SELECT statement you can enter statistics within the statistics item list that belong to different metrics of the source database in case the export format is SINGLE or T4 (see FORMAT clause description). If the export format is MULTI_LINE the statistics defined within the statistics item list have to be member of the same metric. Otherwise the command fails.

For both address modes it is a prerequisite that:

- All statistics specified must exist within the metric(s) defined
- The metric(s) (table(s)) defined must exist in at least in one collection databases / logical storage areas defined by the ALIAS and DATE clause but not in all.

Using the stacked form of the EXPORT statement a column per metric addressed by the EXPORT query (column header: …iElementCnt) that contains the number of elements that match the filter criteria applied (see description of the ELEMENT and WHERE clause) and that are used to calculate the stacked value(s) of the selected statistics is automatically appended to the output CSV file.

For detailed information about the clauses:

- ALIAS
- DATE
- WHERE
- LIMIT

please refer to the SELECT command description.

As with the SELECT statement the optional ELEMENT clause can be used to filter the elements the EXPORT query applies to. Enter the element filter as a comma (,), or OR sign (|) separated list. Elements that should be excluded from the EXPORT query have to be preceded with the '!=' or '<>' tag in the comma separated list of the ELEMENT clause. VSI PERFDAT V3.0and higher versions provide full wildcard support. Asterisk (*) and percent sign (%) wildcard characters can be placed anywhere within each string of the comma separated element list.If you enter quotation marks at the beginning and the end of an element string the string is taken literally (no wildcard operation performed on that string even if it contains wildcard characters). If you omit the ELEMENT clause all element data of the statistics selected stored in the collection database / logical storage areas defined by the ALIAS and DATE clause are read.

**Note**

If you export statistics to a CSV file that belong to different metrics using the direct address mode make sure that that the element filter includes filter criteria for all metrics addressed. E.g. you want to export the system wide CPU load and the CPU consumption of all PERFDAT processes from an OpenVMS collection database. Since the system wide CPU load is member of the metric SYSTEM and the metric of the CPU consumption of all

PERFDAT processes is PROCESS you have to apply the element filter for the SYSTEM and the PROCESS filter within the ELEMENT clause. There exists only one element in the metric SYSTEM called OpenVMS. To filter for the PERFDAT processes you can apply the wildcard filter PERFDAT*. Thus, the resulting element filter string is:

DQL>… ELEMENT OpenVMS | PERFDAT* …;

or

DQL>… ELEMENT OpenVMS , PERFDAT* …;

If you omit one of the filter strings either the data selected from the SYSTEM or those selected from the PROCESS metric are not exported.

The file name of the CSV export file can be defined by the optional INTO clause. If you omit the clause the default file name

SYS$LOGIN:EXPORT.CSV

is used.

The optional FORMAT clause defines the layout of CSV file to create. You can specify:

- SINGLE_LINE
  In this case one data record per timestamp is inserted to the CSV file. The first column of the CSV file contains the timestamp. All other columns contain the data of a dedicated statistics and element. If no values are available for a dedicated statistics, element and timestamp the data field is left blank. The header of each column but the timestamp column has the format:

  [MetricName]ElementName.StatisticsName

  The maximum record length of a CSV created by the EXPORT statement using the SINGLE layout option is 16000 characters. Thus, before you apply the EXPORT statement make sure that the resulting header string does not exceed this limit. Otherwise the command fails.
- MULTI_LINE
  In this case each data record contains all data of a single element. Thus, several records with the same timestamp will be inserted into the resulting CSV files if the filter criteria applied addresses more than one element and the metric defined contains more than one element.
  The layout format MULTI_LINE can only be applied if you select statistics that are member of the same metric of a single collection database (the ALIAS clause contains only one collection database alias).
  The first record of the output CSV file contains (header line) user comments. The user comment can be defined by the DEFINE HEADER command in advance of the EXPORT statement. If the header line has not been user-defined either the content of the ALIAS clause in case of the un-stacked form of the statement or the keyword STACKED in case of the stacked form of the EXPORT command is stored. The second row contains the column header. It consists of the names of the statistics requested. All other rows are data rows containing data for the statistics listed in the header record of a particular element.

- T4
  A T4 compatible CSV file will be created that can be directly visualised with T4Viz.

If you omit the FORMAT clause the CSV file layout will be SINGLE_LINE.

To make sure that the output CSV file is formatted (numbers, date, time, list separator) as expected by the target system to transfer the CSV file apply the DEFINE REGION and SET REGION command in advance. For more information about regional settings please refer to the DEFINE REGION and SET REGION command description.

## Examples

Example 1

This examples demonstrates the use of the un-stacked from of the EXPORT query using base address mode. In this example the CPU load (iCpuLoad) and kernel mode load (iKernel) caused by the process PERFDAT on node VMSTM2 between 30-DEC-2005 02:00 and 30-DEC-2005 02:30 including the timestamps ('Time' statistics) are selected. Process data are stored in metric PROCESS. The user-defined output CSV file is SYS$LOGIN:PERFDAT_30122005.CSV. Since the FORMAT clause is omitted the CSV layout in use is SINGLE_LINE.

```
DQL> EXPORTTime, iCpuLoad, iKernel FROM PROCESS
cont> ALIAS VMSTM2_DEFAULT DATE 30-DEC-2005
cont> ELEMENT PERFDAT
cont> WHERE TIME >= 30-DEC-2005 02:00, TIME <= 30-DEC-2005 02:30
cont> INTO SYS$LOGIN:PERFDAT_30122005.CSV;
DQL-EXPORT, start export data to /SYS$LOGIN:PERFDAT_30122005.CSV/
```

Content of output file SYS$LOGIN:PERFDAT_30122005.CSV

```
Time, VMSTM2::[PROCESS]PERFDAT.iCpuLoad, VMSTM2::[PROCESS]SYSTEM.iCpuLoad
30-DEC-2005 02:00:00, 0.033, 0.000
30-DEC-2005 02:02:00, 0.317, 0.000
30-DEC-2005 02:04:00, 0.175, 0.000
30-DEC-2005 02:06:00, 0.142, 0.000
30-DEC-2005 02:08:00, 0.083, 0.000
30-DEC-2005 02:10:00, 0.150, 0.000
30-DEC-2005 02:12:00, 0.108, 0.000
30-DEC-2005 02:14:00, 0.217, 0.000
30-DEC-2005 02:16:00, 0.317, 0.000
30-DEC-2005 02:18:00, 0.250, 0.000
30-DEC-2005 02:20:00, 0.258, 0.000
30-DEC-2005 02:22:00, 0.250, 0.000
30-DEC-2005 02:24:00, 0.275, 0.000
30-DEC-2005 02:26:00, 0.033, 0.000
30-DEC-2005 02:28:00, 0.033, 0.000
```

The ELEMENT clause is applied to filter for process PERFDAT and the WHERE clause to filter for data of the time period 30-AUG-200502:00 to 30-AUG-200502:30. In this case the query accesses the logical storage area of 30-AUG-

2005 of the collection database VMSTM2_DEFAULT. Since no header line has been user-defined (DEFINE HEADER) in advance of the EXPORT statement the default un-stacked header – ALIAS clause content VMSTM2_DEFAULT - is inserted.

Example 2

This example demonstrates the difference in the CSV layout if MULTI_LINE CSV format is defined (FORMAT clause). Except the FORMAT clause the EXPORT statement is identical to example 1.

```
DQL> EXPORTTime, iCpuLoad, iKernel FROM PROCESS
cont> ALIAS VMSTM2_DEFAULT DATE 30-DEC-2005
cont> ELEMENT PERFDAT
cont> WHERE TIME >= 30-DEC-2005 02:00, TIME <= 30-DEC-2005 02:30
cont> INTO SYS$LOGIN:PERFDAT_30122005.CSV FORMAT MULTI_LINE;
DQL-EXPORT, start export data to /SYS$LOGIN:PERFDAT_30122005.CSV/
```

Content of output file SYS$LOGIN:PERFDAT_30122005.CSV

```
VMSTM2_DEFAULT
Time, iCpuLoad
30-DEC-2005 02:00:00, 0.033
30-DEC-2005 02:00:00, 0.000
30-DEC-2005 02:02:00, 0.317
30-DEC-2005 02:02:00, 0.000
.
.
30-DEC-2005 02:24:00, 0.275
30-DEC-2005 02:24:00, 0.000
30-DEC-2005 02:26:00, 0.033
30-DEC-2005 02:26:00, 0.000
30-DEC-2005 02:28:00, 0.033
30-DEC-2005 02:28:00, 0.000
```

As you can see there exist pairs of records containing the same timestamp. The reason is that in multi-line mode a data record consists of all data of a single element. Since two elements were defined by the ELEMENT clause two records per timestamp are inserted - the first one refers to process PERFDAT the second to process SYSTEM.

Example 3

This example demonstrates the use of the un-stacked form of the EXPORT query using the direct address mode. The source collection database and the time range selected to export data are the same as in example 1. In this example the system wide CPU load (member of metric SYSTEM) and the CPU consumption of process PERFDAT (member of metric PROCESS) are exported to the CSV file SYS$LOGIN:EXPORT.CSV since the INTO clause is missing. The output CSV layout is SINGLE_LINE since the FORMAT clause is omitted. Since the name of element of the SYSTEM metric is OpenVMS, and the process element to filter is PERFDAT the element filter has to contain both element names – OpenVMS | PERFDAT.

```
DQL> EXPORTSYSTEM.iCpuLoad, PROCESS.iCpuLoad
cont> ALIAS VMSTM2_DEFAULT DATE 30-DEC-2005
cont> ELEMENT OpenVMS | PERFDAT
cont> WHERE TIME >= 30-DEC-2005 02:00, TIME <= 30-DEC-2005 02:30;
DQL-EXPORT, start export data to /SYS$LOGIN:EXPORT.CSV/
```

Content of output file SYS$LOGIN:EXPORT.CSV

```
Time, VMSTM2::[SYSTEM]OPENVMS.iCpuLoad, VMSTM2::[PROCESS]PERFDAT.iCpuLoad
30-DEC-2005 02:00:00, 0.068, 0.033
30-DEC-2005 02:02:00, 26.590, 0.317
30-DEC-2005 02:04:00, 13.657, 0.175
30-DEC-2005 02:06:00, 14.945, 0.142
30-DEC-2005 02:08:00, 13.325, 0.083
30-DEC-2005 02:10:00, 11.344, 0.150
30-DEC-2005 02:12:00, 19.197, 0.108
,30-DEC-2005 02:14:00, 20.418, 0.217
30-DEC-2005 02:16:00, 23.851, 0.317
30-DEC-2005 02:18:00, 19.404, 0.250
30-DEC-2005 02:20:00, 23.197, 0.258
30-DEC-2005 02:22:00, 23.432, 0.250
30-DEC-2005 02:24:00, 47.851, 0.275
30-DEC-2005 02:26:00, 3.167, 0.033
30-DEC-2005 02:28:00, 0.099, 0.033
```

Example 4

This example demonstrates the use of the un-stacked form of the EXPORT query using the direct address mode to select data from different sources (collection databases / logical storage areas defined by the ALIAS and DATE clause) and the selected statistics and metrics do not exist in all collection data files.

We want to export the I/O rate of all FC-adapters of the system VMSTM1 and the throughput of the FC-switches these adapters are connected into the CSV file SYS$LOGIN:IOPATH.CSV. VMSTM1 has two FC- adapters FGA and FGB. FGA is connected to Port_1

```
DQL> ATTACH ALIAS VMSTM1_DEFAULT DATE 26-NOV-2007;
DQL> ATTACH ALIAS FC-SWITCH1_DEFAULT DATE 26-NOV-2007;
DQL> ATTACH ALIAS FC-SWITCH2_DEFAULT DATE 26-NOV-2007;

DQL> EXPORT IOPATHES.iOpCnt, PORT.TotWordsALIAS
cont> VMSTM1_DEFAULT, FC-SWITCH1_DEFAULT, FC-SWITCH2_DEFAULT
cont> DATE 26-NOV-2007 ELEMENT FG* | PORT_0
cont> INTO SYS$LOGIN:IOPATH.CSV;
```

Example 5

This example demonstrates the use of the stacked form of the EXPORT query using the direct address mode. The source collection database is a cluster view. This view consists of the node VMSTM1 and VMSTM2. The time range selected is 15-JAN-200602:00 to 16-JAN-200602:00. In this example the stacked CPU load

(iCpuLoad) and kernel mode load (iKernel) caused by all PERFDAT and DQL processes active on the members of the cluster view are displayed (cluster wide stacked values).

```
DQL> SHOW VIEW;
          View                    referenced Aliases
-------------------------------------------------------------------
          VMSALL                  VMSTM2_DEFAULT
                                  VMSTM1_DEFAULT

DQL> EXPORTSTACKED PROCESS.iCpuLoad, PROCESS.iKernel
cont> ALIAS VMSALL_VIEW DATE 15-JAN-2006, 16-JAN-2006
cont> ELEMENT PERFDAT*, DQL*
cont> WHERE TIME >=15-JAN-200602:00, TIME <= 16-JAN-200602:00
cont> INTO SYS$LOGIN:PROCESS_CLUSTER.CSV;
DQL-EXPORT, start export data to /SYS$LOGIN:PROCESS_CLUSTER.CSV/
```

Content of output file SYS$LOGIN:PROCESS_CLUSTER.CSV

```
Time, [PROCESS]PERFDAT*, DQL*.iCpuLoad, [PROCESS]PERFDAT*, DQL*.iKernel,
                                                    [PROCESS]PERFDAT*, DQL*.iElementCnt
15-JAN-2006 02:00:00, 26.583, 6.775, 12.000
15-JAN-2006 02:02:00, 123.317, 33.875, 15.000
15-JAN-2006 02:04:00, 34.450, 8.642, 12.000
15-JAN-2006 02:06:00, 25.817, 6.492, 10.000
15-JAN-2006 02:08:00, 36.758, 8.817, 10.000
15-JAN-2006 02:10:00, 0.508, 0.083, 8.000
15-JAN-2006 02:12:00, 0.583, 0.092, 8.000
15-JAN-2006 02:14:00, 0.508, 0.133, 8.000.
.
.
16-JAN-2006 01:44:00, 0.558, 0.083, 0.000
16-JAN-2006 01:46:00, 0.642, 0.108, 0.000
16-JAN-2006 01:48:00, 0.625, 0.092, 0.000
16-JAN-2006 01:50:00, 0.558, 0.083, 0.000
16-JAN-2006 01:52:00, 0.500, 0.067, 0.000
16-JAN-2006 01:54:00, 0.675, 0.108, 0.000
16-JAN-2006 01:56:00, 0.567, 0.058, 0.000
16-JAN-2006 01:58:00, 1.225, 0.292, 0.000
```

Although not defined the CSV output file contains a data column named [PROCESS]PERFDAT*, DQL*.iElemenCnt since the stacked form of the EXPORT statement is used. This statistics displays the number of elements that match the filter criteria of the EXPORT (ELEMENT and WHERE clause) statement and that are used to calculate the stacked values.

Since the FORMAT clause is missing the CSV output layout is SINGLE_LINE. Since no additional header line is inserted when using the single line output format the DEFINE HEADER command in advance has no effect on the CSV output.

If you apply the same export CSV query defining multi line CSV layout the first record of the CSV file contains the string defined by the DEFINE HEADER command.

```
DQL> DEFINE HEDAER "Cluster wide CPU Load / Kernel mode of PERFDAT*, DQL*"

DQL> EXPORTSTACKED PROCESS.iCpuLoad, PROCESS.iKernel
cont> ALIAS VMSALL_VIEW DATE 15-JAN-2006, 16-JAN-2006
cont> ELEMENT PERFDAT*, DQL*
cont> WHERE TIME >=15-JAN-200602:00, TIME <= 16-JAN-200602:00
cont> INTO SYS$LOGIN:PROCESS_CLUSTER.CSV FORMAT MULTI_LINE;
DQL-EXPORT, start export data to /SYS$LOGIN:PROCESS_CLUSTER.CSV/
```

Content of output file SYS$LOGIN:PROCESS_CLUSTER.CSV

```
Cluster wide CPU Load / Kernel mode of PERFDAT*, DQL*
Time, iCpuLoad, iKernel, iElementCnt
15-JAN-2006 02:00:00, 26.583, 6.775, 12.000
15-JAN-2006 02:02:00, 123.317, 33.875, 15.000
15-JAN-2006 02:04:00, 34.450, 8.642, 12.000
15-JAN-2006 02:06:00, 25.817, 6.492, 10.000
15-JAN-2006 02:08:00, 36.758, 8.817, 10.000
15-JAN-2006 02:10:00, 0.508, 0.083, 8.000
15-JAN-2006 02:12:00, 0.583, 0.092, 8.000
15-JAN-2006 02:14:00, 0.508, 0.133, 8.000.
.
.
16-JAN-2006 01:44:00, 0.558, 0.083, 8.000
16-JAN-2006 01:46:00, 0.642, 0.108, 8.000
16-JAN-2006 01:48:00, 0.625, 0.092, 8.000
16-JAN-2006 01:50:00, 0.558, 0.083, 8.000
16-JAN-2006 01:52:00, 0.500, 0.067, 8.000
16-JAN-2006 01:54:00, 0.675, 0.108, 8.000
16-JAN-2006 01:56:00, 0.567, 0.058, 8.000
16-JAN-2006 01:58:00, 1.225, 0.292, 8.000
```

Example 6

The EXPORT query is the same as in example 4 but the regional settings are changed in advance using the DEFINE REGION and SET REGION command to demonstrate the effect of regional settings on the CSV output format.

```
DQL>DEFINE REGION FRANCE;

 Name: FRANCE
      Enter Decimal Symbol [.]:,
      Enter List Seperator [,]:;

      Enter the day format:
      d      defines the day format (minimum number of day digits)
      m      defines the month format. If you assign mmm the month is
             displayed textual (JAN, FEB, MAR ...). Otherwise the month
             is displayed numerical (e.q. m -> 1, 9, 10; mm -> 01, 09, 10)
      y      defines the year format (number of year digits)

      Enter Date Format [dd-mmm-yyyy]: dd/mm/yyyy
      Enter month desriptors as a comma seperated list [e.g. JAN, FEB, ...]
      Enter List:  [JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC]:
```

```
DQL-I-CFGSUCCESS, successfully defined region setting /FRANCE/
        To use it as default for the session use the SET REGION command.

DQL> SET REGION FRANCE;
DQL-I-CFGSUCCESS, default region setting changed to /FRANCE/

DQL> EXPORTSTACKED PROCESS.iCpuLoad, PROCESS.iKernel
cont> ALIAS VMSALL_VIEW DATE 15-JAN-2006, 16-JAN-2006
cont> ELEMENT PERFDAT*, DQL*
cont> WHERE TIME >=15-JAN-200602:00, TIME <= 16-JAN-200602:00
cont> INTO SYS$LOGIN:PROCESS_CLUSTER.CSV;
DQL-EXPORT, start export data to /SYS$LOGIN:PROCESS_CLUSTER.CSV/
```

Content of output file SYS$LOGIN:PROCESS_CLUSTER.CSV

```
Time; [PROCESS]PERFDAT*, DQL*.iCpuLoad; [PROCESS]PERFDAT*, DQL*.iKernel;
                                        [PROCESS]PERFDAT*, DQL*.iElementCnt
15/01/2006 02:00:00; 26,583; 6,775; 12,000
15/01/2006 02:02:00; 123,317; 33,875; 15,000
15/01/2006 02:04:00; 34,450; 8,642; 12,000
15/01/2006 02:06:00; 25,817; 6,492; 10,000
15/01/2006 02:08:00; 36,758; 8,817; 10,000
15/01/2006 02:10:00; 0,508; 0,083; 8,000
15/01/2006 02:12:00; 0,583; 0,092; 8,000
15/01/2006 02:14:00; 0,508; 0,133; 8,000.
.
.
16/01/2006 01:44:00, 0.558, 0.083, 8.000
16/01/2006 01:46:00, 0.642, 0.108, 8.000
16/01/2006 01:48:00, 0.625, 0.092, 8.000
16/01/2006 01:50:00, 0.558, 0.083, 8.000
16/01/2006 01:52:00, 0.500, 0.067, 8.000
16/01/2006 01:54:00, 0.675, 0.108, 8.000
16/01/2006 01:56:00, 0.567, 0.058, 8.000
16/01/2006 01:58:00, 1.225, 0.292, 8.000
```

## *EXTRACT REPORT*

The EXTRACT REPORT command extracts trend and capacity reports according to a report profile.

### Format

```
EXTRACT REPORT report_profile
                    [ALIAS] alias_name
                                [FROM start_time TO stop_time];
```

### Description

The EXTRACT REPORT command extracts trend and capacity reports according to a report profile. The report profile to use is defined by the report_profile parameter. Report profiles are created using the PERFDAT_MGR utility.

For more information about defining trend and capacity reports please see the manual VSI PERFDAT - PERFDAT_MGR Reference Manual.

The ALIAS clause is optional. It is used to extract a trend report with the same settings but accessing a different data source as defined in the report profile (= report_profile parameter). The ALIAS clause specifies the alias of the source collection database. That database alias can't be user-defined. DQL$ assigns these aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. The aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

If you omit the ALIAS clause the target trend database to insert the trend/capacity report defined by the report_profile parameter is addressed by the alias

LocalNode_report_profile

E.g. you execute the EXTRACT REPORT command on node VMSTM1 using the report profile WEEK the target database alias will be VMSTM1_WEEK.

In case the ALIAS clause defining the source database is present the target database alias is

NodeName_report_profile

E.g. the source database is BCSXTC_2MIN and you execute the EXTRACT REPORT command on node VMSTM1 using the report profile WEEK the target database alias will be BCSXTC_WEEK.

The physical storage areas of the target database are automatically created if the report time range for trend and capacity report processing is not covered by the existing physical storage areas of the target collection database.

The FROM and TO clause are optional. These keywords define the time range for trend and capacity report processing. The date/time format is dd-mmm-yyyy hh:mm:ss.mm.If one keyword is used the other is mandatory.

If you omit both – the FROM and TO – clause the time range covered by the whole collection database is used for trend and capacity report processing is

## Example

```
DQL> EXTRACT WEEK ALIAS VMSTM1_DEFAULT
cont>FROM 1-AUG-2005 TO 2-AUG-2005;

     Creating Trend Report: WEEK for Node: VMSTM1

DQL-I-CREATEAREA, storage area not created
DQL-I-CREATEAREA, specified time range totally or partly captured by other storage area
(Version: 1)
DQL-I-CREATEMETRIX, metrix already exists
DQL-I-CREATEMETRIX, metrix already exists
DQL-I-CREATEMETRIX, metrix already exists
DQL-I-CREATEMETRIX, metrix already exists
DQL-I-CREATEMETRIX, metrix already exists
DQL-I-CREATEMETRIX, metrix already exists
DQL-I-CREATEMETRIX, metrix already exists
DQL-I-CREATEMETRIX, metrix already exists

Processing trend report - TimeRange: 1-AUG-200500:00:00 – 2-AUG-200500:00:00

     Processing date: 1-SEP-2005
.....................................................
Memory Allocation statistics ...
15-SEP-2005 02:17:35: PERFDAT-I-VMSTAT,  VM allocated: 10042,
                                         VM allocation size: 273.147 MB
15-SEP-2005 02:17:35: PERFDAT-I-VMSTAT,  VM deallocated: 10042,
                                         VM deallocation size: 273.147 MB
```

This example extracts the trend report by the report profile WEEK. The source collection database is VMSTM1_DEFAULT. The time range for report processing is 1-AUG-2005 to 2-AUG-2005.

## *HELP*

This command invokes the online help.

### Format

HELP

### Description

This command invokes the online help.

## *IMPORT*

This command imports valid CSV files into an existing collection database.

### Format

```
IMPORT file_name AS metric
                [DESCRIBED BY descriptor_filename]
                                TARGET DATABASE alias_name;
                                [FORMAT] { SINGLE_LINE | MULTI_LINE };
```

### Description

This command imports valid CSV files into an existing collection database.

The content of CSV files with two different layouts can be imported (see also the EXPORT command description). The optional FORMAT clause defines the layout of the input CSV files addressed by the MAP command. Valid keywords are SINGLE_LINE and MULTI_LINE. If the FORMAT clause is omitted the default layout format is SINGLE_LINE:

- SINGLE_LINE
  The layout of the CSV file has to fulfill the following criteria:
  1. The first line lists all nodes the content of CSV file belongs to.
  2. The second line contains the column headers. Each column contains the data of a dedicated statistics and element. Thus, the element name and the statistics name have to be encoded in the header of the each column. The supported header formats are:

     ElementName.StatisticsName
     ElementName.StatisticsName
     StatisticsName.ElementName
     StatisticsName.ElementName

     As you can see the statistics name can be placed in front or after the element name string separated by a dot (.) or not. The only exception of the rule is the column that contains the timestamp (see below).
  3. Each data row has to contain as many data values as columns defined by the column header.
  4. At least one column must contain timestamps and the column header has to match the wildcard string *Time or Time*.
  5. The max. length of a record in the CSV file may not exceed 32767 bytes.
  6. The rows of the CSV file have to be sorted ascending by the column that contains the timestamps.

7. A record descriptor must exist that describes the CSV file records –either in a descriptor file or in the CSV mapping database.
8. The time field of each record has to be unique throughout the whole CSV file.

- MULTI_LINE
  The layout of the CSV file has to fulfill the following criteria:
  1. The first line lists all nodes the content of CSV file belongs to.
  2. The second line contains the column headers - one header item per column. The maximum length of a header item is limited to 64 characters.
  3. One column header field has to be named TIME. This column contains the timestamps of the data records in OpenVMS date/time format.
  4. The maximum number of columns is limited to 200.
  5. The max. length of a record in the CSV file may not exceed 32767 bytes.
  6. The rows of the CSV file have to be sorted ascending by the TIME column.
  7. A record descriptor must exist that describes the CSV file records –either in a descriptor file or in the CSV mapping database.
  8. Several rows can co-exist with the same timestamp as long as the rows refer different elements. In that case 1 to max 3 columns must exist that defines the element key.Which columns are parts of the element key is defined in the mapping descriptor file (see the section Descriptor file of the MAP command description). With other words each data record has to be unique with respect to the element key or the timestamp.

- T4
  The CSV files were created by the T4 utility. The T4 format is similar to the SINGLE_LINE format. The only difference is that two extra header lines exist. Thus, a T4 formatted CSV file has to fulfil the criteria:
  1. The first line lists all nodes the content of CSV file belongs to.
  2. The fourth line contains the column headers. Each column contains the data of a dedicated statistics and element. Thus, the element name and the statistics name have to be encoded in the header of each column. The supported header formats are:

     ElementName.StatisticsName
     ElementName.StatisticsName
     StatisticsName.ElementName
     StatisticsName.ElementName

     As you can see the statistics name can be placed in front or after the element name string separated by a dot (.) or not. The

only exception of the rule is the column that contains the timestamp (see below).

3. Each data row has to contain as many data values as columns defined by the column header.
4. At least one column must contain timestamps and the column header has to match the wildcard string *Time or Time*.
5. The max. length of a record in the CSV file may not exceed 32767 bytes.
6. The rows of the CSV file have to be sorted ascending by the column that contains the timestamps.
7. A record descriptor must exist that describes the CSV file records –either in a descriptor file or in the CSV mapping database (see the Descriptor file section of the MAP command description).
8. The time field of each record has to be unique throughout the whole CSV file.

Any CSV file created by the T4 utility fulfils the criteria 1-6, 8. Thus, in order to import T4 files you have to create appropriate descriptor files. Such descriptor files will be provided by the next releases of PERFDAT.

For more information about valid CSV files see the MAP command description.

The IMPORT command is similar to the LOAD command. The main difference between the IMPORT and the LOAD command is that data imported by the IMPORT command are pre-processed before they are inserted. The LOAD command does not pre-process CSV data.

Using the IMPORT command CSV data are normalized before they are inserted into a collection database. This is done to guarantee that all statistical methods provided by the statistics package of the DQL interface can be applied to the imported data too.

It is very likely that the timestamps in the time column of a CSV file do not match the timestamps in the collection database. This is a prerequisite when correlating data. Any correlation based on data that does not match in time (timestamp of a sample, sample interval) will return wrong results. Normalizing means that based on the CSV data expectancy values are calculated for the timestamps of the collection database, and these expectancy values are inserted into the collection database. An integral based algorithm is used to normalize the data.

The file_name parameter specifies the full file name of the CSV files to be inserted. Asterisk (*) and per cent sign (%) wildcard characters can be placed anywhere within the file name string to import several CSV files at once.

The metric parameter of the AS clause specifies the name of the metric (table) to be created in the physical storage areas of the existing collection database. The CSV data will be stored within that metric (table).

The collection database to import the CSV data is specified by the TARGET DATABASE clause. Enter the collection database alias of the target database. That database alias can't be user-defined. DQL$ assigns these aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

The target collection database does not have to be attached before applying the IMPORT command. The target collection database will be attached automatically.

In order to import CSV file content a record descriptor is required that defines the (field) layout of the CSV file(s).

The optional DESCRIBED BY clause specifies the CSV descriptor file containing the record descriptor for the CSV files to insert. Depending if an entry with the name specified by the AS clause already exists in the CSV mapping database PERFDAT$CFG:CSV_PROFILES.CFG, this clause can be omitted or not. If you omit this clause DQL$ searches for the entry defined by the AS clause in the CSV mapping database and uses the record descriptor stored in that entry to insert the CSV data. If no such entry exists the command fails.

For detailed information about record descriptors and descriptor files please see the Descriptor file section of the MAP command description.

Before applying the IMPORT command verify that the default regional setting (format of numbers, date, time and list separator) of the current DQL$ session matches the CSV file format. If it does not use the DEFINE REGION and SET REGION command to create a matching default regional setting. For more information about regional settings please refer to the DEFINE REGION and SET REGION command description.

Before DQL$ imports the data it performs some general checks:

- It checks if the target collection database exist. If it does not exist the command fails.
- It checks if the target database contains data. If this is not the case the command fails. Remember that the IMPORT command normalizes CSV data by calculating expectancy values for the timestamps of the collection database. Thus, if no timestamps exist in the target database

no data can be normalized. Consequently the IMPORT command fails if you want to insert CSV data into an empty collection database manually created by the CREATE command. In this case use the LOAD command to insert the CSV data.

In addition the following checks are performed per CSV file to import:

- It checks if the records of the CSV file are sorted ascending by the TIME column. If this not the case the data of the CSV file will not be imported.
- It checks if the target collection database specified by the TARGET DATABASE clause refers a node that is listed in the first line of the CSV file. If this not the case the CSV file is not valid for the node and the data are not imported.
- It checks if the metric defined by the AS clause already exists in the physical storage areas of the target collection database. If this is the case and the record descriptor does not match the existing metric definition the command fails.
- It checks if data exist in the target database for the time range defined by the timestamps in the TIME column of the CSV file. If the CSV file contains records with a timestamp that are outside the time range of the data stored in the target collection database these records of the CSV file are not imported. Remember that the IMPORT command normalizes CSV data by calculating expectancy values for the timestamps of the collection database. Thus, this can be done only if corresponding timestamps exist in the target database.
- It checks if the shortest time interval defined by the timestamps in the CSV file is within the trusted range. The trusted range is 3 times the sample interval of the target database. If this is not the case no data will be imported since it makes no sense to import CSV data into a collection database with a sample interval that is much greater than the sample interval of the collection that created the target collection database.
- If checks if the list separator of the default regional setting of the current DQL$ session is found in the CSV file to load. If this is not the case the default regional setting does not match the format of the CSV file and the command fails.

## Example

This example shows how to import existing CSV files into the existing collection database BCSXTC_DEFAULT created by a collection started with the collection profile DEFAULT on node BCSXTC. The layout of the input CSV files is MULTI_LINE. Thus, the FORMAT clause is entered with the keyword MULTI_LINE.

```
DQL>IMPORTPERFDAT$DB_ARCHIVE:*.CSV AS SPHINX
cont>DESCRIBED BY PERFDAT$DB_ARCHIVE:SPHINX_DSC.CFG
cont>TARGET DATABASE BCSXTC_DEFAULT;
cont> FORMAT MULTI_LINE;
```

```
DQL-I-IMPORT, importing data from file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_AUG_2000.CSV;2/
DQL-I-GETELEM, fetching elements from source file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_AUG_2000.CSV;2/
DQL-I-IMPORT, importing Metrix /SPHINX/ in storage area /BCSXTC_DEFAULT_2005-08-25:00:00:00:1/
DQL-I-STARTLOAD, start importing data -> element count: 1
DQL-I-IMPORT, file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_AUG_2000.CSV;2/successfully imported

DQL-I-IMPORT, importing data from file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_15_JUL_2000.CSV;6/
DQL-I-GETELEM, fetching elements from source file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_15_JUL_2000.CSV;6/
DQL-W-IMPORT, time range of the data stored CSV file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_15_JUL_2000.CSV;6/ not covered by target database
/BCSXTC_DEFAULT/
DQL-W-IMPORT, file /PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_15_JUL_2000.CSV;6/ not imported

DQL-I-IMPORT, importing data from file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_NOV_2003.CSV;13/
DQL-I-GETELEM, fetching elements from source file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_NOV_2003.CSV;13/
DQL-W-IMPORT, time range of the data stored CSV file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_NOV_2003.CSV;13/ not covered by target database
/BCSXTC_DEFAULT/
DQL-W-IMPORT, file /PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_NOV_2003.CSV;13/ not imported
```

Due to the statement applied all CSV files of the directory PERFDAT$DB_ARCHIVE are selected to be imported into the collection database BCSTC_DEFAULT. Since the only file that contains records with valid timestamps (within time range of the target database) is PERFDAT$DB_ARCHIVE:SXWS_CL0_25_AUG_2005.CSV, this is the only one that is actually imported. The file import of all other files is denied.

## *INSERT*

Insert data fields of a record or the whole record into an existing metric of a physical storage area.

### Format

```
INSERT INTO metric_name
        ALIAS alias_name [DATE] date
                (statistics_itemlist)
                        VALUE (value_list);

or

INSERT INTO metric_name
        'FILE filename_alias'
                (statistics_itemlist)
                        VALUE (value_list);
```

### Description

Insert data fields of a record or the whole record into an existing metric of a physical storage area.

The INTO clause specifies the metric to insert the records. The metric must exist in the physical storage areas defined by the ALIAS/DATE or FILE clause.

The target data file to insert the record can be addressed either by the ALIAS/DATE clause or the FILE clause. The FILE and ALIAS/DATE clause are mutual exclusive. If the FILE clause in used the INSERT command tries to insert the record (or parts of it) into the physical storage area referenced by the filename_alias. If the ALIAS/DATE clauses are used DQL$ searches the physical storage area within the collection database/logical storage area addressed by the ALIAS/DATE clauses to insert the record.

The ALIAS clause specifies the alias of the target collection database to insert the record (or parts of it).That database alias can't be user-defined. DQL$ assigns these aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

If you apply the optional DATE clause the query addresses a single logical storage area. Use OpenVMS date format to define the day of interest (a logical storage area is the sum of all physical storage areas created on the same day). If you omit the DATE clause all data files (physical storage areas) of the collection database defined by the ALIAS clause are searched to insert the record.

The access mode to the physical storage area(s) defined by the ALIAS/DATE or FILE clause has (have) to be READ WRITE. Use the SET TRANSACTION command to grant READ WRITE access to the data file(s). Otherwise the INSERT command fails.

The statistics_itemlist defines the statistics (data fields) of the record to insert. Enter the statistics as a comma separated list. Each of the statistics listed have to exist in the metric defined by the INTO clause. To display the statistics available for a particular metric apply the SHOW STATISTICS command.

The VALUE clause contains the values of the statistics listed in the statistics item list, as a comma separated list. The first item of the value list is assigned to the first item of the statistics list. The second item of the value list is assigned to the second item of the statistics list and so on. The value list has to contain as many values as the statistics item list.

At minimum all data fields that are defined as part of the element key (apply the SHOW STATISTICS command to see which data fields (statistics) are defined as element key data fields) and the TIME statistics (date/time format is dd-mmm-yyyy hh:mm:ss) have to be defined. The TIME statistics is part of any valid metric stored in the data files of the distributed performance database (see the CREATE METRIX command description). The value assigned to the TIME statistics is used by the INSERT command to validate the physical storage are to insert the record. Records can be inserted to a physical storage area if the timestamp of the record (TIME data field) is covered by the time range the physical storage area is valid for (see also the CREATE STORAGE AREA command description).

No duplicate inserts are allowed for data files of the PERFDAT distributed database. Thus, if you insert a record containing a timestamp and an element key already stored in the target physical storage area the INSERT command fails due to duplicates.

All statistics of the metric defined by the INTO clause not listed in the statistics item list are automatically zeroed.

## Example

This example shows how to insert a record into the metric SPHINX. This metric exists (see CREATE METRIX command example) within the collection database SPHINX_DEFAULT previously created (see CREATE STORAGE AREA command example).

Attach the target database:

```
DQL> ATTACH ALIAS SPHINX_DEFAULT;
DQL-I-ATTACH, successfully attached file
/SPHINX_DEFAULT_2005-08-25:00:00:00:1/
```

Set access mode:

```
DQL> SET TRANSACTION READ WRITE ON ALIAS SPHINX_DEFAULT;
DQL-I-SET, file /SPHINX_DEFAULT_2005-08-25:00:00:00:1/ will be READ WRITE
accessible
```

Insert record:

```
DQL> INSERT INTO SPHINX ALIAS SPHINX_DEFAULT
cont> (TIME, PART9 Ges, PART10 Ges, PART11 Ges, PART12 Ges, Request Ges)
cont> VALUE (25-AUG-200513:40, 1.2, 2.0, 3.1, 0.3, 6.5);
DQL-I-INSERT, successfully inserted record into Metrix /SPHINX/
```

The metric SPHINX consists of more statistics than applied by the INSERT command.

```
DQL> SHOW STATISTICS FROM SPHINX ALIAS SPHINX_DEFAULT;

SPHINX METRIX DEFINITION of storage area SPHINX_DEFAULT_2005-08-25:00:00:00:1

---------------------------------------------------------------------------------------------------------------
 Field definitions of Metrix: SPHINX

        FieldName   Type          Description
        ---------   ----          -----------

        Time                      DATETIME(8)       Time
        PART9 Ges                 FLOAT(4)          Partion 9 Gesamt TA
        PART9 ACK                 FLOAT(4)          Partion 9 Ack TA
        PART9 NAK                 FLOAT(4)          Partion 9 Nack TA
        PART10 Ges                FLOAT(4)          Partion 10 Gesamt TA
        PART10 ACK                FLOAT(4)          Partion 10 Ack TA
        PART10 NAK                FLOAT(4)          Partion 10 Nack TA
        PART11 Ges                FLOAT(4)          Partion 11 Gesamt TA
        PART11 ACK                FLOAT(4)          Partion 11 Ack TA
        PART11 NAK                FLOAT(4)          Partion 11 Nack TA
        PART12 Ges                FLOAT(4)          Partion 12 Gesamt TA
        PART12 ACK                FLOAT(4)          Partion 12 Ack TA
        PART12 NAK                FLOAT(4)          Partion 12 Nack TA
        Request Ges               FLOAT(4)          Gesamt TA
        GES ACK                   FLOAT(4)          Gesamt Ack TA
        GES NAK                   FLOAT(4)          Gesamt Nack TA

Element count    1
---------------------------------------------------------------------------------------------------------------
```

Thus the statistics:

- PART9 ACK
- PART9 NAN
- PART10 ACK
- PART10 NAK
- PART11 ACK
- PART11 NAK

- PART12 ACK
- PART12 NAK
- Ges ACK
- Ges NAK

are all zeroed automatically.

## *LIST METRIX*

This command displays all the performance metrics (tables) stored in the record descriptor table of the PERFDAT configuration database.

### Format

LIST METRIX metric_name [OSTYPE OS_name];

### Description

The LIST METRIX command displays all the performance metrics (tables) stored in the record descriptor table of the PERFDAT configuration database.

The record descriptor table of the PERFDAT configuration database contains the field descriptors of all predefined performance metrics available for any system supported by PERFDAT. For more information about the record descriptor table of the PERFDAT configuration database please refer to the manual VSI PERFDAT – Architecture and Technical Description.

The metric_name parameter defines the filter for the performance metrics to be displayed. Full wildcard support is provided. Asterisk (*) and per cent sign (%) wildcard characters can be placed anywhere within the metric_name string. If you omit the metric_name parameter a full wildcard operation is performed.

The OSTYPE clause is optional and can be used to filter for performance metrics of a specific system. The record descriptor table of the PERFDAT configuration database contains predefined metrics for:

- OpenVMS systems
- Tru64 systems
- EVA storage arrays
- Brocade switches
- Solaris Systems
- Linux systems
- RDB databases
- CACHE databases
- If the VSI PERFDAT API is used by any application the record descriptors of the metrics accessed by these applications

Thus, in order to display the predefined metrics of only one of these systems listed above apply one these keywords in the OSTYPE clause:

- OpenVMS
- Tru64
- EVA
- Brocade

- Solaris
- Linux
- RDB
- CACHE
- Name of the application that uses the VSI PERFDAT API

## Examples

Example 1

In this example all predefined metrics stored in the record descriptor table of the PERFDAT configuration database are displayed.

```
DQL> LIST METRIX;

        Predefined Metrics for OS: EVA
-------------------------------------------------
        ARRAY
        CTRL
        CTRL.HOSTCONN
        CTRL.PORT
        DISKGROUP
        DISKGROUP.PDISK
        DISKGROUP.VDISK
        DRM.TUNNEL

        Predefined Metrics for OS: RDB
-------------------------------------------------
        CACHE
        CACHE.UNMARK
        INDEX.HASH
        INDEX.INSERTION
        INDEX.REMOVAL
        INDEX.RETRIEVAL
        IO.ASYNCH_IO
        IO.FETCH
        IO.FILE
        IO.PREFETCH
        IO.STALL_IO
        JOURNAL.2PC
        JOURNAL.AIJ
        JOURNAL.ALS
        JOURNAL.DBR
        JOURNAL.RUJ
        LOCK.TYPE
        LOGNAM
        OBJECT.TYPE
        RECORD
        SNAPSHOT
        STALLS
        TRANS
        TRANS.HISTOGRAMM
```

```
        Predefined Metrics for OS: CACHE
-------------------------------------------------------
        CACHE

        Predefined Metrics for OS: LINUX
-------------------------------------------------------
        LINUX_DEAMON
        LINUX_FILESYS
        LINUX_IP
        LINUX_NIC
        LINUX_PROCESS
        LINUX_SYSTEM
        LINUX_TCP

        Predefined Metrics for OS: TRU64
-------------------------------------------------------
        TRU64_CPU
        TRU64_DEAMON
        TRU64_DISK
        TRU64_FILESYS
        TRU64_IP
        TRU64_NIC
        TRU64_PROCESS
        TRU64_SYSTEM
        TRU64_USER

        Predefined Metrics for OS: BROCADE
-------------------------------------------------------
        PORT
        SYSTEM

        Predefined Metrics for OS: OPENVMS
-------------------------------------------------------
        ACCOUNT
        CPU
        DEVICE
        DEVICE.CAPACITY
        DEVICE.FILE
        DEVICE.IOSIZE
        DEVICE.PATH
        DEVICE.PROCESS
        DEVICE.PROCESS.FILE
        IMAGE
        IOPATHES
        LANADAPTER
        LANADAPTER.DEVICE
        LANPROTOCOL
        PROCESS
        SCSPORT
        SCSPORT.VC
        SCSPORT.VC.CHANNEL
        SYSTEM
        USER
        XFCVOLUME
        XFCVOLUME.FILE
        XFCVOLUME.FILE.IOSIZE
        XFCVOLUME.IOSIZE
```

```
        Predefined Metrics for OS: SOLARIS
-------------------------------------------------
        SUN_DEAMON
        SUN_DEVICE
         SUN_FILESYS
         SUN_IP
        SUN_NIC
        SUN_PROCESS
        SUN_SYSTEM
        SUN_TCP
```

Example 2

In this example all predefined device metrics for OpenVMS stored in the record descriptor table of the PERFDAT configuration database are displayed.

```
DQL> LIST METRIX ALIAS DEVICE* OSTYPE OpenVMS;

        Predefined Metrics for OS: OPENVMS
-------------------------------------------------------------
        DEVICE
        DEVICE.CAPACITY
        DEVICE.FILE
        DEVICE.IOSIZE
        DEVICE.PATH
        DEVICE.PROCESS
        DEVICE.PROCESS.FILE
```

## *LIST STATISTICS*

The LIST STATISTICS command displays the statistics stored in the record description table of the PERFDAT configuration database and the user-defined statistics (stored procedures) of a particular performance metric (table). The field name, datatype, field length and the field description is displayed.

### Format

```
LIST STATISTICS FROM metric_name
                        OSTYPE OS_name [NODE] node_name;
```

### Description

The LIST STATISTICS command displays the statistics stored in the record description table of the PERFDAT configuration database and the user-defined statistics (stored procedures) stored in its stored procedure table of a particular performance metric (table). The field name, datatype, field length and the field description is displayed.

The record descriptor table of the PERFDAT configuration database contains the field descriptors of all predefined performance metrics available for any system supported by PERFDAT. The stored procedure table of the PERFDAT configuration database contains all user-defined statistics and the associated parameters configured by the user via the DQL$ utility. User-defined statistics are calculated values that can be accessed by all users that are connected to access servers that share the same PERFDAT configuration database as if these statistics are part of a collection database. For more information about the record descriptor table und stored procedure table of the PERFDAT configuration database please refer to the manual VSI PERFDAT – Architecture and Technical Description.

The OSTYPE clause defines the system type the metric defined by the metric_name parameter is member of. The record descriptor table of the PERFDAT configuration database contains predefined metrics for:

- OpenVMS systems
- Tru64 systems
- EVA storage arrays
- Brocade switches
- Solaris Systems
- Linux systems
- RDB databases
- CACHE databases
- If the VSI PERFDAT API is used by any application the record descriptors of the metrics accessed by these applications

Thus, valid keywords for the OS_name parameter are:

- OpenVMS
- Tru64
- EVA
- Brocade
- Solaris
- Linux
- RDB
- CACHE
- Name of the application that uses the VSI PERFDAT API

The NODE clause is optional. It can be applied to display the statistics of a particular metric valid for a specific node. The LIST STATISTICS displays the statistics of a particular metric stored in the record descriptor table of the PERFDAT configuration database as well as the user-defined statistics (stored procedures) stored in its stored procedure table. User-defined statistics can be defined generic for a specific system (OSTYPE) as well as node specific. If you omit the NODE clause the generic defined user-defined statistics are displayed. If you define a specific node using the NODE clause the generic and the node specific user-defined statistics valid for the node you applied are displayed.

For more information about user-defined statistics please refer to the DEFINE PROCEDURE command description.

## Examples

Example 1

In this example the statistics of the process metric of OpenVMS are displayed. Since the NODE clause is omitted only generic user statistics are displayed.

```
DQL> LIST STATISTICS FROM PROCESS OSTYPE OpenVMS;

--------------------------------------------------------------------------------
Field definitions of Metric: PROCESS

FieldName     Type                Description
------------  -------             -------------

PrcName       STRING(32)   [P]    ProcessName
$iCpuNorm     FLOAT(4)     [C]    CPU Load normalized [0..100%]
Time          DATETIME(8)         Time
UserName      STRING(16)   [I]    User Name Reference
ImageName     STRING(256)  [I]    Image Name Reference
iDIO          FLOAT(4)            Direct IO rate
iBIO          FLOAT(4)            Buffered IO rate
iGlbMem       FLOAT(4)            Gbl Memory allocated by image
iPrcMem       FLOAT(4)            Private Memory allocated by image
```

| | | |
|---|---|---|
| iPfl | FLOAT(4) | PFL total |
| iPflFOR | FLOAT(4) | PFL on read faults |
| iPflFOW | FLOAT(4) | PFL on write faults |
| iPflFOE | FLOAT(4) | PFL on executive fault |
| iPageIO | FLOAT(4) | IO PageIOs |
| iCpuLoad | FLOAT(4) | CPU Load total |
| iKernel | FLOAT(4) | CPU Mode kernel |
| iExec | FLOAT(4) | CPU Mode exec |
| iSuper | FLOAT(4) | CPU Mode super |
| iUser | FLOAT(4) | CPU Mode user |
| iIOthres | FLOAT(4) | IO request threshold |
| iMemthres | FLOAT(4) | Memory usage threshold |
| iCputhres | FLOAT(4) | CPU load threshold |

Element count 0

---

All fields marked with [P] are members of the element key (index), and the fields marked with [C] are user-defined statistics.All fields marked with [I] are informational fields. These fields are not visible to the GUI.

In the exampleone user-defined statistics is listed since only one generic user-defined statistics exists in the stored procedure table of the PERFDAT configuration database for the process metric of OpenVMS.

DQL> SHOW PROCEDURE * METRIX PROCESS OSTYPE OPENVMS;

Generic Stored Procedures valid for all nodes of OS Type: OPENVMS

Metrix: PROCESS $iCpuNorm = iCpuLoad / iCpus
Dscr: CPU load normalized [0..100%], Unit: [%]

Node specific stored Procedures valid for OS Type: OPENVMS

Metrix: PROCESS
Node: VMSTM1 $iExecNorm= iExec / iCpus
Dscr: CPU exec mode normalized [0..100%], Unit: [%]
Node: VMSTM1 $iUserNorm= iUser / iCpus
Dscr: CPU user mode normalized [0..100%], Unit: [%]

Example 2

As you can see from the output of the SHOW PROCEDURE command in example 1 there exists one generic and two node specific user statistics for node VMSTM1. Thus, if you recall the LIST STATISTICS command from example 1 and additionally apply VMSTM1 in the NODE clause all three user-defined statistics are displayed.

```
DQL> LIST STATISTICS FROM PROCESS OSTYPE OpenVMS NODE VMSTM1;

--------------------------------------------------------------------------------
Field definitions of Metric: PROCESS

FieldName       Type                Description
-------------   -------             --------------

PrcName         STRING(32)   [P]    ProcessName
$iCpuNorm       FLOAT(4)     [C]    CPU Load normalized [0..100%]
$iExecNorm      FLOAT(4)     [C]    CPU exec mode normalized [0..100%]
$iUserNorm      FLOAT(4)     [C]    CPU user mode normalized [0..100%]
Time            DATETIME(8)         Time
UserName        STRING(16)   [I]    User Name Reference
ImageName       STRING(256)  [I]    Image Name Reference
iDIO            FLOAT(4)            Direct IO rate
iBIO            FLOAT(4)            Buffered IO rate
iGlbMem         FLOAT(4)            Gbl Memory allocated by image
iPrcMem         FLOAT(4)            Private Memory allocated by image
iPfl            FLOAT(4)            PFL total
iPflFOR         FLOAT(4)            PFL on read faults
iPflFOW         FLOAT(4)            PFL on write faults
iPflFOE         FLOAT(4)            PFL on executive fault
iPageIO         FLOAT(4)            IO PageIOs
iCpuLoad        FLOAT(4)            CPU Load total
iKernel         FLOAT(4)            CPU Mode kernel
iExec           FLOAT(4)            CPU Mode exec
iSuper          FLOAT(4)            CPU Mode super
iUser           FLOAT(4)            CPU Mode user
iIOthres        FLOAT(4)            IO request threshold
iMemthres       FLOAT(4)            Memory usage threshold
iCputhres       FLOAT(4)            CPU load threshold

Element count   0
--------------------------------------------------------------------------------
```

All fields marked with [P] are members of the element key (index), and the fields marked with [C] are user-defined statistics.All fields marked with [I] are informational fields. These fields are not visible to the GUI.

## *LOAD*

This command loads data of valid CSV files into an existing collection database.

### Format

```
LOAD file_name AS metric
                [DESCRIBED BY descriptor_filename]
                                TARGET DATABASE alias_name
                                [FORMAT] { SINGLE_LINE | MULTI_LINE };
```

### Description

This command loads data of valid CSV files into an existing collection database.

The content of CSV files with two different layouts can be loaded (see also the EXPORT command description). The optional FORMAT clause defines the layout of the input CSV files addressed by the MAP command. Valid keywords are SINGLE_LINE and MULTI_LINE. If the FORMAT clause is omitted the default layout format is SINGLE_LINE:

- SINGLE_LINE
  The layout of the CSV file has to fulfill the following criteria:
  1. The first line lists all nodes the content of CSV file belongs to.
  2. The second line contains the column headers. Each column contains the data of a dedicated statistics and element. Thus, the element name and the statistics name have to be encoded in the header of the each column. The supported header formats are:

     ```
     ElementName.StatisticsName
     ElementName.StatisticsName
     StatisticsName.ElementName
     StatisticsName.ElementName
     ```

     As you can see the statistics name can be placed in front or after the element name string separated by a dot (.) or not. The only exception of the rule is the column that contains the timestamp (see below).
  3. Each data row has to contain as many data values as columns defined by the column header.
  4. At least one column must contain timestamps and the column header has to match the wildcard string *Time or Time*.
  5. The max. length of a record in the CSV file may not exceed 32767 bytes.
  6. The rows of the CSV file have to be sorted ascending by the column that contains the timestamps.

7. A record descriptor must exist that describes the CSV file records –either in a descriptor file or in the CSV mapping database.
8. The time field of each record has to be unique throughout the whole CSV file.

- MULTI_LINE
  The layout of the CSV file has to fulfill the following criteria:
  1. The first line lists all nodes the content of CSV file belongs to.
  2. The second line contains the column headers - one header item per column. The maximum length of a header item is limited to 64 characters.
  3. One column header field has to be named TIME. This column contains the timestamps of the data records in OpenVMS date/time format.
  4. The maximum number of columns is limited to 200.
  5. The max. length of a record in the CSV file may not exceed 32767 bytes.
  6. The rows of the CSV file have to be sorted ascending by the TIME column.
  7. A record descriptor must exist that describes the CSV file records –either in a descriptor file or in the CSV mapping database.
  8. Several rows can co-exist with the same timestamp as long as the rows refer different elements. In that case 1 to max 3 columns must exist that defines the element key. Which columns are parts of the element key is defined in the mapping descriptor file (see the section Descriptor file of the MAP command description). With other words each data record has to be unique with respect to the element key or the timestamp.

- T4
  The CSV files were created by the T4 utility. The T4 format is similar to the SINGLE_LINE format. The only difference is that two extra header lines exist. Thus, a T4 formatted CSV file has to fulfil the criteria:
  1. The first line lists all nodes the content of CSV file belongs to.
  2. The fourth line contains the column headers. Each column contains the data of a dedicated statistics and element. Thus, the element name and the statistics name have to be encoded in the header of each column. The supported header formats are:

     ElementName.StatisticsName
     ElementName.StatisticsName
     StatisticsName.ElementName
     StatisticsName.ElementName

     As you can see the statistics name can be placed in front or after the element name string separated by a dot (.) or not. The

only exception of the rule is the column that contains the timestamp (see below).

3. Each data row has to contain as many data values as columns defined by the column header.
4. At least one column must contain timestamps and the column header has to match the wildcard string *Time or Time*.
5. The max. length of a record in the CSV file may not exceed 32767 bytes.
6. The rows of the CSV file have to be sorted ascending by the column that contains the timestamps.
7. A record descriptor must exist that describes the CSV file records –either in a descriptor file or in the CSV mapping database (see the Descriptor file section of the MAP command description).
8. The time field of each record has to be unique throughout the whole CSV file.

Any CSV file created by the T4 utility fulfils the criteria 1-6, 8. Thus, in order to load T4 files you have to create appropriate descriptor files. Such descriptor files will be provided by the next releases of PERFDAT.

For more information about valid CSV files see the MAP command description.

The LOAD command is similar to the IMPORT command. The main difference between the LOAD and the IMPORT command is that data loaded by the LOAD command are not pre-processed before they are inserted.

The LOAD command should only be used if the timestamps in the CSV file(s) match exactly the timestamps stored in the target collection database, or the CSV data shall be inserted to an empty collection database manually created. Otherwise it is recommended to use the IMPORTcommand. The LOAD command consumes less system resources and is faster than the IMPORTcommand.

The file_name parameter specifies the full file name of the CSV files to be inserted. Asterisk (*) and percent sign (%) wildcard characters can be placed anywhere within the file name string to load several CSV files at once.

The metric parameter of the AS clause specifies the name of the metric (table) to be created in the physical storage areas of the existing collection database. The CSV data will be stored within that metric (table).

The collection database to load the CSV data is specified by the TARGET DATABASE clause. Enter the collection database alias of the target database. That database alias can't be user-defined. DQL$ assigns these aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

The target collection database does not have to be attached before applying the LOAD command. The target collection database will be automatically attached.

In order to load CSV file content a record descriptor is required that defines the (field) layout of the CSV file(s).

The optional DESCRIBED BY clause specifies the CSV descriptor file containing the record descriptor for the CSV files to insert. Depending if an entry with the name specified by the AS clause already exists in the CSV mapping database or a metric with the same name exists in the PERFDAT configuration database, this clause can be omitted or not. If you omit this clause DQL$ searches for the metric descriptor defined by the AS clause in the record descriptor table of the PERFDAT configuration database. If no such metric descriptor exists DQL$ searches the CSV mapping database for a matching entry. If both checks fail the LOAD command fails.

For detailed information about record descriptors and descriptor files please see the descriptor file section of the MAP command description.

Before applying the LOAD command verify that the default regional setting (format of numbers, date, time and list separator) of the DQL$ session matches the CSV file format. If it does not use the DEFINE REGION and SET REGION command to create a matching default regional setting. For more information about regional settings please refer to the DEFINE REGION and SET REGION command description.

Before DQL$ loads the data it performs the general check:

- It checks if the target collection database exist. If it does not exist the command fails.

In addition the following checks are performed per CSV file to import:

- It checks if the records of the CSV file are sorted ascending by the TIME column. If this not the case the data of the CSV file will not be loaded.
- It checks if the target collection database specified by the TARGET DATABASE clause refers a node that is listed in the first line of the CSV file. If this not the case the CSV file is not valid for the node and the data are not loaded.
- It checks if the metric defined by the AS clause already exists in the physical storage areas of the target collection database. If this is the case and the record descriptor does not match the existing metric definition the command fails.

- It checks if physical storage areas of the target collection database exist that cover the time range or parts of it defined by the timestamps in the TIME column of the CSV file. It does not matter if the physical storage areas contain data or not, but they have to exist. Otherwise no record is loaded.
- If checks if the list separator of the default regional setting of the current DQL$ session is found in the CSV file to load. If this is not the case the default regional setting does not match the format of the CSV file and the command fails.

## Example

This example shows how to load existing CSV files into an existing collection database. The target database SPHINX_DEFAULT consists of the logical storage area 25-AUG-2005 which in turn consists of one physical storage area previously created (see CREATE STORAGE AREA command example). The metric (table) SPHINX has been added to the physical storage area using the CREATE METRIX command (see CREATE METRIX command example). The layout of the input CSV files is MULTI_LINE. Thus, the FORMAT clause is entered with the keyword MULTI_LINE.

```
DQL> LOAD PERFDAT$DB_ARCHIVE:*.CSV AS SPHINX
cont> DESCRIBED BY PERFDAT$DB_ARCHIVE:SPHINX_DSC.CFG
cont> TARGET DATABASE SPHINX_DEFAULT
cont> FORMAT MULTI_LINE;

DQL-I-LOAD, loading data from file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_AUG_2000.CSV;2/
DQL-I-GETELEM, fetching elements from source file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_AUG_2000.CSV;2/
DQL-I-LOAD, loading Metrix /SPHINX/ in storage area /SPHINX_DEFAULT_2005-08-25:00:00:00:1/
DQL-I-CREATEMETRIX, metrix already exists
DQL-I-STARTLOAD, start loading data -> element count: 1
DQL-I-LOAD, file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_AUG_2000.CSV;2/successfully loaded

DQL-I-LOAD, loading data from file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_15_JUL_2000.CSV;6/
DQL-I-GETELEM, fetching elements from source file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_15_JUL_2000.CSV;6/
DQL-W-LOAD, time range of the data stored CSV file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_15_JUL_2000.CSV;6/ not covered by target database
/SPHINX_DEFAULT/
DQL-W-LOAD, file /PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_15_JUL_2000.CSV;6/ not loaded

DQL-I-LOAD, loading data from file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_NOV_2003.CSV;13/
DQL-I-GETELEM, fetching elements from source file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_NOV_2003.CSV;13/
DQL-W-LOAD, time range of the data stored CSV file
/PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_NOV_2003.CSV;13/ not covered by target database
/SPHINX_DEFAULT/
DQL-W-LOAD, file /PERFDAT$COMMON:[ARCHIVE]SXWS_CL0_25_NOV_2003.CSV;13/ not loaded
```

Due to the statement applied all CSV files of the directory PERFDAT$DB_ARCHIVE are selected to be loaded into the collection database SPHINX_DEFAULT. Since the only file that contains records with valid timestamps (within time range of the target database) is PERFDAT$DB_ARCHIVE:SXWS_CL0_25_AUG_2005.CSV, this is the only one that is actually loaded. The file load of all other files is denied.

## *MAP*

This command adds a new entry to the CSV mapping database

### Format

```
MAP FILE file_name
        TO COLLECTION collection_profile
                AS metrix
                        [REGION reg_name]
                        [DESCRIBED BY descriptor_file]
                        [FORMAT { SINGLE_LINE | MULTI_LINE }];
```

### General description

This command adds a new entry to the CSV mapping database.

The DQL interface provides the feature to map CSV file content online. Content mapping means, that the data stored in a particular CSV file is accessible via the DQL$ interface in the same manner as if data were originally written by the OpenVMS data collector or the SNMP extension.

CSV file content mapping is done by adding appropriate entries to the CSV mapping database PERFDAT$CFG:CSV_PROFILE.CFG. These entries are read by DQL$SRV which actually maps the data of the CSV file.

This feature provides the ability to include foreign data(= not written by the data collectors) easily.

CSV files with two different layouts can be mapped (see also the EXPORT command description). The FORMAT clause defines the layout of the input CSV files addressed by the MAP command. Valid keywords are SINGLE_LINE and MULTI_LINE. If the FORMAT clause is omitted the default layout format is SINGLE_LINE:

- SINGLE_LINE
  The layout of the CSV file has to fulfill the following criteria:
  1. The first line lists all nodes the content of CSV file belongs to.
  2. The second line contains the column headers. Each column contains the data of a dedicated statistics and element. Thus, the element name and the statistics name have to be encoded in the header of each column. The supported header formats are:

     ElementName.StatisticsName
     ElementName.StatisticsName
     StatisticsName.ElementName

StatisticsName.ElementName

As you can see the statistics name can be placed in front or after the element name string separated by a dot (.) or not. The only exception of the rule is the column that contains the timestamp (see below).

3. Each data row has to contain as many data values as columns defined by the column header.
4. At least one column must contain timestamps and the column header has to match the wildcard string *Time or Time*.
5. The max. length of a record in the CSV file may not exceed 32767 bytes.
6. The rows of the CSV file have to be sorted ascending by the column that contains the timestamps.
7. A record descriptor must exist that describes the CSV file records –either in a descriptor file or in the CSV mapping database (see the Descriptor file section).
8. The time field of each record has to be unique throughout the whole CSV file.

- MULTI_LINE
  The layout of the CSV file has to fulfill the following criteria:
  1. The first line lists all nodes the content of CSV file belongs to.
  2. The second line contains the column headers - one header item per column. The maximum length of a header item is limited to 64 characters.
  3. One column header field has to be named TIME. This column contains the timestamps of the data records in OpenVMS date/time format.
  4. The maximum number of columns is limited to 200.
  5. The max. length of a record in the CSV file may not exceed 32767 bytes.
  6. The rows of the CSV file have to be sorted ascending by the TIME column.
  7. A record descriptor must exist that describes the CSV file records –either in a descriptor file or in the CSV mapping database.
  8. Several rows can co-exist with the same timestamp as long as the rows refer different elements. In that case 1 to max 3 columns must exist that defines the element key.Which columns are parts of the element key is defined in the mapping descriptor file (see the Descriptor file section). With other words each data record has to be unique with respect to the element key or the timestamp.

- T4
  The CSV files were created by the T4 utility. The T4 format is similar to the SINGLE_LINE format. The only difference is that two extra header lines exist. Thus, a T4 formatted CSV file has to fulfil the criteria:

1. The first line lists all nodes the content of CSV file belongs to.
2. The fourth line contains the column headers. Each column contains the data of a dedicated statistics and element. Thus, the element name and the statistics name have to be encoded in the header of each column. The supported header formats are:

   ElementName.StatisticsName
   ElementName.StatisticsName
   StatisticsName.ElementName
   StatisticsName.ElementName

   As you can see the statistics name can be placed in front or after the element name string separated by a dot (.) or not. The only exception of the rule is the column that contains the timestamp (see below).
3. Each data row has to contain as many data values as columns defined by the column header.
4. At least one column must contain timestamps and the column header has to match the wildcard string *Time or Time*.
5. The maximum length of a record in the CSV file may not exceed 32767 bytes.
6. The rows of the CSV file have to be sorted ascending by the column that contains the timestamps.
7. A record descriptor must exist that describes the CSV file records –either in a descriptor file or in the CSV mapping database (see the Descriptor file section).
8. The time field of each record has to be unique throughout the whole CSV file.

Any CSV file created by the T4 utility fulfils the criteria 1-6, 8. Thus, in order to map T4 files you have to create appropriate descriptor files. Such descriptor files will be provided by the next releases of PERFDAT.

It does not matter if the CSV file includes data of different days. DQL splits the file virtually in order to map the CSV file content to the database scheme.

Mapped CSV files are only read accessible.

CSV data content can't be correlated with other CSV file content or data content created by the OpenVMS data collector or SNMP extension. If you want to use the correlation feature you have to import/load CSV file content into an existing collection database. CSV files are not processed by the auto archiving process. Data management of mapped CSV files is left to system management.

CSV file mappings are only valid on the node where the mapping is done but the CSV content can be accessed by any member of the community the node that hosts the CSV file belongs to in case the node(s) listed in the first line of the CSV file are also member(s) of the community.

## Statement description

This command adds a new entry to the CSV mapping database

The file_name parameter specifies the full file name of the CSV files to be mapped. Asterisk (*) and percent sign (%) wildcard characters can be placed anywhere within the file name string to map several CSV files at once.

The TO COLLECTION clause specifies the collection profile the CSV files are mapped to. If more than one node is listed in the first line of the CSV files, the CSV files becomes member of different collection databases. E.g. the collection profile specified is 60S, and the nodes defined in the first line of the CSV files are SIFNOS, VMSTM1. The CSV files become member of the collection databases SIFNOS_60S and VMSTM1_60S. It is not required that the collection profile specified exists in the collection profile table of the PERFDAT configuration database. You can enter any name. If you choose an existing profile name the CSV mapping will become a member (physical storage area) of existing collection database(s).If you enter a non-existing profile name the CSV content will be visible as a standalone collection database.

The metrix parameter in the AS clause specifies the virtual metric (table) DQL$ uses to address the CSV file content and the entry in the CSV file mapping database.

The reg_name parameter in the optional REGION clause defines the regional setting that matches the format of the CSV file.. A regional setting defines the format of numbers, date and time in the CSV file and the list separator used. If you omit the REGION clause the default regional setting on the local host is used. If the regional setting does not match the format of the CSV content the result of the mapping will be unpredictable. To get more information about regional settings please refer to the DEFINE REGION and SET REGION command description.

In order to map CSV file content a record descriptor is required to define the (record) layout of the CSV file(s).

The optional DESCRIBED BY clause specifies the CSV descriptor file containing the record descriptor for the CSV files to map. Depending if an entry with the name specified by the METRIX clause already exists in the CSV mapping database, this clause can be omitted or not. If you omit this clause DQL$ searches for the entry defined by the METRIX clause in the CSV mapping database and adds the CSV files defined to that entry. If no such entry exists the CSV files are not mapped. If the DESCRIBED BY clause is defined DQL$ performs the following actions:

- It checks if the CSV mapping descriptor file defined exists.
- It checks if a record descriptor as defined by the METRIX clause exists in that file. If this check fails, the user is informed and no mapping is done.

- It checks if an entry in the CSV file mapping database already exists that matches the name defined by the METRIX clause.
    - If no such file mapping entry exists, DQL$ creates a new entry and stores the record descriptor in that entry.
    - If a CSV file mapping entry already exists, DQL checks if the record descriptor in the descriptor file matches with the record descriptor stored in the CSV mapping entry. If this check fails no mapping is done.

Thus, if you are about to add new CSV files to an existing mapping definition you can omit the DESCRIBED BY clause. If you want to add a new mapping definition (entry) to the CSV mapping database the DESCRIBED BY clause is mandatory.

The FORMAT clause defines the layout of the input CSV files addressed by the MAP command. Valid keywords are SINGLE_LINE and MULTI_LINE. For more information about these layout options and the criteria CSV files have to fulfil in either of the cases please refer to the General description section. If you omit the FORMAT clause the SINGLE_LINE layout is assumed. If the CSV files addressed by the MAP command do not fulfil the criteria of the layout defined the result of the mapping will be unpredictable.

Once the command succeeds the mapped CSV files are immediately visible to the current DQL$ session and can be accessed. Any DQL$ session started afterwards on the node the mapping was defined can access the CSV data files as if these data are part of the distributed collection database without any additional user action until the CSV mapping is manually removed from the CSV mapping database.

If the CSV files mapped are not accessible although the command succeeds it is very likely that the data content is corrupt (data content is not checked by the MAP command). In that case it is recommended to run the CHECK FILE MAP command to find out what is wrong.

### Descriptor file

In order to map CSV file content a record descriptor is required to define the (record) layout of the CSV file(s). Record descriptors are defined within so called descriptor files. A descriptor file can contain 1…n record descriptors.

As explained in the general description section CSV files that are valid for CSV file mapping contains the column headers in its second line. The column headers are the linkage to the content of the record descriptor.

A record descriptor starts with the keyword “METRIX_metricname:”, and ends with the “METRIX_metricname_END:” keyword. The metric name parameter has to match the metric defined by the METRIX clause in the MAP FILE statement.

In case the layout of the CSV file(s) to map is MULTI_LINE all field descriptors (statistics) defined in the record descriptor have to exist in the mapped CSV file(s). If the layout is SINGLE_LINE the mapping is valid if at least one field descriptor (statistics) exists in the input CSV file(s).

One field descriptor is defined per line. A field descriptor has the general form:

FieldName:DataType|DataOption:DataLength:FieldDescr: Unit:

As you can see from the line above a Field descriptor consists of five mean parameters separated by a colon (":"):

- FieldName
  Max. length is 16 characters long. This is the internal name DQL uses to address the field in the records of the CSV files. Depending if the layout of the input CSV file(s) is SINGLE_LINE or MULTI_LINE different criteria are applied for searching the corresponding data column in the mapped CSV file(s).
  - SINGLE_LINE
    As described in the General description section each column of a CSV file with SINGLE_LINE layout contains data of a dedicated statistics and element. Thus, the element name and the statistics name have to be encoded in the header of the each column. The supported header formats for the columns present are:

    ElementName.StatisticsName
    ElementName.StatisticsName
    StatisticsName.ElementName
    StatisticsName.ElementName

    At least one column header of the mapped CSV file has to match the wildcard search *FieldName or FieldName*. If one of the wildcard searches succeeds the element name is extracted from the column header string by removing the FileName string. If the wildcard searches fail the same wildcard searches are performed using the FieldDesc parameter. If one of these wildcard searches succeeds the element name is extracted by removing the FieldDesc string from the column header. It is important to keep in mind that due to the fact that a column of a CSV file with SINGLE_LINE layout contains data of a dedicated statistics and element multiple columns can be addressed by one field descriptor since data of the statistics defined by the field descriptor are probably present for several elements in the CSV file. Thus, there exists no rule that the columns of the CSV file mapped have to be ordered in a special way like CSV files with MULTI_LINE layout. In addition the input CSV file can contain several columns that are not addressed by the record descriptor.

The data reference rule explained above is applied to any field descriptor present in the record descriptor but the field descriptor that refers the timestamps. As described in the General description section the column of the mapped CSV file that contains the timestamps have to match one of the wildcard searches *Time or Time* and the corresponding FieldName parameter of the field descriptor within the record descriptor that refers the timestamp column has to match the same criteria.

- MULTI_LINE
  Either this or the FieldDesc parameter has to exactly match with the column header of the CSV file at the same position. The first field descriptor in the descriptor file refers to the first column of the CSV file the second field descriptor refers to the second column and so on.

- DataType|DataOption
  - DataType
    Data type of the field. Possible keywords are:
    - FIELD$_STRING string type
    - FIELD$_INTEGER integer type
    - FIELD$_UNSIGNED unsigned integer type
    - FIELD$_QUAD quadword type
    - FIELD$_FLOAT float type
    - FIELD$_DATETIME date/time type

  - DataOption
    This parameter is optional. Keywords allowed are:
    - FIELD$_PRIMKEY
      Apply this data option keyword only if the record descriptor is used to map MULTI_LINE layout CSV files. If the layout of the CSV file is SINGLE_LINE do not use this data option keyword. It indicates that the content of the field is part of the element key DQL shall use to address the elements stored in the CSV file. This field option can be assigned to max. 3 fields. Assigning this field option to more than 3 fields the behavior of DQL will be unpredictable.If this data option keyword is not assigned to any field descriptor and the record descriptor is applied to a MULTI_LINE layout CSV file the timestamps in the mapped CSV file has to unique. In that case DQL assumes that the CSV file contains data of just one element and the metric name defined by the AS clause when mapping CSV files is assigned to that single element.
    - FIELD$_INFO
      This data option indicates that thefield content is only informational, and will not be visible to the GUI.

Use the OR (|) sign to separate the DataType and DataOption. You can enter only one DataType but both DataOptions within a field descriptor.
Example:

FIELD$_STRING|FIELD$_PRIMKEY|FIELD$_INFO

The data field is a string, part of the element key and not visible to the GUI.

- DataLength
  This parameter defines the length of the field in bytes.

  **Note**

  If the type of a field is FIELD$_DATETIME, always enter 8 (quad-word length).

- FieldDesc
  Any comment that describes the content of the column in the CSV file(s). The string length is limited to 64 characters. As described above (description of the FieldName parameter) FieldDesc can be used to refer corresponding data columns in the mapped CSV file(s).

- Unit (optional)
  Specifies the unit of the data field (e.g. 1/s, MB, sec …).

Any valid record descriptor has to contain at least one statistics (data field)

- called TIME if the record descriptor shall be applied to CSV files with MULTI_LINE layout
- that matches the wildcard search string *Time or Time* is the record descriptor shall be applied to CSV files with SINGLE_LINE layout

That statistics refers the data column of the input CVS file(s) that contains the timestamps. The maximum number of statistics (data fields) defined by a record descriptor is 200. If record descriptor exceeds the maximum number of statistics or it contains no TIME data field the MAP command fails.

## Examples

Example for a valid CSV file with SINGLE_LINE layout

```
SIFNOS,VNATIG
Time, BCSXTC.iData1, BCSXTC.iData2, VNOABS.iData1, VNOABS.iData2
13-JUL-2003 10:00, 1, 5, 0, 0
13-JUL-2003 10:04, 1, 7, 0, 0
13-JUL-2003 10:08, 0, 8, 4, 8
13-JUL-2003 10:12, 3, 1, 13, 7
13-JUL-2003 10:14, 0, 0, 8, 19
13-JUL-2003 10:16, 0, 0, 8, 8
```

The first line defines the nodes the CSV file belongs (is visible) to (node SIFNOS and VNATIG). The second line contains the column headers, followed by the data records. The timestamps in the 'Time 'column are unique as required for single line CSV files. Which fields of the record are actually mapped depends on the record descriptor.

Example for a valid CSV file with MULTI_LINE layout

```
SIFNOS,VNATIG
sKey, Time, iData1, iData2
BCSXTC, 13-JUL-2003 10:00, 1,5
BCSXTC, 13-JUL-2003 10:04, 1,7
VNOABS, 13-JUL-2003 10:08, 0,8
BCSXTC, 13-JUL-2003 10:08, 4,8
VNOABS, 13-JUL-2003 10:12, 13,7
BCSXTC, 13-JUL-2003 10:12, 3,1
VNOABS, 13-JUL-2003 10:14, 8, 19
VNOABS, 13-JUL-2003 10:16, 8, 8
```

The first line defines the nodes the CSV file belongs (is visible) to (node SIFNOS and VNATIG). The second line contains the column headers, followed by the data records. There are several rows that contain the same timestamp. This does not matter as long as the rows refer different elements. Which fields of the record are part of the element key or if no element key is available depends on the record descriptor used to map this file. If duplicates (element key and timestamp of different rows match) are detected due to the record descriptor in use the file map command fails.

Record descriptor example

The record descriptor differs depending of you want to map the CSV file with MULTI_LINE layout or the CSV file with SINGLE_LINE layout as sown above.

- SINGLE_LINE layout:

```
METRIX_USERDEFINED:

Time:   FIELD$_DATATIME:   8:   Time:            [sec]:
iData1: FIELD$_FLOAT:      4:   Desc. of iData1: [1/s]:
iData2: FIELD$_FLOAT:      4:   Desc. of iData2: [1/s]:

METRIX_USERDEFINED_END:
```

  The record descriptor in this example defines the fields:

  Time, iData1, iData2

  The content of the CSV files that is mapped to the distributed collection database using this record descriptor will be accessible via the metric USERDEFINED.

  This record descriptor is valid to map the single line formatted CSV file shown in the example before (Example for a valid CSV file with SINGLE_LINE layout) since the all field name wildcard searches applied to the CSV file as described in the Descriptor file section (*Time, *iData1, *iData2) matches CSV column headers. Even if just one field descriptor except the time field descriptor would refer a column of the example CSV file the file mapping would be valid.

  Due to the element detection algorithm used (see Descriptor file section) the CSV file example (example for a valid CSV file with SINGLE_LINE layout) contains 2 elements (= index to the appropriate data records of the CSV file) if this record descriptor is used to map the CSV file:

  - BCSXTC
  - VNOABS

- MULTI_LINE layout:

```
METRIX_USERDEFINED:

sKey:   FIELD$_STRING | FIELD$_PRIMKEY:   16:   Primary key:   N/A:
Time:   FIELD$_DATATIME:   8:   Time:            [sec]:
iData1: FIELD$_FLOAT:      4:   Desc. of iData1: [1/s]:
iData2: FIELD$_FLOAT:      4:   Desc. of iData2: [1/s]:

METRIX_USERDEFINED_END:
```

The record descriptor in this example defines the fields:

sKey, Time, iData1, iData2

The content of the CSV files that is mapped to the distributed collection database using this record descriptor will be accessible via the metric USERDEFINED.

This record descriptor is valid to map the multi-line CSV file as shown in example before (example for a valid CSV file with MULTI_LINE layout)since the field names in the record descriptor matches the CSV column headers. Due to the record descriptor definitions (primary key field definitions) no duplicates exist (no element key and timestamp of different rows match) in the CSV file.

If this record descriptor is used to map the CSV file shown in the CSV file example (example for a valid CSV file with MULTI_LINE layout) the data file consists of 2 elements (= index to the appropriate data records of the CSV file):

- BCSXTC
- VNOABS

When you look at the CSV file shown in the CSV file example (example for a valid CSV file with MULTI_LINE layout) you can see that the data rows does not have to be ordered by the elements but all rows referenced by one element have to be sorted ascending by the time field.

Command example

```
DQL> MAP FILE $2$DKA05:[PERFDAT.CSV]*.CSV;* TO COLLECTION 2MIN
cont> AS USERDEFINED
cont> REGION DEFAULT
cont> DESCRIBED BY$2$DKA105:[PERFDAT]REC_DSC.CFG;
DQL-I-MAP, CSV file(s) /$2$DKA105:[PERFDAT.CSV]*.CSV;*/ successfully mapped
```

In this example all versions of all files with the extension CSV in the directory $2$DKA05:[PERFDAT.CSV] are mapped to the collection profile 2MIN as metric USERDEFINED.The format (numbers, date, time, list separator) of the CSV files matches the regional setting DEFAULT.

The record descriptor of metric USERDEFINED is defined in the file$2$DKA105:[PERFDAT]REC_DSC.CFG.

Since the FORMAT clause is missing the layout of the input CSV files is SINGLE_LINE (for more information about the supported CSV layouts layout please refer to the General description and Descriptor File section).

If the layout of the input CSV files is MULTI_LINE the FORMAT clause is mandatory:

```
DQL> MAP FILE $2$DKA05:[PERFDAT.CSV]*.CSV;* TO COLLECTION 2MIN
cont> AS USERDEFINED
cont> REGION DEFAULT
cont> DESCRIBED BY$2$DKA105:[PERFDAT]REC_DSC.CFG
cont> FORMAT MULTI_LINE;
DQL-I-MAP, CSV file(s) /$2$DKA105:[PERFDAT.CSV]*.CSV;*/ successfully mapped
```

## *REBUILD DATABASE*

This command rescans the data files of the distributed performance database stored on all members of the PERFDAT community the local node is member of for any change and updates the virtual root file created during the start-up phase of the DQL$ utility.

### Format

REBUILD DATABASE;

### Description

This command rescans the data files of the distributed performance database stored on all members of the PERFDAT community the local node is member of for any change and updates the virtual root file created during the start-up phase of the DQL$ utility.

Thus, if any physical storage area (data file) has been relocated during the current DQL$ session this command can be used to keep the virtual root file up to date.

## *REMOVE FILE MAP*

This command removes (deletes) a particular file mapping entry defined by the file_map_name parameter from the CSV file mapping database.

### Format

REMOVE FILE MAP file_map_name;

### Description

This command removes (deletes) a particular file mapping entry defined by the file_map_name parameter from the CSV file mapping database PERFDAT$CFG:CSV_PROFILES.CSV.

For more information about CSV file mapping please refer to the MAP command description.

### Example

This example shows how to remove an existing entry named USERDEFINED from the CSV mapping database.

```
DQL> REMOVE FILE MAP USERDEFINED;
DQL-I-REMMAP, CSV mapping /USERDEFINED/ successfully removed
```

## *REMOVE PROCEDURE*

This command removes (deletes) particular user-defined statistics from the stored procedure table of the PERFDAT configuration database.

### Format

```
REMOVE PROCEDURE user_defined_statistics
        [METRIX metric_name]
                [OSTYPE OS_name]
                        [NODE node_name];
```

### Description

This command removes (deletes) particular user-defined statistics from the stored procedure table of the PERFDAT configuration database.

The user_defined_statistics parameter defines the user-defined statistics to be removed (delete). Full wildcard support is provided. Asterisk (*) and percent sign (%) wildcard characters can be placed anywhere within the user_defined_statistics string.

The clauses METRIX and OSTYPE are optional. The metric_name parameter defines the metric the user-defined statistics is member of. The OS_name parameter defines the operating system the metric defined by the metric_name parameter is valid for.

All user-defined statistics that match the user_defined_statistics parameter and the applied METRIX and OSTYPE filter will be deleted.

**Note**

If you omit the METRIX and OSTYPE filter be aware that if statistics with the same name were defined for different metrics and operating systems all these user-defined statistics will be deleted.

The NODE clause specifies the node the user-defined statistics was valid for. If you omit the NODE clause or you enter a Asterisk (*) wildcard character the user-defined statistics that matches the METRIX and OSTYPE filter criteria are deleted from the generic section of the stored procedure table of the PERFDAT configuration database. Otherwise the appropriate node specific stored procedure is deleted.

A user-defined statistics can be deleted only if it is not used by another stored procedure or a within a report profile. If it is used by another stored procedure

or a within a report profile the remove operation fails and the user gets informed about the current usage of the user-defined statistics.

Once a user-defined statistics is deleted it will inaccessible immediately for all users currently connected to the distributed PERFDAT performance database via one of the node that share the same PERFDAT configuration database as the current DQL$ session.

For more information about generic and node specific user-defined statistics please refer to the DEFINE PROCEDURE command description.

## Example

In this example all user-defined statistics $iCpu* will be deleted. Since no METRIX, OSTYPE and NODE filter is provided all user-defined statistics defined for different metrics and operating systems that matches the wildcard string are deleted from the generic section of the stored procedure table of the PERFDAT configuration database.

```
DQL> SHOW PROCEDURE *;

Generic Stored Procedures valid for all nodes of OS Type: OPENVMS

Metrix: DEVICE.CAPACITY      $iFreePerc = ifree/isize*100
                             Dscr: Device Free Percentage, Unit: [%]

Metrix: PROCESS              $iCpuNorm = iCpuLoad/iCPUs
                             Dscr: CPU load normalized, Unit: [%]

Metrix: SYSTEM               $iCpuNorm = iCpuLoad / iCpuCnt
                             Dscr: CPU load normalized, Unit: [%]
                             $iKernExec = iKernel+iExec
                             Dscr: Kernel + Exec Mode, Unit: [%]

Generic Stored Procedures valid for all nodes of OS Type: SUNOS

Metrix: SUN_SYSTEM           $iCpuNorm = iCpuLoad / iCpuCnt
                             Dscr: CPU load normalized, Unit: [%]


Node specific stored Procedures valid for OS Type: OPENVMS

Metrix:  PROCESS
         Node:   VMSTM1      $iCpuNorm= iCpuLoad / iCpus * 2
                             Dscr: CPU exec mode normalized [0..200%], Unit: [%]
         Node:   VMSTM1      $iExecNorm= iExec / iCpus
                             Dscr: CPU exec mode normalized [0..100%], Unit: [%]
         Node:   VMSTM1      $iUserNorm= iUser / iCpus
                             Dscr: CPU user mode normalized [0..100%], Unit: [%]
```

```
DQL> REMOVE PROCEDURE $iCpu*;
DQL-I-PROC, generic stored procedure /$iCpuNorm/ for metric /PROCESS/,
                                                    OS Type /OPENVMS/removed
DQL-I-PROC, generic stored procedure /$iCpuNorm/ for metric /SYSTEM/,
                                                    OS Type /OPENVMS/removed
DQL-I-PROC, generic stored procedure /$iCpuNorm/ for metric /SUN_SYSTEM/,
                                                    OS Type /SUNOS/removed

DQL> SHOW PROCEDURE *;

Generic Stored Procedures valid for all nodes of OS Type: OPENVMS

Metrix: DEVICE.CAPACITY      $iFreePerc = ifree/isize*100
                             Dscr: Device Free Percentage, Unit: [%]

Metrix: SYSTEM               $iKernExec = iKernel+iExec
                             Dscr: Kernel + Exec Mode, Unit: [%]


Node specific stored Procedures valid for OS Type: OPENVMS

Metrix: PROCESS
        Node:   VMSTM1       $iCpuNorm= iCpuLoad / iCpus * 2
                             Dscr: CPU exec mode normalized [0..200%], Unit: [%]
        Node:   VMSTM1       $iExecNorm= iExec / iCpus
                             Dscr: CPU exec mode normalized [0..100%], Unit: [%]
        Node:   VMSTM1       $iUserNorm= iUser / iCpus
                             Dscr: CPU user mode normalized [0..100%], Unit: [%]
```

## *REMOVE REGION*

This command removes (deletes) existing regional settings from the regional setting table of the PERFDAT configuration database.

### Format

REMOVE REGION reg_name;

### Description

This command removes (deletes) existing regional settings from the regional setting table of the PERFDAT configuration database.

The reg_name parameter defines the name of the regional settings to be deleted. If the name of the regional setting was stored case sensitive enter the reg_name parameter with parenthesis.

If the regional setting is the current default the command fails. In this case use the SET REGION command to redefine the current default and try again.

When installing PERFDAT the regional setting DEFAULT is automatically applied. This regional setting cannot be deleted either.

For more information about user-defined statistics please refer to the description of the DEFINE REGION and SET REGION command.

### Example

In this example the regional setting “German” will be deleted. We apply the SHOW REGION command in advance to check if the regional setting “German” exists. Since it is displayed case sensitive we enter the regional setting name with parenthesis.

```
DQL> SHOW REGION;

 Default region setting:

Name: DEFAULT
      Decimal Symbol: .
      List Seperator: ,
      Date Format   : dd-mmm-yyyy
      sMonths (ASCII): JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC


 Available region settings:

Name: DEFAULT
```

```
        Decimal Symbol: .
        List Seperator: ,
        Date Format   : dd-mmm-yyyy
        Months (ASCII): JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC

 Name: German
        Decimal Symbol: ,
        List Seperator: ;
        Date Format   : dd-mmm-yyyy
        Months (ASCII): JAN,FEB,MAR,APR,MAI,JUN,JUL,AUG,SEP,OKT,NOV,DEZ

DQL> REMOVE REGION "German";
DQL-I-CFGSUCCESS, successfully removed region setting /German/

DQL> SHOW REGION;

 Default region setting:

Name: DEFAULT
        Decimal Symbol: .
        List Seperator: ,
        Date Format   : dd-mmm-yyyy
        Months (ASCII): JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC


 Available region settings:

Name: DEFAULT
        Decimal Symbol: .
        List Seperator: ,
        Date Format   : dd-mmm-yyyy
        Months (ASCII): JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC
```

## *REMOVE VIEW*

This command removes (deletes) an existing cluster view.

### Format

REMOVE VIEW view_name;

### Description

This command removes (deletes) an existing cluster.The view_name parameter defines the name of the cluster view to be deleted. Apply the SHOW VIEW command in advance to display the cluster view available.

Once a cluster view is deleted it will inaccessible for all users currently connected to the distributed PERFDAT performance database via the local node immediately.

For more information about cluster views please refer to the description of the DEFINE VIEW command.

### Example

In this example the cluster view VIECLU consisting of the collection databases

- BCSXTC_DEFAULT
- VMSTM1_DEFAULT
- VMSTM2_DEFAULT

will be deleted.

```
DQL>SHOW VIEW;
        View                    referenced Aliases
-------------------------------------------------------------------
         RLP                    ADMIN3_DEFAULT
                                ADMIN4_DEFAULT
                                TRANS3_DEFAULT
                                TRANS4_DEFAULT
        VIECLU                  BCSXTC_DEFAULT
                                VMSTM1_DEFAULT
                                VMSTM2_DEFAULT
        VMSALL                  VMSTM2_DEFAULT
                                VMSTM1_DEFAULT

DQL> REMOVE VIEW VIECLU;
DQL-I-VIEW, view /VIECLU/ removed

DQL> SHOW VIEW;
        View                    referenced Aliases
-------------------------------------------------------------------
         RLP                    ADMIN3_DEFAULT
```

| | |
|---|---|
| | ADMIN4_DEFAULT |
| | TRANS3_DEFAULT |
| | TRANS4_DEFAULT |
| VMSALL | VMSTM2_DEFAULT |
| | VMSTM1_DEFAULT |

## *SELECT*

The SELECT command reads element data of selected statistics of a metric and displays the data on screen timely ordered.

### Format

```
SELECT [STACKED] statistics_itemlist
        [FROM] metrix_name
                ALIAS alias_namelist [DATE] date_list
                        [ELEMENT] element_list
                                [WHERE] filter_list
                                        [LIMIT] number;
```

### Description

The SELECT command reads element data of selected statistics of a metric and displays the data on screen timely ordered.

Depending if the STACKED argument is applied or not the SELECT query is stacked or un-stacked.

The stacked form of the SELECT query stacks the element data of each statistics fetched from the source collection databases before displaying them on screen. Using the un-stacked form the data are not pre-processed.

As said before the stacked form of the SELECT statement stacks the element data of each statistics selected. Thus, the stacked form of the query can be used:

- If you are interested in the stacked values of a group of elements stored in a single collection database (e.g. overall CPU load caused by a group of processes on a single node)
- If you are interested in the stacked values of a particular element stored in collection databases that refer different nodes (e.g. total I/O requests from all cluster members on a cluster wide mounted disk)
- If you are interested in the stacked values of a group of elements stored in collection databases that refer different nodes (e.g. overall I/O requests from all cluster members on all cluster wide mounted disks, cluster wide CPU load caused by a number of processes).

The un-stacked form of the statement applies to a single collection database only (data collected for a single node).

Using the stacked form of the SELECT statement a statistics called iElementCnt is automatically appended to the result table. It displays the number of elements that match the filter criteria of the SELECT (see description of the ELEMENT and

WHERE clause) statement and that are used to calculate the stacked value(s) of the selected statistics.

If the clauses of the SELECT statement are defined in a way that only one element of the metric specified is selected and the data source is a single collection database (data collected for one node) the stacked and the un-stacked form of the query returns the same result.

Prerequisite:
The data files of the collection database / logical storage areas defined by the ALIAS and DATE clause have to be attached in advance using the ATTACH command.

DQL$ evaluates the clauses of the SELECT statement in the following order:

1. FROM
2. ALIAS
3. DATE
4. ELEMENT
5. WHERE
6. LIMIT

There exist two modes to address the data fields (statistics) to fetch:

- Base address mode
  In this case the statistics to fetch from the source database and the metric the selected statistics are member of are defined separately. The statistics_itemlist specifies the names of the statistics (field names) to fetch from the metric defined by the FROM clause. Enter the statistics as a comma separated list. In this case the FROM clause is mandatory. E.g. the SELECT statement to fetch the data of the system wide CPU load (field name: iCpuLoad) and kernel mode (field name: iKernel) from metric SYSTEM of an OpenVMS collection database has the form:

  DQL$> SELECT iCpuLoad, iKernel FROM SYSTEM ALIAS…;

- Direct address mode
  In this case the statistics to fetch from the source database are entered using the full data field address. Enter the full data field addresses of all statistics to fetch (statistics_itemlist parameter) as a comma separated list. The full data field address of a statistics consists of the metric the statistics belongs to and its name (field name). The format is:

  MetricName.StatisticsName

  Since the metric the statistics belongs to is part of the full data field address the FROM most not be defined. If you enter the FROM clause the SELECT statement fails. The SELECT statement using the direct address mode to fetch the same data from a source database as shown in the example above has the form:

```
DQL$> SELECT SYSTEM.iCpuLoad, SYSTEM.iKernel ALIAS…;
```

The direct address mode is restricted to select data (statistics) form a single metric. If the statistics item list entered contains statistics that refer different metrics (E.g. SYSTEM.iCpuload, PROCESS.iCpuload) the command fails.

For both address modes it is a prerequisite that:

- All statistics specified must exist in the metric defined
- The metric (table) defined must exist in the collection databases / logical storage areas defined by the ALIAS and DATE clause.

The ALIAS clause defines the source collection databases. If you are using the un-stacked form of the SELECT statement only one collection database can be entered. The stacked form of the statement can be applied to a group of nodes. Enter the appropriate collection database aliases of the nodes of interest as a comma separated list. It does not matter if the data samples stored in the collection databases specified by the ALIAS clause were collected exactly at the same time since the data are normalized before they are stacked, but the sample interval the data were collected has to be equal. Otherwise the command fails. Use the SHOW HEADER command to verify the sample intervals of the collection databases defined by the ALIAS clause.

The database aliases can't be user-defined. DQL$ assigns these aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

```
NodeName_CollectionProfile
```

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

If you want to address specific logical storage areas (all data files that have been created on the same day) of the collection databases defined by the ALIAS clause the DATE clause is mandatory. Enter all the days of interest as a comma separated list. Use OpenVMS date format to define the days of interest. If you omit the DATE clause all attached data files (physical storage areas) that belong to the database aliases defined by the ALIAS clause are accessed.

The optional ELEMENT clause can be used to filter the elements the SELECT query applies to. Enter the element filter as a comma (,), or OR sign (|) separated list. Elements that should be excluded from the SELECT query have to be preceded with the '!=' or '<>' tag in the comma separated list of the ELEMENT clause. VSI PERFDAT V3.0 and higher versions provide full wildcard support. Asterisk (*) and percent sign (%) wildcard characters can be placed anywhere within each string of the comma separated element list.If you enter quotation marks at the beginning and the end of an element string the string is taken literally (no wildcard operation performed on that string even if it

contains wildcard characters).If you omit the ELEMENT clause all element data of the statistics selected stored in the collection database / logical storage areas defined by the ALIAS and DATE clause are read.

The optional WHERE clause can be applied to define additional filter criteria. Enter the filter criteria as a comma separated list. A single filter criterion consists of a valid statistics name of the metric defined by the FROM clause, an operator and comparison values. Valid operators are

- < less than
- <= less than or equal
- = equal
- >= greater than or equal
- > greater than
- <> not equal
- != not equal

If the operator applied is <, <=, => or > a single comparison value can be entered only. If the operator applied is =, != or <> you can enter a comparison value list. Enter the comparison value list as an OR sign (|) separated list. If the data type of the statistics defined by a single filter criterion is STRING, full wildcard support is provided. Asterisk (*) and percent sign (%) wildcard characters can be placed anywhere within the comparison strings. If you enter quotation marks at the beginning and end of a comparison string the string is taken literally (no wildcard operation performed on that string even if it contains wildcard characters).

Examples:
If you want to limit the query to the time period 25-AUG-200301:00 to 25-AUG-200304:00 enter (the TIME statistic exists for each metric):

WHERE TIME >= 25-AUG-200301:00, TIME <= 25-AUG-200304:00

If the PROCESS metric is defined by the FROM clause and you want to filter forall processes of user SYSTEM before 25-AUG-200304:00 enter:

WHERE USERNAME = SYSTEM, TIME <= 25-AUG-200304:00

The PROCESS metric is defined by the FROM clause. If you want to filter for all processes of all users that match either the wildcard filter criteria *SYS* (e.g. SYSTEM, SYSTEST …) or *TCPIP* (e.g. TCPIP$FTP, TCPIP$SNMP, TCPIP$LPD …) enter:

WHERE USERNAME = *SYS*| *TCPIP*, TIME <= 25-AUG-200304:00

With the optional LIMIT clause you can limit the output to a defined number of rows.

## Examples

Example 1

This examples demonstrates the use of the un-stacked from of the SELECT query. In this example the CPU load (iCpuLoad) and kernel mode load (iKernel)

caused by the process PERFDAT on node BCSXTC between 30-AUG-200502:00 and 30-AUG-200502:30 including the timestamps('Time' statistics) are displayed. Process data are stored in metric PROCESS.

```
DQL> SELECT Time, iCpuLoad, iKernel FROM PROCESS
cont> ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005
cont> ELEMENT PERFDAT
cont> WHERE TIME >= 30-AUG-200502:00, TIME <= 30-AUG-200502:30;
```

or

```
DQL> SELECT PROCESS.Time, PROCESS.iCpuLoad, PROCESS.iKernel
cont> ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005
cont> ELEMENT PERFDAT
cont> WHERE TIME >= 30-AUG-200502:00, TIME <= 30-AUG-200502:30;
```

| Time | iCpuLoad | iKernel |
|---|---|---|
| 30-AUG-2005 02:01:00 | 0.467 | 0.283 |
| 30-AUG-2005 02:03:00 | 1.183 | 0.875 |
| 30-AUG-2005 02:05:00 | 1.550 | 1.192 |
| 30-AUG-2005 02:07:00 | 0.658 | 0.442 |
| 30-AUG-2005 02:09:00 | 1.100 | 0.842 |
| 30-AUG-2005 02:11:00 | 1.267 | 1.000 |
| 30-AUG-2005 02:13:00 | 1.808 | 1.392 |
| 30-AUG-2005 02:15:00 | 1.400 | 1.108 |
| 30-AUG-2005 02:17:00 | 0.208 | 0.058 |
| 30-AUG-2005 02:19:00 | 0.275 | 0.142 |
| 30-AUG-2005 02:21:00 | 0.292 | 0.133 |
| 30-AUG-2005 02:23:00 | 0.308 | 0.133 |
| 30-AUG-2005 02:25:00 | 0.300 | 0.133 |
| 30-AUG-2005 02:27:00 | 0.258 | 0.117 |
| 30-AUG-2005 02:29:00 | 0.317 | 0.133 |

The ELEMENT clause is applied to filter for process PERFDAT and the WHERE clause to filter for data of the time period 30-AUG-200502:00 to 30-AUG-200502:30. In this case the SELECT query accesses the logical storage area of 30-AUG-2005 of the collection database BCSXTC_DEFAULT.

Example 2

This examples demonstrates the use of the stacked from of the SELECT query. In this example the stacked CPU load (iCpuLoad) and kernel mode load (iKernel) caused by all PERFDAT process on node BCSXTC between 30-AUG-200502:00 and 30-AUG-200502:30 including the timestamps ('Time' statistics) are displayed. Process data are stored in metric PROCESS. Although not defined the result table of the select statement includes a statistics called iElemenCnt since the stacked form of the SELECT statement is used. This statistics displays the number of elements that match the filter criteria of the SELECT (ELEMENT and WHERE clause) statement and that are used to calculate the stacked values.

```
DQL> SELECT STACKED Time, iCpuLoad, iKernel FROM PROCESS
```

```
cont> ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005
cont> ELEMENT PERFDAT*
cont> WHERE TIME >= 30-AUG-200502:00, TIME <= 30-AUG-200502:30;
```

or

```
DQL> SELECT STACKED PROCESS.Time, PROCESS.iCpuLoad, PROCESS.iKernel
cont> ALIAS BCSXTC_DEFAULT DATE 30-AUG-2005
cont> ELEMENT PERFDAT*
cont> WHERE TIME >= 30-AUG-200502:00, TIME <= 30-AUG-200502:30;
```

| Time | iCpuLoad | iKernel……iElementCnt | |
|---|---|---|---|
| 30-AUG-200502:00:00 | 15.217 | 4.117 | 5 |
| 30-AUG-2005 02:02:00 | 22.517 | 6.042 | 5 |
| 30-AUG-2005 02:04:00 | 25.909 | 7.079 | 5 |
| 30-AUG-2005 02:06:00 | 31.066 | 7.983 | 5 |
| 30-AUG-2005 02:08:00 | 37.604 | 9.229 | 5 |
| 30-AUG-2005 02:10:00 | 35.612 | 8.837 | 4 |
| 30-AUG-2005 02:12:00 | 27.812 | 7.567 | 4 |
| 30-AUG-2005 02:14:00 | 10.454 | 3.642 | 4 |
| 30-AUG-2005 02:16:00 | 0.829 | 0.608 | 4 |
| 30-AUG-2005 02:18:00 | 0.250 | 0.108 | 4 |
| 30-AUG-2005 02:20:00 | 0.283 | 0.137 | 4 |
| 30-AUG-2005 02:22:00 | 0.312 | 0.146 | 4 |
| 30-AUG-2005 02:24:00 | 0.317 | 0.146 | 4 |
| 30-AUG-200502:26:00 | 0.279 | 0.125 | 4 |
| 30-AUG-200502:28:00 | 0.308 | 0.133 | 4 |

The ELEMENT clause is applied to filter for processesPERFDAT* and the WHERE clause to filter for data of the time period 30-AUG-200502:00 to 30-AUG-200502:30. In this case the SELECT query accesses the logical storage area 30-AUG-2005 of the collection database BCSXTC_DEFAULT only.

The element count varies since all processes that match the wildcard criteria PERFDAT* (ELEMENT Clause) are selected. Since the auto-trend engine (process name PERFDAT_REPORT) completed on 30-AUG-200502:08 the element count is 5 the time before and 4 the time after.

Example 3

This is another example of the stacked from of the SELECT query. In this example the stacked cluster wide CPU load (iCpuLoad) and kernel mode load (iKernel) caused by the process PERFDAT between 30-AUG-200502:00 and 30-AUG-200502:30 including the timestamps ('Time' statistics) are displayed. The cluster is a two-node cluster using a quorum disk. The cluster members are BCSXTC and HOBEL. Process data are stored in metric PROCESS. Although not defined the result table of the select statement includes a statistics called iElemenCnt since the stacked form of the SELECT statement is used. This statistics displays the number of elements that match the filter criteria of the SELECT (ELEMENT and WHERE clause) statement and that are used to calculate the stacked values.

```
DQL> SELECT STACKED Time, iCpuLoad, iKernel FROM PROCESS
```

```
cont> ALIAS BCSXTC_DEFAULT, HOBEL_DEFAULT DATE 30-AUG-2005
cont> ELEMENT PERFDAT
cont> WHERE TIME >= 30-AUG-200502:00, TIME <= 30-AUG-200502:30;
```

| Time | iCpuLoad | iKernel……iElementCnt |
|---|---|---|
| 30-AUG-200502:00:00 | 0.658 | 0.4002 |
| 30-AUG-2005 02:02:00 | 1.142 | 0.8082 |
| 30-AUG-2005 02:04:00 | 1.846 | 1.3962 |
| 30-AUG-2005 02:06:00 | 1.575 | 1.1252 |
| 30-AUG-2005 02:08:00 | 1.246 | 0.8582 |
| 30-AUG-2005 02:10:00 | 1.729 | 1.3462 |
| 30-AUG-2005 02:12:00 | 1.958 | 1.5332 |
| 30-AUG-2005 02:14:00 | 1.658 | 1.2672 |
| 30-AUG-2005 02:16:00 | 0.854 | 0.5882 |
| 30-AUG-2005 02:18:00 | 0.308 | 0.1082 |
| 30-AUG-2005 02:20:00 | 0.342 | 0.1462 |
| 30-AUG-2005 02:22:00 | 0.337 | 0.1332 |
| 30-AUG-2005 02:24:00 | 0.350 | 0.1382 |
| 30-AUG-200502:26:00 | 0.338 | 0.1382 |
| 30-AUG-200502:28:00 | 0.346 | 0.1422 |

The ELEMENT clause is applied to filter for process PERFDAT and the WHERE clause to filter for data of the time period 30-AUG-200502:00 to 30-AUG-200502:30. Since we are interested in cluster wide stacked values the collection databases of both cluster members are entered as a comma separated list in the ALIAS clause (BCSXTC_DEFAULT, VNOABS_DEFAULT). The DATE clause is present. Thus, the SELECT query accesses the logical storage areas 30-AUG-2005 of both collection databases (BCSXTC_DEFAULT, HOBEL_DEFAULT) only.

The element count is 2 for the whole period of time since PERFDAT was running on both nodes (BCSXTC and VMSTM1).

You get the same result table if you define a cluster view that consists of the member BCSXTC_DEFAULT and VNOABS_DEFAULT and you select the data from the virtual collection database defined by the cluster view (form more information about cluster views please refer to the DEFINE VIEW command description).

```
DQL> DEFINE VIEW VIECLUALIAS BCSXTC_DEFAULT, VNOABS_DEFAULT;
DQL-I-VIEW, view /VIECLU/ defined

DQL> SELECT STACKED Time, iCpuLoad, iKernel FROM PROCESS
cont> ALIAS VIECLU_VIEW DATE 30-AUG-2005
cont> ELEMENT PERFDAT
cont> WHERE TIME >= 30-AUG-200502:00, TIME <= 30-AUG-200502:30;
```

Example 4

In this example the stacked cluster wide CPU load (iCpuLoad) and kernel mode load (iKernel) caused by all PERFDAT and DQL processes between 30-AUG-200502:00 and 31-AUG-200502:00including the timestamps (‘Time’ statistics)

are displayed. As in example 3 the cluster consists of node BCSXTC and HOBEL. Process data are stored in metric PROCESS. Although not defined the result table of the select statement includes a statistics called iElemenCnt since the stacked form of the SELECT statement is used. This statistics displays the number of elements that match the filter criteria of the SELECT (ELEMENT and WHERE clause) statement and that are used to calculate the stacked values.

```
DQL> SELECT STACKED Time, iCpuLoad, iKernel FROM PROCESS
cont> ALIAS BCSXTC_DEFAULT, HOBEL_DEFAULT DATE 30-AUG-2005, 31-AUG-2005
cont> ELEMENT PERFDAT*, DQL*
cont> WHERE TIME >= 30-AUG-200502:00, TIME <= 31-AUG-200502:00;
```

| Time | iCpuLoad | iKernel | iElementCnt |
|---|---|---|---|
| 30-AUG-200502:00:00 | 49.908 | 14.533 | 12 |
| 30-AUG-2005 02:02:00 | 87.092 | 30.529 | 12 |
| 30-AUG-2005 02:04:00 | 125.984 | 49.079 | 12 |
| 30-AUG-2005 02:06:00 | 123.025 | 44.017 | 12 |
| 30-AUG-2005 02:08:00 | 118.833 | 39.387 | 10 |
| 30-AUG-2005 02:10:00 | 109.679 | 39.167 | 8 |
| 30-AUG-2005 02:12:00 | 100.616 | 40.154 | 8 |
| 30-AUG-2005 02:14:00 | 51.334 | 22.750 | 8 |
| 30-AUG-2005 02:16:00 | 0.879 | 0.613 | 8 |
| 30-AUG-2005 02:18:00 | 0.317 | 0.117 | 8 |
| . | | | |
| . | | | |
| . | | | |
| 31-AUG-2005 01:50:00 | 0.425 | 0.204 | 11 |
| 31-AUG-2005 01:52:00 | 0.379 | 0.150 | 11 |
| 31-AUG-2005 01:54:00 | 0.304 | 0.096 | 11 |
| 31-AUG-200501:56:00 | 0.342 | 0.137 | 11 |
| 31-AUG-200501:58:00 | 0.413 | 0.154 | 11 |

The ELEMENT clause is applied to filter for processes PERFDAT* and DQL* and the WHERE clause to filter for data of the time period 30-AUG-200502:00 to 31-AUG-200502:00. Since we are interested in cluster wide stacked values the collection databases of both cluster members are entered as a comma separated list in the ALIAS clause (BCSXTC_DEFAULT, VNOABS_DEFAULT). The data of interest is stored in different logical storage areas of the collection databases. Thus, the DATE clause contains the dates 30-AUG-2005 and 31-AUG-2005.

You get the same result table if you define a cluster view that consists of the member BCSXTC_DEFAULT and HOBEL_DEFAULT and you select the data from the virtual collection database defined by the cluster view (form more information about cluster views please refer to the DEFINE VIEW command description).

```
DQL> DEFINE VIEW TESTCLU ALIAS BCSXTC_DEFAULT, HOBEL_DEFAULT;
DQL-I-VIEW, view /TESTCLU/ defined

DQL> SELECT STACKED Time, iCpuLoad, iKernel FROM PROCESS
cont> ALIAS TESTCLU_VIEW DATE 30-AUG-2005, 31-AUG-2005
cont> ELEMENT PERFDAT*, DQL*
```

```
cont> WHERE TIME >= 30-AUG-200502:00, TIME <= 31-AUG-200502:00;
```

## *SET INFORMATIONAL*

This command controls whether or not the DQL$ utility displays informational messages at the terminal or if informational messages are printed in a batch job log file.

### Format

SET INFORMATIONAL;  default

SET NOINFORMATIONAL;

### Description

This command controls whether or not the DQL$ utility displays informational messages at the terminal or if informational messages are printed in a batch job log file.

## *SET REGION*

This command is used to change the default regional setting for the current DQL$ session and all subsequent DQL$ sessions started on any node that share the same PERFDAT configuration database.

### Format

SET REGION reg_name;

### Description

This command is used to change the default regional setting for the current DQL$ session and all subsequent DQL$ sessions started on any node that share the same PERFDAT configuration database.

The reg_name parameter defines the name of the regional setting to be used as the new default. Prerequisite is that the regional setting defined by this parameter exists in the regional setting table of the PERFDAT configuration database. Use the SHOW REGION command to display the regional settings available. Enter the parameter with parenthesis if the name of the regional setting is displayed case sensitive.

The default regional setting defines how the DQL$ utility formats numbers, date, time and the list separator when exporting performance data to CSV files.

**Note**

Once the default regional setting has been changed for the current DQL$ session these settings will be used for any subsequent data exports, CSV data import and load operation until the default is redefined using this command. Even if you close the session and start a new DQL$ session the default regional setting does not change. Thus, if you want to change the format for data exports to CSV files or CSV data import or load operations change the default regional settings in advance.

### Example

In this example the default regional setting for the current DQL$ session shall be changed to “German”. We apply the SHOW REGION command in advance to check if the regional setting “German” exists. Since it is displayed case sensitive it is entered with parenthesis.

```
DQL> SHOW REGION;

 Default region setting:
```

```
 Name: DEFAULT
       Decimal Symbol: .
       List Seperator: ,
       Date Format   : dd-mmm-yyyy
       sMonths (ASCII): JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC


 Available region settings:

Name: DEFAULT
       Decimal Symbol: .
       List Seperator: ,
       Date Format   : dd-mmm-yyyy
       Months (ASCII): JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC

 Name: German
       Decimal Symbol: ,
       List Seperator: ;
       Date Format   : dd-mmm-yyyy
       Months (ASCII): JAN,FEB,MAR,APR,MAI,JUN,JUL,AUG,SEP,OKT,NOV,DEZ

DQL> SET REGION "German";
DQL-I-CFGSUCCESS, default region setting changed to /German/

DQL> SHOW REGION;

 Default region setting:

 Name: German
       Decimal Symbol: ,
       List Seperator: ;
       Date Format   : dd-mmm-yyyy
       sMonths (ASCII): JAN,FEB,MAR,APR,MAI,JUN,JUL,AUG,SEP,OKT,NOV,DEZ


 Available region settings:

Name: DEFAULT
       Decimal Symbol: .
       List Seperator: ,
       Date Format   : dd-mmm-yyyy
       Months (ASCII): JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC

 Name: German
       Decimal Symbol: ,
       List Seperator: ;
       Date Format   : dd-mmm-yyyy
       Months (ASCII): JAN,FEB,MAR,APR,MAI,JUN,JUL,AUG,SEP,OKT,NOV,DEZ
```

## *SET TRANSACTION ALIAS*

This command defines the access mode to a specific collection database or logical storage area.

### Format

```
SET TRANSACTION { READ ONLY | READ WRITE }
                                    ON ALIAS alias_name [DATE date];
```

### Description

This command defines the access mode to a specific collection database or logical storage area. The access mode can be changed dynamically.

- READ ONLY read access only
- READ WRITE read/write (select/insert) access

The default value is READ ONLY.

The collection database/logical storage area is defined by the ALIAS and DATE clause. The ALIAS clause defines the collection database alias, the DATE clause the day of interest.

The database alias can't be user-defined. DQL$ assigns the aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

If you omit the DATE clause the access mode is set for the whole collection database defined by the ALIAS clause. If the DATE clause is present only the logical storage area that is referred by the ALIAS and DATE clause is affected.

## Example

This example shows how to set read write access to a logical storage area. The logical storage area 30-AUG-2005 of the collection database HOBEL_DEFAULT contains a single physical storage area.

```
DQL> SET TRANSACTION READ WRITE ON
cont> ALIAS HOBEL_DEFAULT DATE 30-AUG-2005;
DQL-I-SET, file /HOBEL_DEFAULT_2005-08-30:00:03:00:1/ will be READ WRITE
accessible
```

## *SET TRANSACTION FILE*

This command defines the access mode to a specific physical storage area.

### Format

```
SET TRANSACTION { READ ONLY | READ WRITE }
                                        ON 'FILE filename_alias';
```

### Description

This command defines the access mode to a specific physical storage area specified by the 'filename_alias' parameter. The access mode can be changed dynamically.

- READ ONLY read access only
- READ WRITE read/write (select/insert) access

The default value is READ ONLY.

### Example

This example shows how to set read only access to a physical storage area.

```
DQL> SET TRANSACTION READ ONLY ON
cont> 'FILE HOBEL_DEFAULT_2005-08-30:00:03:00:1';
DQL-I-SET, file /HOBEL_DEFAULT_2005-08-30:00:03:00:1/ will be READ ONLY
accessible
```

## *SET VERIFY*

This command controls whether command lines in a valid DQL$ script are displayed at the terminal or are printed in a batch job log file.

### Format

SET VERIFY;

SET NOVERIFY;  default

### Description

This command controls whether command lines in a valid DQL$ script are displayed at the terminal or are printed in a batch job log file.

For information about DQL$ scripts please refer to the @ command description.

## *SHOWDATABASE*

This command lists all collection databases accessible to the DQL$ session.

### Format

SHOW DATABASE;

### Description

This command lists all collection databases that can be accessed by the current DQL$ session. Depending of the community definition (logical PERFDAT$COMMUNTY) all or some databases of the whole distributed performance database are accessible.

For more information about the distributed performance database and database organization please see the chapters VSI PERFDAT distributed performance database and VSI PERFDAT Query Interface (DQL)or the manual VSI PERFDAT - Architecture and Technical Description.

### Example

This example demonstrates the use of the SHOW DATABASE command on node VNOABS. The node VNOABS and VMSTM1 are assigned to the logical PERFDAT$COMMUNITY.

```
DQL> SHOW DATABASE;
```

| Nodes | Type | Collection[Profile] | File Name (Alias) | Start Time | Alias | O |
|---|---|---|---|---|---|---|
| VNOABS | D | 2MIN | | | VNOABS_2MIN | N |
| | R | WEEK | | | VNOABS_WEEK | N |
| VMSTM1 | D | 2MIN | | | VNOABS_2MIN | N |
| | R | WEEK | | | VNOABS_WEEK | N |
| VNOCLU | DV | VIEW | | | VNOCLU_VIEW | N |
| REPORT | RV | VIEW | | | REPORT_VIEW | N |

The SHOW DATABASE command provides the information which collection databases can be accessed by (visible to) the DQL$ session due to the community definition. In this case the collection databases that refer to the nodes VNOABS and VMSTM1 are visible. In addition two cluster views – VNOCLU and REPORT - are defined on VNOABS. Since all members of the cluster views are accessible by the current DQL$ session these cluster views are listed too (for more information about cluster views please refer to the chapter VSI PERFDAT Query Interface (DQL)and the command description of the DEFINE VIEW command).

The database view displays only the Nodes, (collection database) Type, Collection [Profile] and (database) Alias columns. If the collection databases or parts of it (logical or physical storage areas) are already attached, is shown in the “O” (open) column.

Four different collection database types exist. The collection database type defines the type of data stored in the collection database and the source that created the data:

- D Raw Data - performance data collected by the OpenVMS data collector or the SNMP extension
- R Report – data were created by the auto-trend engine by applying the EXTRACT command.
- DV Cluster view – members of the cluster view are collection databases created by the OpenVMS data collector or the SNMP extension
- RV Report cluster view – members of the cluster view are report collection database.

## *SHOW ELEMENT*

This command lists all elements (indices) of a particular metric (table).

### Format

```
SHOW ELEMENT element_itemlist
        FROM metrix_name
                ALIAS alias_name [DATE] date
                        [ORDERED BY] statistics_name
                                [ASCENDING/DESCENDING]
                                        [WHERE] filterlist
                                                [INTO] file_name;
```

### Description

The SHOW ELEMENT command lists all elements (indices) of a particular metric (table).

The element_itemlist specifies the elements to be searched for in the metric defined by the FROM clause. Enter the element item list as a comma (,), or OR sign (|) separated list. Elements that should be excluded from the query have to be preceded with the '!=' or '<>' tag in the comma separated list. VSI PERFDAT V3.0 and higher versions provide full wildcard support. Asterisk (*) and percent sign (%) wildcard characters can be placed anywhere within each string of the comma separated element list. If you enter quotation marks at the beginning and the end of an element item the string is taken literally (no wildcard operation performed on that string even if it contains wildcard characters). If you omit the element_itemlist parameter the SHOW ELEMENT lists all elements available.

The FROM clause specifies the metric to search for elements.

Prerequisite:
The data files of the collection database / logical storage areas defined by the ALIAS and DATE clause have to be attached in advance using the ATTACH command.

The ALIAS clause defines the collection databases the SHOW ELEMENT command applies to. You can define 1…n collection database aliases as a comma separated list to search for elements of the metric defined by the FROM clause.

The database aliases can't be user-defined. DQL$ assigns these aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

If the SHOW ELEMENT query shall be applied to specific logical storage areas (all data files that have been created on the same day) of the collection databases defined by the ALIAS clause the DATE clause is mandatory. Enter all the days of interest as a comma separated list. Use OpenVMS date format to define the days of interest. If you omit the DATE clause all attached data files (physical storage areas) that belong to the database aliases defined by the ALIAS clause are accessed.

The optional ORDERED BY clause can be applied to sort the output. The statistics (field) defined by the ORDERED BY clause is the sort criterion. The stacked integral mean value of that statistics is calculated for each element and the elements are displayed in ascending order of these values if the ASCENDING keyword is applied or in descending order if the DESCENDING keyword is applied.

In order to obtain more information about calculating stacked values please see the CALCULATE (base form) command description.

The DESCENDING or ASCENDING keyword is mandatory if you use the ORDERED BY keyword.

The optional WHERE clause can be applied to define additional filter criteria. Enter the filter criteria as a comma separated list. A single filter criterion consists of a valid statistics name of the metric defined by the FROM clause, an operator and comparison values. Valid operators are

- < less than
- <= less than or equal
- = equal
- >= greater than or equal
- > greater than
- <> not equal
- != not equal

If the operator applied is <, <=, => or > a single comparison value can be entered only. If the operator applied is =, != or <> you can enter a comparison value list. Enter the comparison value list as an OR sign (|) separated list. If the data type of the statistics defined by a single filter criterion is STRING, full wildcard support is provided. Asterisk (*) and percent sign (%) wildcard characters can be placed anywhere within the comparison strings. If you enter quotation marks at the beginning and end of a comparison string the string is taken literally (no wildcard operation performed on that string even if it contains wildcard characters).

Examples:
If you want to limit the query to the time period 25-AUG-200301:00 to 25-AUG-200304:00 enter (the TIME statistic exists for each metric):

WHERE TIME >= 25-AUG-200301:00, TIME <= 25-AUG-200304:00

If the PROCESS metric is defined by the FROM clause and you want to filter for all processes of user SYSTEM before 25-AUG-200304:00 enter:

WHERE USERNAME = SYSTEM, TIME <= 25-AUG-200304:00

The PROCESS metric is defined by the FROM clause. If you want to filter for all processes of all users that match either the wildcard filter criteria *SYS* (e.g. SYSTEM, SYSTEST …) or *TCPIP* (e.g. TCPIP$FTP, TCPIP$SNMP, TCPIP$LPD …) enter:

WHERE USERNAME = *SYS*| *TCPIP*, TIME <= 25-AUG-200304:00

You can redirect the output of the query to a user definable CSV file if you apply the optional INTO clause. If you omit the INTO clause the result of the query is only displayed on screen.

## Examples

Example 1

The collection database COS03_10SEC is attached. This collection database consists of a single logical storage1-Dec-2003which in turn consists of two physical storage areas created on 19:02:00 and 17:35:00.

To display all elements of the DEVICE metric stored in the whole collection database enter:

```
DQL> SHOW ELEMENT FROM DEVICE ALIAS COS03_10SEC;

ELEMENT LIST of storage area DEVICE

Element                    refers to          Ratio
-------------------------------------------------------------------
$1$DGA101                                     0.0
$1$DGA201                                     0.0
DSA0                                          0.0
DSA1                                          0.0
DSA10                                         0.0
DSA11                                         0.0
DSA12                                         0.0
DSA13                                         0.0
DSA8                                          0.0
DSA9                                          0.0

    Elements: 10
```

Example 2

In order to display the elements of the DEVICE metric ordered by their QIO load between 1-DEC-2003 17:40 and1-DEC-2003 18:10. (The device QIO load is stored in the field iQios) enter:

```
DQL> SHOW ELEMENT FROM ALIAS COS03_10SEC DEVICE
cont> ORDERED BY IQIOS DESCENDING
cont> WHERE TIME > 1-DEC-2003 17:40, TIME < 1-DEC-2003 18:10;

ELEMENT LIST of storage area DEVICE

Element                    refers to          Ratio
-------------------------------------------------------------------
DSA1                                          98.0
DSA0                                          1.2
DSA13                                         0.3
DSA12                                         0.2
DSA8                                          0.2
DSA10                                         0.0
DSA9                                          0.0
DSA11                                         0.0

    Elements: 8
```

You can see that only 8 of 10 elements meet the criterions specified with the WHERE clause (no QIOs done on device $1$DGA101 and$1$DGA201 between 1-DEC-2003 17:40 and 1-DEC-2003 18:10).

During the specified time period (WHERE clause) 98% of all QIO's where done on DSA1.

Example 3

The collection databases HOBEL_DEFAULT and BCSXTC_DEFAULT are attached.

In order to display all PERFDAT processes stored in the PROCESS metric in both collection databases attached ordered by the CPU load (iCpuLoad statistics) enter:

```
DQL> SHOW ELEMENT PERFDAT* FROM PROCESS
cont> ALIAS HOBEL_DEFAULT, BCSXTC_DEFAULT DATE 30-AUG-2005
cont> ORDERED BY iCpuLoad DESCENDING;

ELEMENT LIST

Element                          refers to          Ratio
-----------------------------------------------------------------------
PERFDAT                                             54.3
PERFDAT_REPORT                                      43.5
PERFDAT_SNMP_0                                      2.1
PERFDAT_ARCHIVE                                     0.1
PERFDAT_SNMP                                        0.0

    Elements: 5
```

Five processes are found in the logical storage areas of both collection databases. The element listing does not indicate that each of the elements are found in both logical storage areas, but that a particular element was found in at least one of it.

Be careful interpreting the result table. It does not indicate that the process PERFDAT was the top CPU consumer at all. The result table shows that on 30-AUG-2005 the top CPU consumer of all PERFDAT* processes active on both nodes (BCSXTC, HOBEL) was the process PERFDAT. It caused 54.3 % of the overall (stacked) CPU load caused by all PERFDAT* processesactive on the nodes HOBEL and BCSXTC.

## *SHOW FILE MAP*

This command displays the entries of the CSV file mapping database available on the local node.

### Format

SHOW FILE MAP [file_map_name];

### Description

This command displays the entries of the CSV file mapping database available on the local node.

If the file_map_name parameter is omitted brief information about the existing entries stored in the CSV file mapping database are displayed. In order to get detailed information about a particular file map entry you have to enter the file map name.

For more information about CSV file mapping please see the MAP FILE command.

### Examples

Example 1

This example shows how to get brief information about existing mapping entries stored in CSV mapping database.

```
DQL> SHO FILE MAP;

    CSV mapping: SPHINX
```

Example 2

This example shows how to full brief information about an existing mapping entry (SPHINX) stored in CSV mapping database.

```
DQL> SHO FILE MAP SPHINX;

  CSV mapping: SPHINX
      Member of Profile:     DEFAULT
      Metrix Name in use:    SPHINX
      Metrix Descriptor file: PERFDAT$DB_ARCHIVE:SPHINX_DSC.CFG
      Mapped CSV files:
          1. Entry:      PERFDAT$DB_ARCHIVE:*.CSV
```

## *SHOW HEADER*

This command reads the header of the attached physical storage areas and displays basic information about the data collection that created the physical storage area.

### Format

SHOW HEADER;

### Description

This command reads the header of the attached physical storage areas and displays basic information about the data collection that created the physical storage area. This includes start and stop time of the collection, the collection profile the data collection was started with, the number of metrics and the number of time the data collector was triggered to collect data.

Depending on the creator of the physical storage area (OpenVMS data collector, SNMP extension, auto-trend engine, mapped CSV file) the output differs.

Example

Attach logical storage area:

DQL> ATTACH ALIAS HOBEL_DEFAULT DATE 30-AUG-2005;

Show header of the attached logical storage area:

DQL> SHOW HEADER;

```
HEADER of storage area HOBEL_DEFAULT_2005-08-30:00:03:00:1
```

---

```
OpenVMS Header field definitions

                                        Description     Type
                                        -----------     ----

Version of PerfDat that created this file     V3.0

                                          Node Name     HOBEL
OpenVMS Node originally created this storage area     HOBEL
                                          Node Type     Server
                                   Operating system     OpenVMS
                              OS version of the node     V7.3-1
                                        Profile Name     DEFAULT
                                          Start Time     30-AUG-200500:03:00
                                           Stop Time     31-AUG-200500:01:00
                          Actual Sample Inter. [s]     120
```

```
                  Original set Sample Inter. [s]     120
                       System metric enabled     TRUE
                          Cpu metric enabled     TRUE
 Process metric enabled     TRUE
                             Selected Processes     ALL
                           User metric enabled     TRUE
                                  Selected User     ALL
                          Image metric enabled     TRUE
                                    Selected Images     ALL
                        Account metric enabled     TRUE
                            Selected Account Names     All
                     XFC Volume metric enabled     TRUE
              XFC IO Size metric on Volume enabled     FALSE
                       XFC File metric enabled     FALSE
                XFC IO Size metric on File enabled     TRUE
                             XFC Selected Volumes     ALL
                         Device metric enabled     TRUE
                                   Selected Devices     *$D*, *DSA*
       IO Size Metric on selected Devices enabled     FALSE
          File Metric on selected Devices enabled     FALSE
Process Metric on selected Devices enabled     FALSE
                        Selected Processes/Devices     ALL
  File Metric/Process on selected Devices enabled     FALSE
   Device Capacity & Path Info metric enabled     TRUE
     LAN Adapter metric enabled     TRUE
       LAN Device/Adapter metric enabled     TRUE
LAN Protocol metric enabled     TRUE
                       SCS metric enabled     FALSE
                          Collection Creation Time     30-AUG-2005 00:01:00
                             Collection Flush Time     31-AUG-200500:00:00
                                      # of Samples     719
                     # of Metrics     14
```

## *SHOW LOGICAL STORAGE AREA*

This command displays the logical storage area view of the collection databases accessible to the DQL$ session.

### Format

SHOW LOGICAL STORAGE AREA [ALIAS alias_name];

### Description

This command displays the logical storage area view of the collection databases accessible to the DQL$ session. The ALIAS clause is optional. It defines the collection database alias the SHOW command applies to.

The database alias can't be user-defined. DQL$ assigns the aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

If the ALIAS clause is omitted, the logical storage areas of all collection databases accessible (visible) to DQL$ session are displayed.

For more information about the distributed collection database and database organization please see the chapters VSI PERFDAT distributed performance database and VSI PERFDAT Query Interface (DQL)or the manual VSI PERFDAT - Architecture and Technical Description.

### Example

Example 1

DQL> SHOW LOGICAL STORAGE AREA;

| Nodes | Type | Collection[Profile] | File Name (Alias) | Start Time | Alias | O |
|---|---|---|---|---|---|---|
| VNOABS | D | 10S | | 24-SEP-2003 14:29:00 | VNOABS_10S | N |
| | | 20S | | 24-SEP-2003 14:29:00 | VNOABS_20S | N |
| | | 2MIN | | 19-SEP-2003 00:03:00 | VNOABS_2MIN | Y |
| | | | | 24-SEP-2003 00:03:00 | VNOABS_2MIN | Y |
| | | | | 25-SEP-2003 00:03:00 | VNOABS_2MIN | Y |

The logical storage area is defined to be the sum of all physical storage areas created on the same day. Thus, the logical storage area view displays these days

performance data are collected and stored in the appropriate collection databases that are accessible (visible) to the DQL$ session. The column 'Start Time' displays the time of the first data sample stored in each of the logical storage areas.

Example 2

```
DQL> SHOW LOGICAL STORAGE AREA ALIAS VNOABS_2MIN;
```

| Nodes | Type | Collection[Profile] | File Name (Alias) | Start Time | Alias | O |
|---|---|---|---|---|---|---|
| VNOABS | D | 2MIN | | 19-SEP-2003 00:03:00 | VNOABS_2MIN | Y |
| | | | | 24-SEP-2003 00:03:00 | VNOABS_2MIN | Y |
| | | | | 25-SEP-2003 00:03:00 | VNOABS_2MIN | Y |

In this example only the logical storage areas of collection databaseVNOABS_2MIN are displayed since the ALIAS clause is present.

## *SHOW METRIX*

This command displays all the metrics (tables) stored in attached physical storage areas and the number of elements that exists within each metric.

### Format

SHOW METRIX [ALIAS] alias_name [DATE] date;

### Description

The SHOW METRIX command displays all the metrics (tables) stored in attached physical storage areas and the number of elements that exists within each metric.

The ALIAS and DATE clause are optional. They define the collection database alias and the day of interest (logical storage area) the SHOW command applies to.

The database alias can't be user-defined. DQL$ assigns the aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

If the DATE clause is omitted the command applies to all physical storage areas of the collection database defined by the ALIAS clause. If the ALIAS clause is omitted too, the command applies to all attached physical storage areas.

### Examples

The collection database VNOABS_2MIN has been attached. It consists of the logical storage area 18-Sep-2003 and 19-Sep-2003. The logical storage area 18-Sep-2003 consists of one physical storage area (VNOABS_2MIN_2003-SEP-18:20:43:00) and the logical storage area 19-Sep-2003 consists of 2 physical storage areas (VNOABS_2MIN_2003-SEP-19:00:03:00, VNOABS_2MIN_2003-SEP-19:15:03:00).

Example 1

In this example the metrics stored in all physical storage areas previously attached are displayed since no collection database / logical storage area filter have been defined.

```
DQL> SHOW METRIX;

METRIX DEFINITION of storage area VNOABS_2MIN_2003-SEP-18:20:43:00

Metrics enabled                Element Count
-------------------------------------------------------
CPU                            4
DEVICE                         2
IMAGE                          35
PROCESS                        38
SCSPORT                        1
SCSPORT.VC                     1
SCSPORT.VC.CHANNEL             3
SYSTEM                         1
USER                           8
XFCVOLUME                      2

METRIX DEFINITION of storage area VNOABS_2MIN_2003-SEP-19:00:03:00

Metrics enabled                  Element Count
-------------------------------------------------------
CPU                              4
DEVICE                           2
IMAGE                            45
PROCESS                          42
SCSPORT                          1
SCSPORT.VC                       1
SCSPORT.VC.CHANNEL               3
SYSTEM                           1
USER                             8
XFCVOLUME                        2

METRIX DEFINITION of storage area VNOABS_2MIN_2003-SEP-19:15:03:00

Metrics enabled          Element Count
-----------------------------------------------
CPU                      4
DEVICE                   2
IMAGE                    45
PROCESS                  42
SCSPORT                  1
SCSPORT.VC               1
SCSPORT.VC.CHANNEL       3
SYSTEM                   1
USER                     8
XFCVOLUME                2
```

Example 2

In this example the metrics stored in the physical storage areas that have been created on 18-SEP-2003 are displayed only, since the command is entered with the appropriate logical storage area filter (ALIAS and DATE clause are present).

```
DQL> SHOW METRIX ALIAS VNOABS_2MIN DATE 18-SEP-2003;

METRIX DEFINITION of storage area VNOABS_2MIN_2003-SEP-18:20:43:00
```

| Metrics enabled | Element Count |
|---|---|
| CPU | 4 |
| DEVICE | 2 |
| IMAGE | 35 |
| PROCESS | 38 |
| SCSPORT | 1 |
| SCSPORT.VC | 1 |
| SCSPORT.VC.CHANNEL | 3 |
| SYSTEM | 1 |
| USER | 8 |
| XFCVOLUME | 2 |

## *SHOW PHYSICAL STORAGE AREA*

This command displays the physical storage area view of the collection databases accessible to the DQL$ session.

### Format

SHOW PHYSICAL STORAGE AREA [ALIAS alias_name];

### Description

This command displays the physical storage area view of the collection databases accessible to the DQL$ session. The ALIAS clause is optional. It defines the collection database alias the SHOW command applies to.

The database alias can't be user-defined. DQL$ assigns the aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

If the ALIAS clause is omitted, the physical storage areas of all collection databases accessible (visible) to DQL$ session are displayed.

For more information about the distributed performance database and database organization please see the chapters VSI PERFDAT distributed performance database and VSI PERFDAT Query Interface (DQL)or the manual VSI PERFDAT - Architecture and Technical Description.

### Examples

Example 1

DQL> SHOW PHYSICAL STORAGE AREA;

| Nodes | Type | Collection[Profile] | File Name (Alias) | Start Time | Alias | O |
|---|---|---|---|---|---|---|
| VNOABS | D | 10S | VNOABS_10S_2003-SEP-24:14:29:00:1 | 24-SEP-200314:29:00 | VNOABS_10S | N |
| | | 20S | VNOABS_20S_2003-SEP-24:14:29:00:1 | 24-SEP-2003 14:29:00 | VNOABS_20S | N |
| | | | VNOABS_2MIN_2003-SEP-19:00:03:00:1 | 19-SEP-2003 00:03:00 | VNOABS_2MIN | Y |
| | | 2MIN | VNOABS_2MIN_2003-SEP-19:15:03:00:1 | 19-SEP-2003 15:03:00 | VNOABS_2MIN | Y |
| | | | VNOABS_2MIN_2003-SEP-24:00:03:00:1 | 24-SEP-2003 00:03:00 | VNOABS_2MIN | Y |
| | | | VNOABS_2MIN_2003-SEP-25:00:03:00:1 | 25-SEP-2003 00:03:00 | VNOABS_2MIN | Y |
| BCSXTC | D | 10S | BCSXTC_10S_2003-SEP-24:14:29:00:1 | 24-SEP-200300:02:00 | BCSXTC_10S | N |

In this example the collection databases of node VNOABS and BCSXTC are visible to DQL$ session due to the community definition. The first column displays all community members (VNOABS, BCSXTC). The second column displays all available collection databases that refer to the node listed in the column before. The third column displays the physical storage areas and their filename aliases (physical storage area aliases) automatically created by DQL$. The fourth column displays the time the first data sample was stored in each of the physical storage areas. The fifth column display the collection database alias the physical storage area belongs to. The last column shows if a specific physical storage area is already attached.

When you look at the output you can see that the logical storage area of 19-Sep-2003 and 20-Sep-2003 consists of 2 physical storage areas. All other logical storage areas consist of a single physical storage area.

Example 2

```
DQL> SHOW PHYSICAL STORAGE AREA ALIAS VNOABS_2MIN;
```

| Nodes | Type | Collection[Profile] | File Name (Alias) | Start Time | Alias | O |
|---|---|---|---|---|---|---|
| VNOABS | D | 2MIN | VNOABS_2MIN_2003-SEP-19:15:03:00:1 | 19-SEP-2003 15:03:00 | VNOABS_2MIN | Y |
| | | | VNOABS_2MIN_2003-SEP-24:00:03:00:1 | 24-SEP-2003 00:03:00 | VNOABS_2MIN | Y |
| | | | VNOABS_2MIN_2003-SEP-25:00:03:00:1 | 25-SEP-2003 00:03:00 | VNOABS_2MIN | Y |

In this example only the physical storage areas of collection databaseVNOABS_2MIN are displayed since the ALIAS clause is present.

## *SHOW PROCEDURE*

The SHOW PROCEDURE command displays the user-defined statistics stored in the stored procedure table of the PERFDAT configuration database.

### Format

```
SHOW PROCEDURE user_defined_statistics
        [METRIX metric_name]
                [OSTYPE OS_name]
                        [NODE node_name];
```

### Description

The SHOW PROCEDURE command displays the user-defined statistics stored in the stored procedure table of the PERFDAT configuration database.

The user_defined_statistics parameter defines the user-defined statistics to display. Full wildcard support is provided. Asterisk (*) and percent sign (%) wildcard characters can be placed anywhere within the user_defined_statistics string.

The clauses METRIX and OSTYPE are optional. The metric_name parameter defines the metric the statistics defined by the user_defined_statistics parameter is member of. The OS_name parameter defines the operating system the metric defined by the metric_name parameter is valid for.

If you omit the optional clause NODE all statistics of the generic as well as of the node specific section that matches the filter criteria defined by the user_defined_statistics parameter and the METRIX and OSTYPE clause are displayed. If you apply the NODE clause only the user-defined statistics of the node specific section of the PERFDAT configuration database that match the all filter criteria are displayed.

For more information about generic and node specific user-defined statistics please refer to the DEFINE PROCEDURE command description.

## Examples

Example 1:

In this example all user-defined statistics $iCpu* will be displayed. Since no METRIX and OSTYPE filter is provided all user-defined statistics defined for different metrics and operating systems that matches the wildcard string are displayed.

```
DQL> SHOW PROCEDURE $iCpu*;

Generic Stored Procedures valid for all nodes of OS Type: OPENVMS

Metrix:  PROCESS          $iCpuNorm = iCpuLoad/iCPUs
                          Dscr: CPU load normalized, Unit: [%]
Metrix:  SYSTEM           $iCpuNorm = iCpuLoad / iCpuCnt
                          Dscr: CPU load normalized, Unit: [%]
                          $iKernExec = iKernel+iExec
                          Dscr: Kernel + Exec Mode, Unit: [%]

Generic Stored Procedures valid for all nodes of OS Type: SUNOS

Metrix:   SUN_SYSTEM      $iCpuNorm = iCpuLoad / iCpuCnt
                          Dscr: CPU load normalized, Unit: [%]

Node specific stored Procedures valid for OS Type: OPENVMS

Metrix:  PROCESS
         Node:    VMSTM1  $iCpuNorm= iCpuLoad / iCpus * 2
                          Dscr: CPU exec mode normalized [0..200%], Unit: [%]
```

Example 2:

In this example the METRIX and OSTYPE clause are defined to filter for all $iCpu* user-defined statistics that are member of the OpenVMS metric SYSTEM.

```
DQL> SHOW PROCEDURE $iCpu* METRIX SYSTEM OSTYPE OpenVMS;

Stored Procedures for: OPENVMS
Metrix:  SYSTEM           $iCpuNorm = iCpuLoad / iCpuCnt
                          Dscr: CPU load normalized, Unit: [%]
```

## *SHOW REGION*

The SHOW REGION command displays the default regional setting of the current DQL$ session and all regional settings defined in the regional setting table of the PERFDAT configuration database.

### Format

SHOW REGION;

### Description

The SHOW REGION command displays the default regional setting of the current DQL$ session and all regional settings defined in the regional setting table of the PERFDAT configuration database.

For more information about user-defined statistics please refer to the description of the DEFINE REGION and SET REGION command.

### Example

```
DQL> SHOW REGION;

 Default region setting:

 Name: DEFAULT
      Decimal Symbol: .
      List Seperator: ,
      Date Format   : dd-mmm-yyyy
      sMonths (ASCII): JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC


 Available region settings:

Name: DEFAULT
      Decimal Symbol: .
      List Seperator: ,
      Date Format   : dd-mmm-yyyy
      Months (ASCII): JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC

 Name: German
      Decimal Symbol: ,
      List Seperator: ;
      Date Format   : dd-mmm-yyyy
      Months (ASCII): JAN,FEB,MAR,APR,MAI,JUN,JUL,AUG,SEP,OKT,NOV,DEZ
```

## *SHOW STATISTICS*

The SHOW STATISTICS command displays the fields defined in a particular metric (table) of attached physical storage areas. The field name, datatype, field length and the field description is displayed.

### Format

SHOW STATISTICS FROM metric_name [ALIAS] alias_name [DATE] date;

### Description

The SHOW STATISTICS command displays the fields defined in a particular metric (table) of attached physical storage areas. The field name, datatype, field length and the field description is displayed.

The ALIAS and DATE clause are optional. They define the collection database alias and the day of interest (logical storage area) the SHOW command applies to.

The database alias can't be user-defined. DQL$ assigns the aliases when it starts up automatically. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

If the DATE clause is omitted the command applies to all physical storage areas of the collection database defined by the ALIAS clause. If the ALIAS clause is omitted too, the command applies to all attached physical storage areas.

### Example

The collection database COS03_10SEC has been attached. On 1-Dec-2003 the logical storage area consists of two physical storage area created on 19:02:00 and 17:35:00.

To display the field definitions of the metric (table) PROCESS stored in that physical storage areas enter:

```
DQL>SHOW STATISTICS FROM PROCESS ALIAS COS03_10SEC DATE 1-DEC-2003;

PROCESS METRIC DEFINITION of storage area COS03_10SEC_2003-12-01:17:35:00

--------------------------------------------------------------------------------
```

Field definitions of Metric: PROCESS

| FieldName | Type | Description |
|---|---|---|
| PrcName | STRING(32) [P] | ProcessName |
| Time | DATETIME(8) | Time |
| UserName | STRING(16) [I] | User Name Reference |
| ImageName | STRING(256) [I] | Image Name Reference |
| iDIO | FLOAT(4) | Direct IO rate |
| iBIO | FLOAT(4) | Buffered IO rate |
| iGlbMem | FLOAT(4) | Gbl Memory allocated by image |
| iPrcMem | FLOAT(4) | Private Memory allocated by image |
| iPfl | FLOAT(4) | PFL total |
| iPflFOR | FLOAT(4) | PFL on read faults |
| iPflFOW | FLOAT(4) | PFL on write faults |
| iPflFOE | FLOAT(4) | PFL on executive fault |
| iPageIO | FLOAT(4) | IO PageIOs |
| iCpuLoad | FLOAT(4) | CPU Load total |
| iKernel | FLOAT(4) | CPU Mode kernel |
| iExec | FLOAT(4) | CPU Mode exec |
| iSuper | FLOAT(4) | CPU Mode super |
| iUser | FLOAT(4) | CPU Mode user |
| iIOthres | FLOAT(4) | IO request threshold |
| iMemthres | FLOAT(4) | Memory usage threshold |
| iCputhres | FLOAT(4) | CPU load threshold |

Element count 47

---

PROCESS METRIC DEFINITION of storage area COS03_10SEC_2003-12-01:19:02:00

---

Field definitions of Metric: PROCESS

| FieldName | Type | Description |
|---|---|---|
| PrcName | STRING(32) [P] | ProcessName |
| Time | DATETIME(8) | Time |
| UserName | STRING(16) [I] | User Name Reference |
| ImageName | STRING(256) [I] | Image Name Reference |
| iDIO | FLOAT(4) | Direct IO rate |
| iBIO | FLOAT(4) | Buffered IO rate |
| iGlbMem | FLOAT(4) | Gbl Memory allocated by image |
| iPrcMem | FLOAT(4) | Private Memory allocated by image |
| iPfl | FLOAT(4) | PFL total |
| iPflFOR | FLOAT(4) | PFL on read faults |
| iPflFOW | FLOAT(4) | PFL on write faults |
| iPflFOE | FLOAT(4) | PFL on executive fault |
| iPageIO | FLOAT(4) | IO PageIOs |
| iCpuLoad | FLOAT(4) | CPU Load total |
| iKernel | FLOAT(4) | CPU Mode kernel |
| iExec | FLOAT(4) | CPU Mode exec |
| iSuper | FLOAT(4) | CPU Mode super |

| | | |
|---|---|---|
| iUser | FLOAT(4) | CPU Mode user |
| iIOthres | FLOAT(4) | IO request threshold |
| iMemthres | FLOAT(4) | Memory usage threshold |
| iCputhres | FLOAT(4) | CPU load threshold |

Element count 53

---

All fields marked with [P] are members of the element key (index).All fields marked with [I] are informational fields. These fields arenot visible to the GUI.

## *SHOW VERSION*

The SHOW VSERION command displays the version of the DQL$ utility and DQL$SRV of the node the current DQL$ session is connected to.

### Format

SHOW VERSION;

### Description

The SHOW VERSION command displays the version of the DQL$ utility and DQL$SRV of the node the current DQL$ session is connected to.

### Example

```
DQL> SHOW VERSION;

DQL$ utility: V4.2
DQL$SRV on node VMSTM1: V4.2
DQL$SRV on node VMSTM2: V4.2
```

## *SHOW VIEW*

The SHOW VIEW command displays the cluster views configured on the local node (content of the local cluster view database).

### Format

SHOW VIEW view_name;

### Description

The SHOW VIEW command displays the cluster views configured on the local node (content of the local cluster view database).

For more information about cluster views please refer to the description of the DEFINE VIEW command.

### Example

```
DQL> SHOW VIEW;
          View                    referenced Aliases
-------------------------------------------------------------------
           RLP                    ADMIN3_DEFAULT
                                  ADMIN4_DEFAULT
                                  TRANS3_DEFAULT
                                  TRANS4_DEFAULT
         VIECLU                   BCSXTC_DEFAULT
                                  VMSTM1_DEFAULT
                                  VMSTM2_DEFAULT
         VMSALL                   VMSTM2_DEFAULT
                                  VMSTM1_DEFAULT
```

## *UPDATE HEADER*

The UPDATE HEADER command can be applied to modify any header attributes of a logical storage area or a whole collection database.

### Format

```
UPDATE HEADER ALIAS alias_name [DATE date]
                                        ATTRIBUTE attr_name= attr_value;
```

### Description

This command modifies header attributes of a logical storage area or a whole database.

The ALIAS clause specifies the alias of the collection database. That database alias cannot be user-defined. DQL$ assigns database aliases automatically when it starts up. The collection database aliases available are displayed when you apply the SHOW DATABASE command. These aliases have the format:

NodeName_CollectionProfile

E.g. the database alias of the database created by OpenVMS performance data collections started with the collection profile 2MIN on node BCSXTC is BCSXTC_2MIN.

If you want to modify the header attributes of a logical storage area (all data files that have been created on the same day) of a collection database, the DATE clause is mandatory. Use OpenVMS date format to define the particular day of interest. If you omit the DATE clause the header attribute defined by the ATTRIBUTE clause of all data files (physical storage areas) of the collection database defined by the ALIAS clause are modified.

The ATTRIBUTE clause is mandatory. The attr_value parameter in the ATTRIBUTE clause defines the new value to be assigned to the header attribute defined by the attr_name parameter.

The header attribute names of a collection database can be displayed with the SHOW HEADER command after a collection database has been attached with the ATTACH command.