VSI OpenVMS

PERFDAT V4.8

Application Programming Interface

Users Guide

February 2019

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

Contents

Preface

This manual provides detailed usage and reference information on the VSI PERFDAT application programming interface.

Audience

This manual is intended for system and application programmers who require a basic understanding of how to use VSI PERFDAT API routines to insert data into the distributed VSI PERFDAT collection database.

The reader should be familiar with

- VSI PERFDAT– Architecture and Technical Description
- VSI PERFDAT– PERFDAT_MGR Reference Manual

Document Structure

Chapter 1	Introduction, Architecture and Technical description
Chapter 2	VSI PERFDAT C API reference section
Chapter 3	Program examples

Conventions Used in this Manual

Special	in examples indicates text that the system displays or user typed 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 simultaneously pressing the appropriate key.

Introduction

The aim of this section is to provide information about:

- VSI PERFDAT API features
- basic architecture
- using the VSI PERFDAT API
- configuration requirements
- managing application data collections
- floating point format considerations
- required user privileges

For a detailed description of all the API routines available, please refer to the VSI PERFDAT API reference section in this manual.

1.1 General Description

VSI PERFDAT provides an easy to use C programming interface (API) to insert any type of performance data collected by the components (programs) of an application directly into the distributed VSI PERFDAT performance database.

The VSI PERFDAT installation procedure provides two object libraries that contain the API routines:

- PERFDAT$LIBRARY:PERFDAT_API_AXP.OLB
  Alpha object library
- PERFDAT$LIBRARY:PERFDAT_API_IA64.OLB
  I64 object library

1.2 Features

The use of the VSI PERFDAT API provides several advantages:

- The programmer does not have to worry about when to open or close a data file. Data files are automatically created, opened and closed as defined by the VSI PERFDAT design rules.
- The VSI PERFDAT environment handles data files created by an application using the API as if these data files had been created by any of the VSI PERFDAT data collectors (OpenVMS, SNMP extension, EVA extension).
  - Application data files are automatically managed by the VSI PERFDAT archive and housekeeping processes reliably and unattended (for more information about VSI PERFDAT archiving and housekeeping please refer to the manual *VSI PERFDAT-Architecture and Technical Description*)
  - Trend, capacity and baseline report profiles can be defined for application data collections. These reports are automatically processed by the VSI PERFDAT auto trend engine (for more information about the VSI PERFDAT auto trend engine please refer to the manual *VSI PERFDAT- Architecture and Technical Description*).
- The API does not create separate data files for each process of an application but inserts the data provided by all processes of an application running on the same node into the same data file. This feature reduces the number of data files.
- Application data collections can be managed with the VSI PERFDAT management utility PERFDAT_MGR in the same way using the same commands as if one was managing OpenVMS, SNMP or EVA data collections without any programming effort, code change or the need of restarting the application.
  - Application data collections can be stopped at any time.

    - Application data collections are profile controlled as with other data collections created by one of the VSI PERFDAT data collectors (OpenVMS data collector, SNMP extension, EVA extension). Application data collection profiles can be user defined. A collection profile defines the sample interval and the metrics that will be enabled when a data collection is started using a particular collection profile.
    - Once an application data collection has been stopped it can be started again with a different collection profile.
    - Online alerting can be enabled or disabled during run-time.
    - The status of application data collections can be monitored.
- Time concurrency
  Performance data is typically provided as averaged values like MB/sec or Transactions/sec. If an application consists of several processes which provide performance data as averaged values, it is important in terms of performance analysis that all these processes gather, calculate and provide the data at the same time so that this data can be compared and correlated to each other without any pre-processing. The VSI PERFDAT API triggers all processes of an application at the same time to collect, to calculate and to insert the data records into the metrics of the collection database regardless on which node the processes are running on within an OpenVMS cluster. Thus, the program developer does not have to care about such timing issues as described herein.

1.3 Using the C Programming API

The aim of this section is to provide architectural and technical background information about the VSI PERFDAT API and to explain the basic steps of how to use the VSI PERFDAT API routines. For a detailed description of all API routines available, please refer to the VSI PERFDAT API reference section in this manual.

Basically only two API calls are required to insert data into the distributed VSI PERFDAT performance database.

1. PerfDatAPIInit()

   This routine initializes the VSI PERFDAT API and requests the VSI PERFDAT DQL interface to associate a collection database with the calling program. The handle of this association is the application database association name which has to be passed as an input parameter to PerfDatAPIInit(). This routine has to be called from the main routine of the calling program.

   Note

   The application name passed to the initialization routine of the VSI PERFDAT API must not exceed 10 characters.

   The DQL interface checks if an application collection database descriptor with the same name as specified by the application name parameter exists in the descriptor table of the VSI PERFDAT configuration database. An application collection database descriptor contains the record definitions for all metrics of an application collection database. Such an application collection database descriptor is required by the VSI PERFDAT API in order to create or access an application collection database (The next section of this document provides detailed information about application collection databases). If this check fails the API initialization fails.

   Due to this implementation the design rules of the distributed VSI PERFDAT collection database allow that several programs on the same node can access the same application collection database as illustrated in Fig. 1.1 (see next section of this document).

   If such an application collection database descriptor exists, PerfDatAPIInit()registers the collection notification method passed to the routine. The VSI PERFDAT API guarantees that all application programs that are associated with the same application are triggered at the same time to collect, to calculate and to insert the data records into the metrics (tables) of the associated application collection database.

   The API always triggers such a collection event at the end of a sample interval. The sample interval is defined by the collection profile used to start the application data collection. Two different methods can be defined of how the calling program will be notified to insert data records:
   - Event flag
     If an event flag number greater than 0 is passed to PerfDatAPIInit() this event flag will be set at the end of a sample interval.
   - AST routine call

If a valid address of a user AST routine is passed to PerfDatAPIInit() this routine will be called at the end of the sample interval except if an event flag number greater than 0 has been specified. If both, an event flag number greater than 0 and a valid user AST routine address are specified only the event flag is set. The user defined AST routine will not be called.

Note

In contrast to other VSI PERFDAT standard collections it is important to emphasize that all programs associated with an application using the VSI PERFDAT API running on any of the OpenVMS cluster members will be triggered at the same time to insert data and not just the programs that are started on a particular node.

Using the AST notification method requires no additional main loop coding compared to the event flag notification method (the calling program has to wait for the event flag to be set by the VSI PERFDAT API in the main loop). The disadvantage of the AST notification method is that the execution of the data collection and data insert processing routine is not under the control of the calling program since the AST routine is directly called from the VSI PERFDAT API at the end of each sample interval.

Neither of these notification methods are triggered by the API unless a data collection is started for the application that the program belongs to. Thus, after the event notification method has been registered the PerfDatAPIInit()routine checks if an entry exists in the auto-start table of the VSI PERFDAT configuration database for the application specified by the application name parameter and the node the program was started on.

If no such auto-start entry exists the routine immediately returns to the caller. Otherwise the data collection defined in the auto-start entry will automatically be started. This means that the API creates or, if the collection database already exists, attaches the associated collection database and starts notifying the calling program to collect its data at the end of each sample interval as specified by the collection profile defined in the auto-start entry.

Note

Due to the design of the VSI PERFDAT API a program can be associated with up to 16 application collection databases. Each application database association has to be explicitly initialized by callingPerfDatAPIInit().

2. PerfDatAPIInsertRecord() or PerfDatAPIAssocInsertRecord()

PerfDatAPIInsertRecord() is recommended to be called if the calling program is associated with only one application collection database. It inserts a data record into the application collection database. This routine requires two input parameters:

- Metric name
  This argument defines the metric (table) of the associated application collection database to insert the data record addressed by the data record descriptor. If no such metric exists in the associated application collection database the routine fails.
- API data record descriptor
  The API data record descriptor contains the pointer to a buffer that contains the data record and the length of the data record. The type definition of the API data descriptor record is defined in the header file:
  - PERFDAT$INCLUDE:PERFDAT_API.H.

If the program is associated with more than one application collection database PerfDatAPIAssocInsertRecord()has to called to insert a data record into a particular application collection database the program is associated with. This routine requires three input parameters:

- Application name
  This argument specifies the name of an application data collection handle. An application data collection handle refers the application collection database to insert the data record addressed by the data record descriptor. If no application data collection (application database association) exists with the same name the routine fails.
- Metric name
  This argument defines the metric (table) of the associated application collection database to insert the data record addressed by the data record descriptor. If no such metric exists in the associated application collection database the routine fails.
- API data record descriptor
  The API data record descriptor contains the pointer to a buffer that contains the data record and the length of the data record. The type definition of the API data descriptor record is defined in the header file:
  - PERFDAT$INCLUDE:PERFDAT_API.H.

PerfDatAPIInsertRecord()and PerfDatAPIAssocInsertRecord()have to be called from the routines that are triggered whenever the API notifies the program to collect and insert data.

The data types used by the API routines and the C prototypes of the API routines are defined in the header file PERFDAT$INCLUDE:PERFDAT_API.H. This header file has to be included in each module of the application program that calls the VSI PERFDAT API routines.

Beside these three routines the VSI PERFDAT API provides several other callable routines. As has been stated previously it is not the aim to describe all of these in detail in this section. For a detailed description of all API routines please refer to the VSI PERFDAT API reference section in this manual

1.4 Application collection database

Once an application data collection has been started, because either:

- An auto-start entry exists for the application, and the node the program that is associated with the application exists when the program is started (see previous section).
- The application data collection is started using the PERFDAT_MGR utility START COLLECTION command.
- The data collection is started directly from the program by calling the PerfDatAPIStartColl() or PerfDatAPIAssocStartColl()

the programmer does not have to care about closing and opening data files according to the VSI PERFDAT database design rules. The data file management is automatically performed by the VSI PERFDAT API.

Due to the VSI PERFDAT database design rules the VSI PERFDAT API creates one collection data file per day and node for a particular application data collection regardless of how many programs associated with the application are started on that node and regardless if some or all of the programs are re-started during the day. Thus, all programs running on the same node and associated with the same application collection database access the same application collection database files. Fig. 1.1 illustrates this behavior.

An application data collection is defined by the application name and the collection profile used to start the application data collection.

At day change the data file is closed and a new data file is created if there are still some active programs associated with the application collection database.

The sum of all data files created on one node for a particular application and collection profile is called the application collection database.

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

ApplicationName@Nodename_collection-profile

Thus, if at least one program associated with the application TEST is running on node VMSTM1 and the application data collection was defined to start using the collection profile DEFAULT the VSI PERFDAT API creates the application collection database TEST@VMSTM1_DEFAULT. If the same program is started on another OpenVMS node a new application collection database is created. For example if the same program is alsostarted on node VMSTM2 the application database TEST@VMSTM2_DEFAULT will be created.

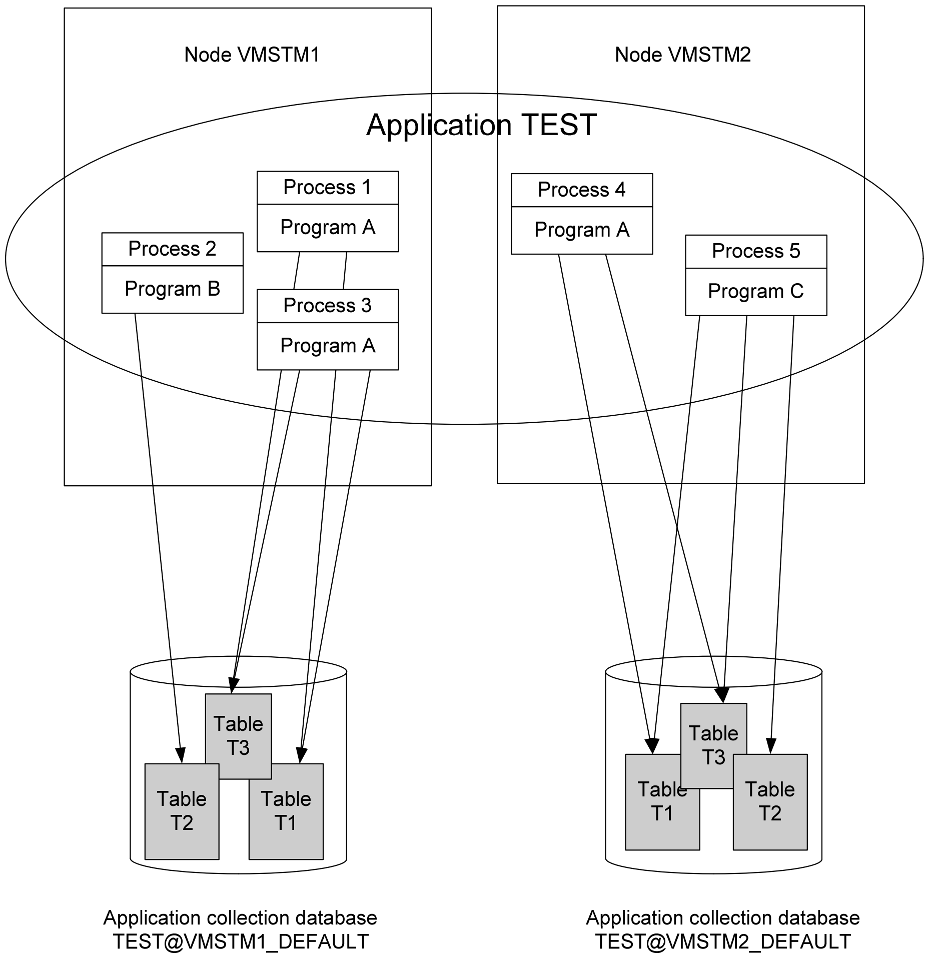

Fig. 1.1 Application collection database access when using the VSI PERFDAT API to insert data into the distributed VSI PERFDAT collection database. In this example the programs A, B and C are members of the same application. Program A is started twice on node VMSTM1 and once on node VMSTM4. Program B runs only on node VMSTM1, and program C runs only on node VMSTM2. Since these programs belong to the same application all processes running one of these programs on a particular node access the same application database. Assuming the application data collection was started with the DEFAULT collection profile the processes 1, 2 and 3 access the application collection database TEST@VMSTM1_DEFAULT and the processes 4 and 5 access the application database TEST@VMSTM2_DEFAULT. As shown in this example the programs that

are members of the same application can, but do not have to insert data into all metrics of an application collection databases.

1.5 *Configuration requirements*

1.5.1 Collection database descriptor

As described in the previous section an application collection database descriptor must exist in the descriptor table of the VSI PERFDAT configuration database with the same name as the application name passed in the API initialization routine PerfDatAPIInit().

Such a database descriptor can be defined by loading a descriptor file that contains the required definitions into the VSI PERFDAT configuration database using the PERFDAT_MGR command:

```
$ MCR PERFDAT_MGR LOAD METRIX descriptor-load-file
```

A descriptor load file must contain:

- A system definition block
- A metric (table) description section

In the context of the VSI PERFDAT API the system definition block defines the application that can be associated with the collection database defined in the load file.

The metric descriptor section contains the record descriptors for all metrics (tables) of the application collection database defined by the load file.

The system definition block starts with the keyword

- OS_TYPE:

and ends with the keyword

- OS_TYPE_END:

Note

The colon character ":" at the end of the keywords marks the start and end of a metric descriptor block and is mandatory.

The system definition block contains a single line with three parameters as shown in the example below:

```
OS_TYPE:
        TEST:    COM$_APP:       PERFDAT$_NODE_APPL:
OS_TYPE_END:
```

The first parameter (TEST) defines which application can be associated with the collection database. The second parameter (COM$_APP) defines that the collection database is accessed via the VSI PERFDAT API, and the third parameter (PERFDAT$_NODE_APPL) is required by the DQL interface when a user accesses the collection database via the GUI for visualizing and analyzing the collected application data.

Thus, the system definition block in this example defines that the metric descriptor section of the load file contains the metric descriptors for a collection database that can be associated with application TEST.

Note

Use exactly the same keywords to define the second and third parameters in the system definition block as shown in the example above. Only the first parameter defining the application that can be associated with the collection database is user definable.

The metric descriptor section defines the record layout (field name, field description, field data type, field length and unit of the values stored in this data field) of the metrics (tables) of the collection database defined by the load file. Each metric is defined by a metric descriptor block. Since a collection database can contain up to 99 metrics, the metric descriptor section can contain up to 99 metric descriptor blocks.

A metric descriptor block starts with the keyword

- METRIX_*metric-name*:

and ends with

- METRIX_*metric-name*_END:

The *metric-name* parameter specifies the name of the metric defined within the metric descriptor block. The metric name (not case sensitive) has to be passed as one of the required input parameter to the API routine PerfDatAPIInsertRecord() when inserting a data record into a particular metric (table) of the application collection database.

Note

The colons character ":" at the end of the keywords marks the start and end of a metric descriptor block and is mandatory.

A metric descriptor block contains the field definitions of all fields of the metric. Each line describes one data field of the metric. Five properties have to be defined for each data field:

- Field name
  The field name is used by the DQL interface to address a particular data field in a metric. The field name is a string with a maximum length of 15 characters and has to be unique within a metric descriptor block.

- Data type and options

- Data type keywords
  - FIELD$_STRING
    Data field contains a zero terminated string.
  - FIELD$_INTEGER
    Data field contains an integer.
  - FIELD$_UNSIGNED
    Data field contains an unsigned integer.
  - FIELD$_QUAD
    Data field contains a quad word.
  - FIELD$_FLOAT
    Data field contains a float.
  - FIELD$_DATETIME
    Data field contains date and time (quad word).

- Data option keyword
  - FIELD$_PRIMKEY
    It indicates that the content of the field is part of the element key. The element key is used by the DQL interface to select all data of a particular element from a metric. For example, the data collection files created by the OpenVMS data collector contain the metric PROCESS. The OpenVMS data collector stores the performance data of the OpenVMS processes running on a node into this metric. This metric contains one primary key field - the string field PrcName - which contains the process name of a particular OpenVMS process. The DQL interface uses this field to select all data stored in the PROCESS metric to select the data of a particular OpenVMS process collected by the OpenVMS data collector.

    This option has to be assigned to at least one of the data fields defined by a metric descriptor block. Otherwise the metric descriptor block is invalid. The primary key option can be assigned to a maximum of 3 data fields. Assigning this field option to more then 3 data fields causes unpredictable behavior of the DQL interface.

    Note

    The primary key data option can be assigned to any data field of any data type except to data fields of type FIELD$_DATETIME.

  - FIELD$_INFO
    This data option indicates that the field content is only informational, and will not be visible to the GUI.

These two data options FIELD$_PRIMKEY and FIELD$_INFOare mutually exclusive.

Use the OR (|) sign to separate the data type and data option.
Examples:

FIELD$_STRING|FIELD$_PRIMKEY
The data field is a string and part of the element key.
FIELD$_INTEGER|FIELD$_INFO
The data field is an integer and not visible to the GUI.

- Length of the data field
  This parameter defines the field length in bytes.

Note

If the field type is FIELD$_DATETIME, always enter 8 (quadword length)

Note

If the field type is FIELD$_STRING, the data field contains a zero terminated string. Thus, the maximum length of the string that can be stored in such a string data field is the length defined herein minus 1. For example if you define a length of 32 characters the maximum length of the string is 31 characters.

- Short description of the data field
  A comment that briefly describes the contents of the data field. The maximum length of the field description is 64 characters.

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

The data field properties have to be separated by a colon character (“:”).

Note

One (and only one) time data field must be defined in a metric descriptor block and at least one data field has to be defined as a primary key field (except the time data field – see above). The data fields in a metric descriptor have to be ordered according to the following rules:

- All primary key data fields
- Time data field
- Remaining data fields.

Example of a metric descriptor block:

METRIX_PRCIO:

```
        Process: FIELD$_STRING|FIELD$_PRIMARY:32:        Process name: [N/A]:
        Time:    FIELD$_DATETIME:8:                      Time:[s]:
        DIO:     FIELD$_FLOAT:4:             Direct I/O rate: [1/s]:
        BIO:     FIELD$_FLOAT:4:             Buffered I/O rate: [1/s]:

METRIX_PRCIO_END:
```

The metric descriptor block in the example above defines the metric PRCIO. The record of the metric contains four data fields. The *Process* data field is the only primary key data field. Thus, it is the first entry in the descriptor block followed by the required time data field. The remaining data fields DIO and BIO of the metric PRCIO both have floating point data types.

The VSI PERFDAT installation procedure provides an example database descriptor load file:

- PERFDAT$EXAMPLES:PERFDAT_API_TEST.CFG

1.6 Managing application data collections

As described in the previous sections application data collections can be managed without any programming effort using the VSI PERFDATPERFDAT_MGR utility. Once an application database descriptor has been loaded into the VSI PERFDAT configuration database the user can:

- Add/Modify application collection profiles.
- Add/Modify auto-start entries for the application data collection.
- Add/Modify report profiles used by the auto-trend engine to extract trend or capacity reports from the data collection files created by the application programs using the VSI PERFDAT API.
- Start/Stop application data collections without affecting the application programs that run the application data collection.
- Enable/disable online alerting for the application data collection without affecting the application programs that run the application data collection.
- Monitor the status of an application data collection.

1.6.1 Application collection profiles

As with any other VSI PERFDAT data collection created by any of the VSI PERFDAT data collectors (OpenVMS data collector, SNMP extension, EVA extension), application data collections are profile controlled. In order to add, modify or delete an application data collection profile use the PERFDAT_MGR commands:

- ADD RPOFILE
- MODIFY PROFILE

- DELETE PROFILE

The use of the /OS_TYPE qualifier is mandatory. The value assigned to the /OS_TYPE qualifier specifies the application the collection profile is valid for. An application collection database descriptor with the same name must exist in the VSI PERFDAT configuration database. Otherwise the PERFDAT_MGR commands fails.

Example:

```
$ MCR PERFDAT_MGR ADD PROFILE DEFAULT/OS_TYPE=TEST
```

This command starts the collection profile wizard to configure the collection profile DEFAULT valid for the application TEST.

For detailed information about adding, modifying or deleting collection profiles using the PERFDAT_MGR utility please refer to the utility's online help or to the manual:

- VSI PERFDAT – PERFDAT_MGR Reference Manual

1.6.2 Application report profiles

Application data collections can be processed by the VSI PERFDAT auto-trend engine as with any other collection database created by any other data collector provided with VSI PERFDAT (OpenVMS data collector, SNMP extension, EVA extension). In order to extract trend and capacity reports from a collection database a report profile has to be defined.

Report profiles for application data collections can be managed using the PERFDAT_MGR commands:

- ADD REPORT
- MODIFY REPORT
- DELETE REPORT

The use of the /OS_TYPE qualifier is mandatory. The value assigned to the /OS_TYPE qualifier specifies the application the report profile is valid for. An application collection database descriptor with the same name must exist in the VSI PERFDAT configuration database. Otherwise the PERFDAT_MGR commands fails.

Example:

```
$ MCR PERFDAT_MGR REPORT PROFILE WEEK/OS_TYPE=TEST
```

This command starts the report profile wizard to configure the collection report WEEK valid for the application TEST.

For more detailed information about adding, modifying or deleting report profiles using the PERFDAT_MGR utility please refer to the utility's online help or to the manual:

- VSI PERFDAT – PERFDAT_MGR Reference Manual

1.6.3 Application auto-start entries

As described in section1.3 Using the C Programming API the initialization routine of the API searches for an entry in the VSI PERFDAT auto-start table of the configuration database valid for the node a program using the VSI PERFDAT API was started on and the application name passed to the initialization routine. If such an auto-start entry exists the VSI PERFDAT API starts the application data collection processing without any additional programming effort.

To add, modify or delete an auto-start entry for a particular node and application use the PERFDAT_MGR commands:

- ADD AUTOSTART
- MODIFY AUTOSTART
- DELETE AUTOSTART

The use of the /OS_TYPE qualifier is mandatory. The value assigned to the /OS_TYPE qualifier specifies the application the auto-start entry is valid for. An application collection database descriptor with the same name must exist in the VSI PERFDAT configuration database. Otherwise the PERFDAT_MGR commands fails.

Example:

```
$ MCR PERFDAT_MGR ADD AUTOSTART VMSTM1/OS_TYPE=TEST
```

This command starts the auto-start wizard to add an auto-start entry for node VMSTM1 valid for application TEST.

For more detailed information about adding, modifying or deleting auto-start entries using the PERFDAT_MGR utility please refer to the utility's online help or to the manual:

- VSI PERFDAT – PERFDAT_MGR Reference Manual

1.6.4 Start/Stop of an application collection

Application data collections can be started and stopped during the run-time of the programs that use the VSI PERFDAT API to insert data records into an application collection database without any programming effort and without affecting the running program.

To start or to stop an application data collection use the PERFDAT_MGR commands:

- START COLLECTION*profile-name*
- STOP COLLECTION*profile-name*

The use of the /OS_TYPE qualifier is mandatory. The value assigned to the /OS_TYPE qualifier addresses the application that is affected by the start or stop command. The *profile-name* parameter specifies an existing collection profile for the application defined by the /OS_TYPE qualifier.

The START COLLECTION command fails if:

- The collection profile specified by the *profile-name* parameter does not exist in the VSI PERFDAT configuration database for the application defined by the /OS_TYPE qualifier.
- No program is running on any cluster node that is associated with the application defined by the /OS_TYPE qualifier.
- An application data collection using another collection profile is already active for the application defined by the /OS_TYPE qualifier.

The STOP COLLECTION command fails if:

- The application data collection had not been started with the collection profile defined by the *profile-name* parameter.
- No program associated with the application defined by the /OS_TYPE qualifier is running on any cluster node.

When starting an application data collection with the START COLLECTION command auto-start entries are automatically created or modified (if they already exist) for all cluster members. This guarantees that if a program associated with the application defined by the /OS_TYPE qualifier is restarted after the start command has been executed that this program will automatically start data collection processing.

Note

The /NODE qualifier is not valid for starting or stopping an application data collection. If an application data collection is started or stopped all programs associated with the application defined by the /OS_TYPE qualifier running on any OpenVMS cluster member are triggered to start or to stop data collection processing.

Example:

```
$ MCR PERFDAT_MGR START COLLECTION DEFAULT/OS_TYPE=TEST
```

This command triggers all programs associated with application TEST running on any OpenVMS cluster member to start data collection processing using the collection profile DEFAULT.

For more detailed information about the following tasks, please refer to the PERFDAT_MGR utility online help or to the manual *VSI PERFDAT_MGR Reference Manual:*

- Starting and stopping data collections
- Adding, modifying or deleting auto-start entries

1.6.5 Enable/Disable online alerting

Online alerting can be enabled or disabled during the run-time for any application data collection as with any other VSI PERFDAT data collection.

To enable or disable online alerting for a particular application data collection use the PERFDAT_MGR commands:

- ENABLE ALERT
- DISABLE ALERT

The use of the /OS_TYPE qualifier is mandatory. Online alerting will be enabled for the application addressed by the value assigned to the /OS_TYPE qualifier.

When online alerting is enabled for an application data collection the statistics defined within an alert definition file are monitored if they exceed specified thresholds. The alert definition file must be defined whenever the ENABLE ALERT command is executed since no default alert definition file exists for application data collections. Thus, the /ALERT_FILENAME qualifier is mandatory when the user enables online alerting for application data collections.

For detailed information about online alerting and alert definition files please refer to the PERFDAT_MGR utility online help or to the following manuals:

- VSI PERFDAT – PERFDAT_MGR Reference Manual
- VSI PERFDAT – Architecture and Technical Description

1.6.6 Displaying the application data collection status

To display the status of an application data collection use the PERFDAT_MGR command:

- SHOW COLLECTION

As with any other VSI PERFDAT data collections the SHOW COLLECTION command displays the collection profile used to run a particular application data collection and in addition the process names of all processes that run programs associated with the same application.

All qualifiers of the SHOW COLLECTION command are valid for displaying the status of application data collections.

Example:

```
$ MCR PERFDAT_MGR SHOW COLLECTION/OS_TYPE=TEST

PROFILE: DEFAULT            Application: TEST@VMSTM1

Collection sample interval: 120 sec
PRCIO Metrix enabled: TRUE
PRCMEM Metrix enabled: TRUE

Online alerting enabled: FALSE
Collection data can be accessed online: TRUE

Processes running the collection: _FTA5:@VMSTM1
                                  _FTA8:@VMSTM1


PROFILE: DEFAULT            Application: TEST@VMSTM4

Collection sample interval: 120 sec
PRCIO Metrix enabled: TRUE
PRCMEM Metrix enabled: TRUE

Online alerting enabled: FALSE
Collection data can be accessed online: TRUE

Processes running the collection: _FTA3:@VMSTM4
```

In this example the status of the application data collection associated with the application TEST is displayed. The application data collection was started with collection profile DEFAULT. The SHOW command lists the settings of this profile and the processes that are associated with this application.

For detailed information about displaying the status of data collections please refer to the PERFDAT_MGR utility's online help or to the following manual:

- VSI PERFDAT – PERFDAT_MGR Reference Manual

1.7 Floating point format

The VSI PERFDAT API as well as the DQL interface uses the G_FLOAT format for double precision floating point variables and VAX F_FLOAT for single precision floating point variables. Thus, if any metric of an application collection database contains single precision floating point data fields, make sure that these floating point data are passed in VAX F_FLOAT format to the API routine PerfDatAPIInsertRecord(). Otherwise these data fields will not be readable if a user accesses this data either using the DQL$ command line utility or the VSI PERFDAT GUI.

G_FLOAT/VAX F-FLOAT are the default floating point formats on Alpha but not on IA64. The default floating point formats on IA64 are IEEE T_FLOAT/IEEE S-FLOAT. Thus, one has to take special care of floating point variables when using the VSI PERFDAT API to insert application data into the distributed VSI PERFDAT collection database on IA64.

Options:

- Compile all programs using VSI PERFDAT API routines with /FLOAT=G_FLOAT on IA64
- Do not define any floating point data field in a metric of an application collection database.
- Convert the floating point values from the internal floating format into VAX F_FLOAT within your program format before inserting the data using thePerfDatAPIInsertRecord() API routine.

1.8 User privileges

Only users that at least have the NETMBX, TMPMBX and SYSLCK privileges assigned can run programs that call the VSI PERFDAT API routines. In addition the PERFDAT_API identifier has to be granted to the user regardless if the user is a full privileged user (i.e. SYSTTEM) or not.

To grant a user the PERFDAT_API identifier, use the OpenVMS AUTHORIZE utility.

For more detailed information about how to grant a user an identifier please refer to the OpenVMS documentation.

2

API Routine Reference Section

This section contains detailed descriptions of the routines provided by the VSI PERFDAT application programming interface.

2.1 PerfDatAPIAssocDisableAlert

Disable online alerting for a particular application database association.

C Prototype

```
int  PerfDatAPIAssocDisableAlert (char *sAppName);
```

Arguments

sAppName

API usage:	application database association handle
Type:	zero terminated character-coded text string
Access:	read-only
Mechanism:	by reference

The sAppName argument contains the reference to a zero terminated character-coded text string that contains the name of a particular application database association handle. The maximum length is 10 characters.

Description

This routine disables online alerting for a particular application database association specified by the sAppName argument.

If the VSI PERFDAT API is not initialized the routine returns PD$_NOINIT to the caller. If the application data collection specified by sAppName does not exist (PerfDatApiInit() has not been called for this application database association) PD$_ASSOCNOTEXIST is returned. If online alerting is already disabled the routine returns PD$_ALERTDIS.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Condition values returned

PD$_SUCCESS	Online alerting has been successfully disabled.
PD$_NOINIT	VSI PERFDAT API is not initialized.
PD$_ALERTDIS	Online alerting is already disabled.
PD$_ASSOCNOTEXIST	Application database association handle defined by sAppName does not exist.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.2 *PerfDatAPIAssocEnableAlert*

Enable online alerting for a particular application database association.

C Prototype

```
int  PerfDatAPIAssocEnableAlert (          char *sAppName,
                                           char *sAlertFileName);
```

Arguments

sAppName

API usage:	application database association handle
Type:	zero terminated character-coded text string
Access:	read-only
Mechanism:	by reference

The sAppName argument contains the reference to a zero terminated character-coded text string that contains the name of a particular application database association handle. The maximum length is 10 characters.

sAlertFileName

API usage:	alert definition file name
Type:	zero terminated character-coded text string
Access:	read only
Mechanism:	by reference

The sAlertFileName argument contains the reference to a zero terminated character-coded text string that specifies the alert definition file name used by the online alerting sub-system to check performance alert conditions.

Description

This routine enables online alerting for a particular application database association defined by sAppName with the alert definition file specified by the sAlertFileName argument.

If the VSI PERFDAT API is not initialized the routine returns PD$_NOINIT to the caller.If the application database association specified by sAppName does not exist (PerfDatApiInit() has not been called for this application database association) PD$_ASSOCNOTEXIST is returned. If data collection processing is inactive the routine returns PD$_SHUTDOWN.

If online alerting is already enabled, online alerting will be automatically disabled and re-enabled with the alert definition file specified by the sAlertFileName argument.

If the alert definition file does not exist SS$_NOSUCHENTRY is returned to the caller.

If the sAlertFileName argument refers to a zero length text string or the reference is invalid (i.e. NULL pointer) this routine searches for an entry in the auto-start table of the VSI PERFDAT configuration database valid for the node on which the program is running and the application the program is associated with. If no auto-start entry exists SS$_NOSUCHENTRY is returned to the caller.

If an auto-start entry exists the routine checks whether an alert definition file is defined in the auto-start entry. If neither an alert definition file has been defined nor the alert definition file defined in the auto-start entry exists SS$_NOSUCHENTRY is returned. Otherwise online alerting is enabled with the alert definition file defined in the auto-start entry.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Condition values returned

PD$_SUCCESS	Online alerting has been successfully enabled
PD$_NOINIT	VSI PERFDAT API is not initialized.
PD$_SHUTDOWN	Data collection processing is inactive.
PD$_ASSOCNOTEXIST	Application database association handle defined by sAppName does not exist.
SS$_NOSUCHENTRY	Alert definition file does not exist.

Any condition values returned by the DQL interface of VSI PERFDAT

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.3 *PerfDatAPIAssocInsertRecord*

This routine inserts a data record into a metric of a particular application collection database that has been associated with the calling program.

C Prototype

```
int  PerfDatAPIAssocInserRecord (        char *sAppName,
                                         char * sMetrix,
                                         tPerfDatAPIDataDsc *prData);
```

Arguments

sAppName

API usage:	application database association handle
Type:	zero terminated character-coded text string
Access:	read-only
Mechanism:	by reference

The sAppName argument contains the reference to a zero terminated character-coded text string that contains the name of a particular application database association handle. The maximum length is 10 characters.

sMetrix

API usage:	metric name of the associated collection database
Type:	zero terminated character-coded text string
Access:	read-only
Mechanism:	by reference

This argument defines the metric (table) of the associated application collection database to insert the data record addressed by the prData argument.

The sMetrix argument contains the 32 bit address pointing to a zero terminated character-coded text string.

prData

API usage:	data to insert
Type:	API data descriptor
Access:	read-only
Mechanism:	by 32-bit API data descriptor reference

This argument contains the 32-bit address pointing to an API data descriptor. An API data descriptor addresses the buffer that contains the data record to be inserted and the length of the data record.

Description

This routine inserts a data record into a particular metric of the associated application collection database specified by the sAppName argument.

The sMetrix argument defines the metric (table) of the associated collection database to insert the data record addressed by the data record descriptor prData. If no such metric exists in the associated collection database the routine fails.

The data record to be inserted into the metric defined by the sMetrix argument has to be passed to this routine by use of an API data descriptor. An API data descriptor (see the data type definition below) contains the address to a buffer containing the data record to be inserted into the metric and the length of the data record.

```
typedef struct perfdat$data_dsc
{
        long    dsc$l_length;
        void    *dsc$v_pointer;
} tPerfDatAPIDataDsc;
```

dsc$l_length	length of the data record to insert.
dsc$v_pointer	void pointer to the buffer containing the data record.

The data fields of the data record addressed by the dsc$v_pointer field of the API data descriptor have to be ordered according to the record definition of the metric specified by the sMetrix argument. The record definitions of all metrics of an application collection database are defined by a collection database descriptor stored in the VSI PERFDAT configuration database. A collection database descriptor of an application database can be defined by loading a descriptor load file. For more information about collection database descriptors and how to create a descriptor load file and how to define metric record descriptor please refer to the section 1.5.1 Collection database descriptor.

Any metric record descriptor has to contain one time data field (see 1.5.1 Collection database descriptor). The collection time is automatically inserted into the time field of the data record passed to this routine before it is inserted into the metric defined.

Condition values returned

PD$_SUCCESS	Data record has been successfully inserted into the metric defined by the sMetrix argument.
PD$_NOTINCOLL	Metric defined by the sMetrix argument is not enabled (has been disabled by the collection profile used to start the data collection) for the application data collection.

PD$_ASSOCNOTEXIST	Application database association handle defined by sAppName does not exist.
SS$_INVARG	SS$_INVARG is returned to the caller if: • The metric defined by the sMetrix argument does not exist in the application database associated with the program. • The data record passed to the routine does not match the data record definition of the metric defined by the sMetrix argument.
RMS$_DUP	Data record with the same primary key index already exist in the metric defined by the sMetrix argument.

Any condition values returned by RMS.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.4 *PerfDatAPIAssocIsAlertEnabled*

Tests if online alerting is enabled for a particular application database association.

C Prototype

```
int  PerfDatAPIAssocIsAlertEnabled (     char *sAppName,
                                         char *sAlertFileName,
                                         int iLen);
```

Arguments

sAppName

API usage:	application database association handle
Type:	zero terminated character-coded text string
Access:	read-only
Mechanism:	by reference

The sAppName argument contains the reference to a zero terminated character-coded text string that contains the name of a particular application database association handle. The maximum length is 10 characters.

sAlertFileName

API usage:	alert file name buffer
Type:	character-coded text string
Access:	write only
Mechanism:	by reference

The sAlertFileName argument contains the 32-bit address of a user defined string buffer.

If online alerting is enabled for the application database association, this routine copies the alert definition file name used by the online alerting sub-system into the string buffer as a null terminated character-coded text string.

iLen

API usage:	length of the alert file name buffer
Type:	integer
Access:	read only
Mechanism:	by value

Length of the user defined string buffer addressed by the sAlertFileName argument.

Description

This routine checks if online alerting is enabled for the application database association specified by the sAppName argument.

If online alerting is enabled for the application database association, this routine copies the alert definition file name used by the online alerting sub-system into the string buffer addressed by the sAlertFileName argument as a zero terminated character-coded text string. If the size of the user define string buffer is less than the string length of the alert definition file name SS$_BADPARAM is returned.

If the VSI PERFDAT API is not initialized the routine returns PD$_NOINIT to the caller.

If online alerting is disabled or data collection processing is inactive SS$_NOSUCHENTRY is returned.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Condition values returned

PD$_SUCCESS	Online alerting is enabled and the alert definition file used by the online alerting sub-system has been successfully copied into the user defined string buffer.
PD$_NOINIT	VSI PERFDAT API is not initialized.
PD$_ASSOCNOTEXIST	Application database association handle defined by sAppName does not exist..
SS$_BADPARAM	Online alerting is enabled but the alert definition file used by the online alerting sub-system cannot be copied into the user defined string buffer because the string buffer is too small.
SS$_NOSUCHENTRY	Data collection processing is inactive.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.5 *PerfDatAPIAssocIsCollStarted*

Tests if data collection processing is active for a particular application data collection.

C Prototype

```
int  PerfDatAPIAssocIsCollStarted (       char *sAppName,
                                          char *sProfileName,
                                          int iLen);
```

Arguments

sAppName

API usage:	application database association handle
Type:	zero terminated character-coded text string
Access:	read-only
Mechanism:	by reference

The sAppName argument contains the reference to a zero terminated character-coded text string that contains the name of a particular application database association handle. The maximum length is 10 characters.

sProfileName

API usage:	collection profile name buffer
Type:	character-coded text string
Access:	write only
Mechanism:	by reference

The sProfileName argument contains the 32-bit address of a user defined string buffer.

If data collection processing is currently active this routines copies the collection profile name used to start the application data collection into the string buffer as a zero terminated character-coded text string. The maximum length of a collection profile name is 48 characters. Thus, the length of the string buffer addressed by the sProfileName argument should be at least 48 characters.

iLen

API usage:	length of the collection profile name buffer
Type:	integer
Access:	read only
Mechanism:	by value

Length of the user defined string buffer addressed by the sProfileName argument.

Description

This routine checks if data collection processing is active for the application database association specified by thesAppName argument.

If data collection processing is currently active this routines copies the collection profile name used to start the application data collection into the string buffer addressed by the sProfileName argument as a zero terminated character-coded text string. The maximum length of a collection profile name is 48 characters. Thus, the length of user defined string buffer should be greater than 48 characters. If the size of the user define string buffer is less than the string length of the collection profile name SS$_BADPARAM is returned.

If the VSI PERFDAT API is not initialized the routine returns PD$_NOINIT to the caller.

If data collection processing is inactive SS$_NOSUCHENTRY is returned.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Condition values returned

PD$_SUCCESS	Data collection processing is active and the collection profile used to start the application data collection has been successfully copied into the user defined string buffer.
PD$_NOINIT	VSI PERFDAT API is not initialized.
PD$_ASSOCNOTEXIST	Application database association handle defined by sAppName does not exist.
SS$_BADPARAM	Data collection processing is active but the collection profile name used to start the application data collection cannot not be copied into the user defined string buffer because the size of the string buffer is too small.
SS$_NOSUCHENTRY	Data collection processing is inactive.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.6 PerfDatAPIAssocIsInit

Tests if a particular application database association exists and has been initialized by a previous call to PerfDatAPIInit().

C Prototype

```
int  PerfDatAPIAssocIsInit (char *sAppName);
```

Arguments

sAppName

API usage:	application database association handle
Type:	zero terminated character-coded text string
Access:	read-only
Mechanism:	by reference

The sAppName argument contains the reference to a zero terminated character-coded text string that contains the name of a particular application database association handle. The maximum length is 10 characters.

Description

This routine checks if the application database association specified by the sAppName argument exists and has been initialized by a previous call to PerfDatAPIInit(). The routine returns to the caller:

- PD$_SUCCESS
  Application database association specifiedby sAppName is initialized.
- PD$_NOINIT
  VSI PERFDAT API is not initialized.
- PD$_ASSOCNOTEXIST
  Application database association specified by sAppName does not exist.

The return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Condition values returned

PD$_SUCCESS	Database association specifiedby sAppName is initialized.
PD$_NOINIT	VSI PERFDAT API is not initialized.
PD$_ASSOCNOTEXIST	Application database association handle defined by sAppName does not exist.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.7 PerfDatAPIAssocRelAssoc

Releases an application database association previously created by calling the routine PerfDatAPIInit().

C Prototype

```
int  PerfDatAPIAssocRelAssoc (char *sAppName);
```

Arguments

sAppName

API usage:	application database association handle
Type:	zero terminated character-coded text string
Access:	read-only
Mechanism:	by reference

The sAppName argument contains the reference to a zero terminated character-coded text string that contains the name of a particular application database association handle. The maximum length is 10 characters.

Description

This routine releases the application database association previously created by calling the routine PerfDatAPIInit().

The sAppName argument specifies the name of the application database association to release.

Data collection processing has to be stopped in advance of calling this routine either by executing the STOP COLLECTION command of the PERFDAT_MGR utility or by calling the PerfDatAPIAssocStopColl() routine. Otherwise the routine fails.

Condition values returned

PD$_SUCCESS	The application database association has been successfully released.
PD$_COLLACT	The routine was unable to release the application database association because data collection processing is still in progress.
PD$_ASSOCNOTEXIST	Application database association handle defined by sAppName does not exist.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.8 *PerfDatAPIAssocStartColl*

Start data collection processing for a particular application database association.

C Prototype

```
int  PerfDatAPIAssocStartColl (char *sAppName , char *sProfileName);
```

Arguments

sAppName

API usage:	application database association handle
Type:	zero terminated character-coded text string
Access:	read-only
Mechanism:	by reference

The sAppName argument contains the reference to a zero terminated character-coded text string that contains the name of a particular application database association handle. The maximum length is 10 characters.

sProfileName

API usage:	collection profile name
Type:	zero terminated character-coded text string
Access:	read only
Mechanism:	by reference

Collection profile name used to start application data collection processing for the program.

The sProfileName argument contains the 32 bit address pointing to a zero terminated character-coded text string.

Description

This routine starts data collection processing for a particular application database association specified by the sAppName argument with the collection profile defined by the sProfileName argument.

If data collection processing is already active, the active data collection is not stopped and restarted with the collection profiled defined by the sProfileName argument. In this case PD$_STARTUP is returned.

If the VSI PERFDAT API is not initialized the routine returns PD$_NOINIT to the caller.

If the application database association specified by sAppName does not exist (PerfDatApiInit() has not been called for this application database association) PD$_ASSOCNOTEXIST is returned.

If the collection profile does not exist in the VSI PERFDAT configuration database for the application the program is associated with SS$_NOSUCHENTRY is returned.

If the sProfileName argument refers to a zero length string or the reference is invalid (i.e. NULL pointer) this routine searches for an entry in the auto-start table of the VSI PERFDAT configuration database valid for the node on which the program is running and the application the program is associated with. If no auto-start entry exist SS$_NOSUCHENTRY is returned to the caller.

If an auto-start entry exists the routine starts data collection processing with the collection profile defined by the auto-start entry.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Condition values returned

PD$_SUCCESS	Data collection processing has been successfully started.
PD$_NOINIT	VSI PERFDAT API is not initialized.
PD$_STARTUP	Data collection processing is already active.
PD$_ASSOCNOTEXIST	Application database association handle defined by sAppName does not exist.
SS$_NOSUCHENTRY	Collection profile name passed to the routine does not exist or no auto-start entry exists for the node the program is running and the application the program is associated with.

Any condition values returned by the DQL interface of VSI PERFDAT

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.9 PerfDatAPIAssocStopColl

Stop data collection processing for a particular application database association.

C Prototype

```
int  PerfDatAPIAssocStopColl (char *sAppName);
```

Arguments

sAppName

API usage:	application database association handle
Type:	zero terminated character-coded text string
Access:	read-only
Mechanism:	by reference

The sAppName argument contains the reference to a zero terminated character-coded text string that contains the name of a particular application database association handle. The maximum length is 10 characters.

Description

This routine stops data collection processing the collection database association specified by the sAppName argument and disables online alerting if it has been enabled. The collection database association remains initialized. Thus, data collection processing can be restarted either by calling the API routine PerfDatAPIAssocStartColl() or with the START COLLECTION command of the PERFDAT_MGR utility at any point in time after data collection processing has been stopped by calling this routine.

If the VSI PERFDAT API is not initialized the routine returns PD$_NOINIT to the caller.

If the application database association specified by sAppName does not exist (PerfDatApiInit() has not been called for this application database association) PD$_ASSOCNOTEXIST is returned.

If data collection processing is inactive PD$_SHUTDOWN is returned.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Condition values returned

PD$_SUCCESS	Data collection processing has been successfully stopped.
PD$_NOINIT	VSI PERFDAT API is not initialized.

PD$_SHUTDOWN	Data collection processing is inactive.
PD$_ASSOCNOTEXIST	Application database association handle defined by sAppName does not exist.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.10 PerfDatAPIDisableAlert

Disable online alerting for the first application database that has been associated with the calling program.

C Prototype

```
int  PerfDatAPIDisableAlert (void);
```

Arguments

None

Description

Disable online alerting for the first application database that has been associated with the calling program.

If the VSI PERFDAT API is not initialized the routine returns PD$_NOINIT to the caller. If online alerting is already disabled PD$_ALERTDIS is returned.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Note

This function should be called only if the calling program is associated with one application collection database. If the program is associated with more than one application database PerfDatAPIAssocDisableAlert()should be used.

Condition values returned

PD$_SUCCESS	Online alerting has been successfully disabled.
PD$_NOINIT	VSI PERFDAT API is not initialized.
PD$_ALERTDIS	Online alerting is already disabled.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.11 PerfDatAPIEnableAlert

Enable online alerting for the first application database that has been associated with the calling program.

C Prototype

```
int  PerfDatAPIEnableAlert (char *sAlertFileName);
```

Arguments

sAlertFileName

API usage:	alert definition file name
Type:	zero terminated character-coded text string
Access:	read only
Mechanism:	by reference

The sAlertFileName argument contains the reference to a zero terminated character-coded text string that specifies the alert definition file name used by the online alerting sub-system to check performance alert conditions.

Description

This routine enables online alerting for the first application database that has been associated with the calling program with the alert definition file specified by the sAlertFileName argument.

If the VSI PERFDAT API is not initialized the routine returns PD$_NOINIT to the caller. If data collection processing is inactive the routine returns PD$_SHUTDOWN.

If online alerting is already enabled, online alerting will be automatically disabled and re-enabled with the alert definition file specified by the sAlertFileName argument.

If the alert definition file does not exist SS$_NOSUCHENTRY is returned to the caller.

If the sAlertFileName argument refers to a zero length text string or the reference is invalid (i.e. NULL pointer) this routine searches for an entry in the auto-start table of the VSI PERFDAT configuration database valid for the node on which the program is running and the application the program is associated with. If no auto-start entry exists SS$_NOSUCHENTRY is returned to the caller.

If an auto-start entry exists the routine checks whether an alert definition file is defined in the auto-start entry. If neither an alert definition file has been defined nor the alert definition file defined in the auto-start entry exists SS$_NOSUCHENTRY is returned. Otherwise online alerting is enabled with the alert definition file defined in the auto-start entry.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Note

This function should be called only if the calling program is associated with one application collection database. If the program is associated with more than one application database PerfDatAPIAssocEnableAlert()should be used.

Condition values returned

PD$_SUCCESS	Online alerting has been successfully enabled
PD$_NOINIT	VSI PERFDAT API is not initialized.
PD$_SHUTDOWN	Data collection processing is inactive.
SS$_NOSUCHENTRY	Alert definition file does not exist.

Any condition values returned by the DQL interface of VSI PERFDAT

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.12 PerfDatAPIInit

This routine initializes the VSI PERFDAT API and requests the VSI PERFDAT DQL interface to associate a collection database with the calling program.

C Prototype

```
int  PerfDatAPIInit (unsigned int iEf, void (*pCollectAst) (),
                 unsigned __int64 qAstPara, void (*pExceptionAst) (),
                 char *sAppName, char *sVersion);
```

Arguments

iEf

API usage:	ef_number
Type:	unsigned integer
Access:	read only
Mechanism:	by value

Event flag that the VSI PERFDAT API sets at the end of each collection sample interval to notify the calling program to collect, to calculate and to insert the data records into the metrics (tables) of the associated application collection database.

The iEf argument is an unsigned integer value containing the number of the event flag.

If the iEf argument is zero the VSI PERFDAT API executes the collection AST routine defined by the pCollectAst argument at the end of each data collection interval. If both arguments are not defined this routine fails.

pCollectAst

API usage:	AST procedure
Type:	procedure value
Access:	call without stack unwinding
Mechanism:	by 32-bit reference

User defined collection AST routine to be called by the VSI PERFDAT API at the end of each data collection interval. This argument contains the address of the user defined collection AST routine.

If the iEf argument contains a value greater than zero the pCollectAst argument is ignored. If both arguments are not defined the routine fails.

qAstPara

API usage:	user argument

Type:	unsigned quad word
Access:	read only
Mechanism:	by 64-bit value

AST parameter to be passed to the user defined collection AST routine specified by the pCollectAst argument. The AST parameter is an unsigned quad word value.

pExceptionAst

API usage:	AST procedure
Type:	procedure value
Access:	call without stack unwinding
Mechanism:	by 32-bit reference

This argument defines a user defined exception handling routine. This routine is called whenever:

1. a run-time error occurs in an API routine
2. data collection processing has been started or stopped via the PERFDAT_MGR utility
3. online alerting has been enabled or disabled via the PERFDAT_MGR utility.

sAppName

API usage:	application database association name
Type:	zero terminated character-coded text string
Access:	read-only
Mechanism:	by reference

The sAppName argument contains the reference to a zero terminated character-coded text string that contains the application database association name. The maximum length of the application name is 10 characters.

This routine requests the DQL interface of VSI PERFDAT to associate a collection database with the calling program. The handle of this association is the name defined by the sAppName argument.

sVersion

API usage:	application version string
Type:	zero terminated character-coded text string
Access:	read-only
Mechanism:	by reference

The sVersion argument contains the reference to a zero terminated character-coded text string. The maximum length of the text string addressed by this argument is 12 characters.

The sVersion argument is optional. This argument can be used to pass the version (i.e. “V1.1”) of the application defined by the sAppName argument to the VSI PERFDAT API. The version string defined by this argument is inserted into the header of the application database the program is associated with. If this argument is omitted either if the NULL pointer is assigned to the argument or the length of the text string addressed by the sVersion argument is zero, the version string “V?.?” is inserted into the application database header.

Description

This routine initializes the VSI PERFDAT API and requests the VSI PERFDAT DQL interface to associate a collection database with the calling program. The handle of this association is the application database association name defined by the sAppName argument. The sAppName contains the reference to a zero terminated character-coded text string that contains the application database association name. The maximum length of the application name is 10 characters.

Due to the design of the VSI PERFDAT API a program can be associated with up to 16 application collection databases. Each application database association has to be explicitly initialized by calling PerfDatAPIInit().

The DQL interface checks if an application collection database descriptor with the same name as specified by the sAppName argument exists in the descriptor table of the VSI PERFDAT configuration database. An application collection database descriptor contains the record definitions for all metrics of an application collection database. Such an application collection database descriptor is required by the VSI PERFDAT API in order to create or access an application collection database (for detailed information about application databases and collection database descriptors please refer to the sections 1.4 Application collection database and 1.5.1 Collection database descriptor in this manual). If this check fails the API initialization fails.

If the application collection database descriptor defined by the sAppName argument exists, this routine registers the collection notification method passed to the routine. The VSI PERFDAT API triggers the calling program to collect, to calculate and to insert the data records into the metrics (tables) of the associated application collection database at the end of each sample interval. The sample interval is defined by the collection profile used to start an application data collection.

If an event flag number greater than zero is defined by the iEf argument this event flag is set at the end of each sample interval. If the iEf argument is greater than zero the pCollectAst and qAstPara arguments are ignored. If the iEf argument is zero the VSI PERFDAT API executes the collection AST routine defined by the pCollectAst at the end of each sample interval. The VSI PERFDAT API passes the AST parameter defined by the qAstPara argument to the pCollectAst routine.

Using the AST notification method requires no additional main loop coding compared with the event flag notification method (the calling program has to wait for the event flag to be set by the VSI PERFDAT API in the main loop). The disadvantage of the AST notification method is that the execution of the data collection and data insert processing routine is not under the control of the calling program since the AST routine is directly called by the VSI PERFDAT API at the end of each sample interval.

After the event notification method has been registered the PerfDatAPIInit()routine checks if an entry exists in the auto-start table of the VSI PERFDAT configuration database for the application specified by the application name parameter and the node the program was started on.

If no such auto-start entry exists the routine immediately returns to the caller. Otherwise the data collection defined in the auto-start entry will automatically be started. This means that the VSI PERFDAT API creates or, if the collection database already exists, attaches the associated collection database and starts notifying the calling program to collect its data at the end of each sample interval as specified by the collection profile defined in the auto-start entry. If online alerting is enabled in the auto-start entry and the defined online definition file exists online alerting is automatically enabled when this routine starts the application data collection.

The pExceptionAst argument specifies a user-defined exception handling routine. This routine is called whenever:

- a run-time error occurs in an API routine
- data collection processing has been started or stopped via the PERFDAT_MGR utility
- online alerting has been enabled or disabled via the PERFDAT_MGR utility.

Whenever the VSI PERFDAT API executes the user defined AST routine defined by the pExceptionAst argument a 32-bit reference to an API error block structure is passed to this AST routine. The type definition of the API error block structure is shown below:

```
typedef struct perfdat$error_block
{
        int     iStatus;
        int     iAPICodeLine;
        char    sAPIRoutine[32];
        char    sVmsErrCode[128];
} tPerfDatAPIErrBlk;
```

iStatus status code value field
If a VSI PERFDAT API routine has failed the status code field contains the status code returned by the failing routine. Otherwise this field contains the status codes listed below:

- PD$_STARTUP
  Data collection processing has been started with the START COLLECTION command of the PERFDAT_MGR utility.
- PD$_SHUTDOWN
  Data collection processing has been stopped with the STOP COLLECTION command of the PERFDAT_MGR utility.
- PD$_ALERTENB
  Online alerting has been enabled with the ENABLE ALERT command of the PERFDAT_MGR utility.
- PD$_ALERTDIS
  Online alerting has been disabled with the ENABLE ALERT command of the PERFDAT_MGR utility.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

iAPICodeLine API source code line of the failing instruction
If a VSI PERFDAT API routine has failed this field contains the VSI PERFDAT API source code line of the failing instruction. If the exception handling AST routine is called due to the execution of aPERFDAT_MGR command (start/stop data collection, enable/disable online alerting) this field always contains the value 0.

sAPIRoutine failed API procedure name
If a VSI PERFDAT API routine has failed this field contains the VSI PERFDAT API procedure name of the failing instruction. If the exception handling AST routine is called due to the execution of a PERFDAT_MGR command (start/stop data collection, enable/disable online alerting) this field contains a zero length text string.

sVmsErrCode textual description of the status code value
If a VSI PERFDAT API routine has failed this field contains the textual description of the status code value in iStatus. If the exception handling AST routine is called due to the execution of a PERFDAT_MGR command (start/stop data collection, enable/disable online alerting) this field contains a zero length text string.

If a VSI PERFDAT API routine has failed the API guarantees that data collection processing and on line alerting has been stopped before the user-defined exception handling routines is executed. Thus, whenever the exception AST routine is called due to an API routine failure the programmer does not have to care about stopping the collection processing, disabling online alerting or to release any data structures allocated for the application data collection or online alerting. The application database association is not released.

ThepExceptionAst argument is optional. If the calling program provides no exception handling AST routine assign the NULL pointer to this argument.

The sVersion argument is optional. This argument can be used to pass the version (i.e. “V1.1”) of the application defined by the sAppName argument to the VSI PERFDAT API. The version string defined by this argument is inserted into the header of the application database the program is associated with. If this argument is omitted either if the NULL pointer is assigned to the argument or the length of the text string addressed by the sVersion argument is zero, the version string “V?.?” is inserted into the application database header.

Condition values returned

PD$_SUCCESS	The VSI PERFDAT API has been successfully initialized.
PD$_INITIALIZED	The VSI PERFDAT API is already initialized.
PD$_STARTUP	The VSI PERFDAT API is already initialized and data collection processing is in progress.
SS$_AUTHFAIL	The user is not authorized to call the VSI PERFDAT API routine. Either the SYSLCK privilege is missing or the PERFDAT_API identifier has not been granted to the user (see 1.8 User privileges).
SS$_BADPARAM	SS$_BADPARAM is returned either: • Neither a valid event flag (iEf argument) nora valid user defined collection AST routine (pCollectAST argument) has been passed to the routine. • An invalid application name (sAppName argument) has been defined. The application name is invalid if: ○ No application collection database descriptor with the same name exists in the VSI PERFDAT configuration database. ○ The reference to the zero terminated character- coded text string is invalid ○ The length of the character-coded text string is 0 or greater than 10.

Any condition values returned by the routine PerfDatAPIStartColl(), PerfDatAPIAssocStartColl()and the VSI PERFDAT DQL interface.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Programming example

The VSI PERFDAT installation procedure provides two program examples that illustrate the use of this routine:

- PERFDAT$EXAMPLES:PERFDAT_API_TEST_EF.C
- PERFDAT$EXAMPLES:PERFDAT_API_TEST_AST.C

2.13 *PerfDatAPIInsertRecord*

This routine inserts a data record into a particular metric of the first application database that has been associated with the calling program.

C Prototype

```
int  PerfDatAPIInserRecord (char * sMetrix, tPerfDatAPIDataDsc *prData);
```

Arguments

sMetrix

API usage:	metric name of the associated collection database
Type:	zero terminated character-coded text string
Access:	read-only
Mechanism:	by reference

This argument defines the metric (table) of the associated application collection database to insert the data record addressed by the prData argument.

The sMetrix argument contains the 32-bit address pointing to a zero terminated character-coded text string.

prData

API usage:	data to insert
Type:	API data descriptor
Access:	read-only
Mechanism:	by 32-bit API data descriptor reference

This argument contains the 32-bit address pointing to an API data descriptor. An API data descriptor addresses the buffer that contains the data record to be inserted and the length of the data record.

Description

This routine inserts a data record into a particular metric of the first application database that has been associated with the calling program.

The sMetrix argument defines the metric (table) of the associated collection database to insert the data record addressed by the data record descriptorprData. If no such metric exists in the associated collection database the routine fails.

The data record to be inserted into the metric defined by the sMetrix argument has to be passed to this routine by use of an API data descriptor. An API data descriptor (see the data type definition below) contains the address to a buffer containing the data record to be inserted into the metric and the length of the data record.

```
typedef struct perfdat$data_dsc
{
            long        dsc$l_length;
            void        *dsc$v_pointer;
} tPerfDatAPIDataDsc;
```

dsc$l_length	length of the data record to insert.
dsc$v_pointer	void pointer to the buffer containing the data record.

The data fields of the data record addressed by the dsc$v_pointer field of the API data descriptor have to be ordered according to the record definition of the metric specified by the sMetrix argument. The record definitions of all metrics of an application collection database are defined by a collection database descriptor stored in the VSI PERFDAT configuration database. A collection database descriptor of an application database can be defined by loading a descriptor load file. For more information about collection database descriptors and how to create a descriptor load file and how to define metric record descriptor please refer to the section 1.5.1 Collection database descriptor.

Any metric record descriptor has to contain one time data field (see 1.5.1 Collection database descriptor). The collection time is automatically inserted into the time field of the data record passed to this routine before it is inserted into the metric defined.

Note

This function should be called only if the calling program is associated with one application collection database. If the program is associated with more than one application database PerfDatAPIAssocInsertRecord()should be used.

Condition values returned

PD$_SUCCESS	Data record has been successfully inserted into the metric defined by the sMetrix argument.
PD$_NOTINCOLL	Metric defined by the sMetrix argument is not enabled (has been disabled by the collection profile used to start the data collection) for the application data collection.
SS$_INVARG	SS$_INVARG is returned to the caller if: • The metric defined by the sMetrix argument does not exist in the application database associated with the program. • The data record passed to the routine does not match the data record definition of the metric defined by the sMetrix argument.

RMS$_DUP	Data record with the same primary key index already exist in the metric defined by the sMetrix argument.

Any condition values returned by RMS.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Programming example

The VSI PERFDAT installation procedure provides two program examples that illustrate the use of this routine:

- PERFDAT$EXAMPLES:PERFDAT_API_TEST_EF.C
- PERFDAT$EXAMPLES:PERFDAT_API_TEST_AST.C

2.14 PerfDatAPIIsAlertEnabled

Tests if online alerting is enabled for the first application database that has been associated with the calling program.

C Prototype

```
int  PerfDatAPIIsAlertEnabled (char *sAlertFileName, int iLen);
```

Arguments

sAlertFileName

API usage:	alert file name buffer
Type:	character-coded text string
Access:	write only
Mechanism:	by reference

The sAlertFileName argument contains the 32-bit address of a user defined string buffer.

If online alerting is enabled for the application data collection the program is associated with, this routine copies the alert definition file name used by the online alerting sub-system into the string buffer as a null terminated character-coded text string.

iLen

API usage:	length of the alert file name buffer
Type:	integer
Access:	read only
Mechanism:	by value

Length of the user defined string buffer addressed by the sAlertFileName argument.

Description

This routine checks if online alerting is enabled for the first application database that has been associated with the calling program

If online alerting is enabled for the application data collection the program is associated with, this routine copies the alert definition file name used by the online alerting sub-system into the string buffer addressed by the sAlertFileName argument as a zero terminated character-coded text string. If the size of the user define string buffer is less than the string length of the alert definition file name SS$_BADPARAM is returned.

If the VSI PERFDAT API is not initialized the routine returns PD$_NOINIT to the caller.

If online alerting is disabled or data collection processing is inactive SS$_NOSUCHENTRY is returned.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Note

This function should be called only if the calling program is associated with one application collection database. If the program is associated with more than one application database PerfDatAPIAssocIsAlertEnabled()should be used.

Condition values returned

PD$_SUCCESS	Online alerting is enabled and the alert definition file used by the online alerting sub-system has been successfully copied into the user defined string buffer.
PD$_NOINIT	VSI PERFDAT API is not initialized.
SS$_BADPARAM	Online alerting is enabled but the alert definition file used by the online alerting sub-system cannot be copied into the user defined string buffer because the string buffer is too small.
SS$_NOSUCHENTRY	Data collection processing is inactive.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Programming example

The VSI PERFDAT installation procedure provides two program examples that illustrate the use of this routine:

- PERFDAT$EXAMPLES:PERFDAT_API_TEST_EF.C
- PERFDAT$EXAMPLES:PERFDAT_API_TEST_AST.C

2.15 PerfDatAPIIsCollStarted

Tests if data collection processing is active for the first application database that has been associated with the calling program.

C Prototype

```
int  PerfDatAPIIsCollStarted (char *sProfileName, int iLen);
```

Arguments

sProfileName

API usage:	collection profile name buffer
Type:	character-coded text string
Access:	write only
Mechanism:	by reference

The sProfileName argument contains the 32-bit address of a user defined string buffer.

Ifdata collection processing is currently active this routines copies the collection profile name used to start the application data collection into the string buffer as a zero terminated character-coded text string. The maximum length of a collection profile name is 48 characters. Thus, the length of the string buffer addressed by the sProfileName argument should be at least 48 characters.

iLen

API usage:	length of the collection profile name buffer
Type:	integer
Access:	read only
Mechanism:	by value

Length of the user defined string buffer addressed by the sProfileName argument.

Description

This routine checks if data collection processing is active for the first application database that has been associated with the calling program.

If data collection processing is currently active this routines copies the collection profile name used to start the application data collection into the string buffer addressed by the sProfileName argument as a zero terminated character-coded text string. The maximum length of a collection profile name is 48 characters. Thus, the length of user defined string buffer should be greater than 48 characters. If the size of the user define string buffer is less than the string length of the collection profile name SS$_BADPARAM is returned.

If the VSI PERFDAT API is not initialized the routine returns PD$_NOINIT to the caller.

If data collection processing is inactive SS$_NOSUCHENTRY is returned.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Note

This function should be called only if the calling program is associated with one application collection database. If the program is associated with more than one application database PerfDatAPIAssocIsCollStarted()should be used.

Condition values returned

PD$_SUCCESS	Data collection processing is active and the collection profile used to start the application data collection has been successfully copied into the user defined string buffer.
PD$_NOINIT	VSI PERFDAT API is not initialized.
SS$_BADPARAM	Data collection processing is active but the collection profile name used to start the application data collection cannot not be copied into the user defined string buffer because the size of the string buffer is too small.
SS$_NOSUCHENTRY	Data collection processing is inactive.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Programming example

The VSI PERFDAT installation procedure provides two program examples that illustrate the use of this routine:

- PERFDAT$EXAMPLES:PERFDAT_API_TEST_EF.C
- PERFDAT$EXAMPLES:PERFDAT_API_TEST_AST.C

2.16 *PerfDatAPIIsInit*

Tests if the VSI PERFDAT API is already initialized and if at least one application database is associated with the calling program.

C Prototype

```
int  PerfDatAPIIsInit (void);
```

Arguments

None

Description

The PerfDatAPIIsInit() routine tests if the VSI PERFDAT API is already initialized and if at least one application database is associated with the calling program. The routine returns to the caller:

• PD$_SUCCESS	VSI PERFDAT API is initialized.
• PD$_NOINIT	VSI PERFDAT API is not initialized.

The return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Note

This function should be called only if the calling program is associated with one application collection database. If the program is associated with more than one application database PerfDatAPIAssocIsInit()should be used.

Condition values returned

PD$_SUCCESS	VSI PERFDAT API is initialized
PD$_NOINIT	VSI PERFDAT API is not initialized

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.17 PerfDatAPIRelAssoc

Releases the first application database association created by calling the routine PerfDatAPIInit().

C Prototype

```
int  PerfDatAPIRelAssoc (void);
```

Arguments

None

Description

This routine releases the first application database association created by calling the routine PerfDatAPIInit().

Data collection processing has to be stopped in advance of calling this routine either by executing the STOP COLLECTION command of the PERFDAT_MGR utility or by calling the PerfDatAPIStopColl() routine. Otherwise the routine fails.

Note

This function should be called only if the calling program is associated with one application collection database. If the program is associated with more than one application database PerfDatAPIAssocRelAssoc() should be used.

Condition values returned

PD$_SUCCESS	The application database association has been successfully released.
PD$_COLLACT	The routine was unable to release the application database association because data collection processing is still in progress.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.18 PerfDatAPIStartColl

Start data colllection processing for the first application database that has been associated with the calling program.

C Prototype

```
int  PerfDatAPIStatup (char *sProfileName);
```

Arguments

sProfileName

API usage:	collection profile name
Type:	zero terminated character-coded text string
Access:	read only
Mechanism:	by reference

Collection profile name used to start application data collection processing for the program.

The sProfileName argument contains the 32 bit address pointing to a zero terminated character-coded text string.

Description

This routine starts data collection processing for the first application database that has been associated with the calling program with the collection profile defined by the sProfileName argument.

If data collection processing is already active, the active data collection is not stopped and restarted with the collection profiled defined by the sProfileName argument. In this case PD$_STARTUP is returned.

If the VSI PERFDAT API is not initialized the routine returns PD$_NOINIT to the caller.

If the collection profile does not exist in the VSI PERFDAT configuration database for the application the program is associated with SS$_NOSUCHENTRY is returned (the application association is defined when the VSI PERFDAT API is initialized – see the description of the routine PerfDatAPIInit()).

If the sProfileName argument refers to a zero length string or the reference is invalid (i.e. NULL pointer) this routine searches for an entry in the auto-start table of the VSI PERFDAT configuration database valid for the node on which the program is running and the application the program is associated with. If no auto-start entry exist SS$_NOSUCHENTRY is returned to the caller.

If an auto-start entry exists the routine starts data collection processing with the collection profile defined by the auto-start entry.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Note

This function should be called only if the calling program is associated with one application collection database. If the program is associated with more than one application database PerfDatAPIAssocStartColl()should be used.

Condition values returned

PD$_SUCCESS	Data collection processing has been successfully started.
PD$_NOINIT	VSI PERFDAT API is not initialized.
PD$_STARTUP	Data collection processing is already active.
SS$_NOSUCHENTRY	Collection profile name passed to the routine does not exist or no auto-start entry exists for the node the program is running and the application the program is associated with.

Any condition values returned by the DQL interface of VSI PERFDAT

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

2.19 *PerfDatAPIStopColl*

Stop data collection processing for the first application database that has been associated with the calling program.

C Prototype

```
int  PerfDatAPIStopColl (void);
```

Arguments

None

Description

This routine stops data collection processing for the first application database that has been associated with the calling program and disables online alerting if it has been enabled. The VSI PERFDAT API remains initialized (application database association is still valid). Thus, data collection processing can be restarted either by calling the API routine PerfDatAPIStartColl() or with the START COLLECTION command of the PERFDAT_MGR utility at any point in time after data collection processing has been stopped by calling this routine.

If the VSI PERFDAT API is not initialized the routine returns PD$_NOINIT to the caller.

If data collection processing is inactive PD$_SHUTDOWN is returned.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Note

This function should be called only if the calling program is associated with one application collection database. If the program is associated with more than one application database PerfDatAPIAssocStopColl()should be used.

Condition values returned

PD$_SUCCESS	Data collection processing has been successfully stopped.
PD$_NOINIT	VSI PERFDAT API is not initialized.
PD$_SHUTDOWN	Data collection processing is inactive.

The PD$ return codes are defined in PERFDAT$INCLUDE:PERFDAT_API.H.

Program Examples

The VSI PERFDAT installation procedure provides two C example programs that illustrate the use of the VSI PERFDAT API:

- PERFDAT$EXAMPLES:PERFDAT_API_TEST_EF.C
- PERFDAT$EXAMPLES:PERFDAT_API_TEST_AST.C

Basically both programs perform the same tasks. They collect:

- the direct I/O rate
- the buffered I/O rate
- the private page count
- global page count

of the process in whose context the example program is running in.

The main difference between these two programs is how they initialize the VSI PERFDAT API. In PERFDAT_API_TEST_EF.C the event flag notification method is used to trigger a data collection. In PERFDAT_API_TEST_AST.C the VSI PERFDAT C API directly calls a user defined AST routine to collect and insert the data.

3.1 PERFDAT_API_TEST_EF.C

In the main routine of the program the API initialization routine is called to associate an application collection database with the application name the user provides in P1 when he starts the program. The application name is copied into the string variable sApplication.

```
iStatus = lib$get_ef (&iEF);
…
iStatus = PerfDatAPIInit (iEf, NULL, 0, APIException, sApplication, NULL);
…
```

In this program a valid event flag number is passed to the initialization routine. Thus, the API sets this event flag at the end of each sample interval to trigger data collection and data insert processing of the program.

Since the APIException exception handling routine is defined, the VSI PERFDAT API will notify the calling program asynchronously whenever:

- a run-time error occurs in an API routines
- data collection processing has been started or stopped via the PERFDAT_MGR utility
- online alerting has been enabled or disabled via the PERFDAT_MGR utility.

After calling the VSI PERFDAT API initialization routine PerfDatAPIInit() the program checks whether a data collection has been automatically started and online alerting has been enabled by calling the API routines

```
iStatus = PerfDatAPIIsCollStarted(sProf, sizeof (sProf));
…
iStatus = PerfDatAPIIsAlertEnabled (sAlertFile, sizeof (sAlertFile));
```

Since the API does not directly call a particular user defined collection AST routine to perform the data collection processing, the program waits in the main loop for the event flag to be set by the VSI PERFDAT API and calls the routine PerfdatAPIExampleCollect() which performs the data collection.

```
for (;;)
{
        iStatus = sys$waitfr (iEf);
        if (!ODD (iStatus)) exit (iStatus);
        iStatus = sys$clref (iEf);
        if (!ODD (iStatus)) exit (iStatus);

        iStatus = PerfdatAPIExampleCollect (&rNew, &rOld);
        if (!ODD (iStatus)) exit (iStatus);
}
```

In the routine PerfdatAPIExampleCollect() the data is collected for all metrics and the data records are inserted into the metric PRCIO and PRCMEM by calling the API routine PerfDatAPIInsertRecord().

```
static char         sIOMetrix[] = “PRCIO”;
static char         sIOMetrix[] = “PRCMEM”;
…
static int PerfdatAPIExampleCollect (tCollection *prNew, tCollection *prOld)
{
…
tPerfDataAPIDataDsc        rData;
…

        /* insert data into PERFDAT tables */
        /* 1. Metrix: PRCIO */
        rData.dsc$l_length = sizeof (tProcIO);
        rData.dsc$v_pointer = (void*) &rIO;
        iStatus = PerfDatAPIInsertRecord (sIOMetrix, &rData);
        …
        /* 2. Metrix: PRCMEM */
        rData.dsc$l_length = sizeof (tProcMem);
        rData.dsc$v_pointer = (void*) &prNew->rMem;
        iStatus = PerfDatAPIInsertRecord (sMEMMetrix, &rData);
        …
}
```

3.2 PERFDAT_API_TEST_AST.C

In this code example the VSI PERFDAT API is initialized to call a user defined collection AST routine at the end of each collection interval which contains the code for data collection and data insert processing of the program.

```
iStatus = PerfDatAPIInit (0, APICollect, 0, APIException, sApplication, NULL);
```

The user define AST routineAPICollect()calls PerfdatAPIExampleCollect(), the same routine used in PERFDAT_API_TEST_EF.C for data collection and data insert processing.

There are no other code differences between PERFDAT_API_TEST_EF.C and PERFDAT_API_TEST_AST.C.

Using the AST notification method requires no additional main loop coding compared with the event flag notification method. The disadvantage of the AST notification method is that the execution of the data collection and data insert processing routine is not under the control of the calling program since the AST routine is called directly from the VSI PERFDAT API at the end of each sample interval.

3.3 Build instructions

To build the example programs on Alpha:

```
$ CC/FLOAT=G_FLOAT PERFDAT_API_TEST_EF
$ CC/FLOAT=G_FLOAT PERFDAT_API_TEST_AST
$ LINK PERFDAT_API_TEST_EF,PERFDAT$LIBRARY:PERFDAT_API_AXP/LIB
$ LINK PERFDAT_API_TEST_AST,PERFDAT$LIBRARY:PERFDAT_API_AXP/LIB
```

To build the example programs on IA64:

```
$ CC/FLOAT=G_FLOAT PERFDAT_API_TEST_EF
$ CC/FLOAT=G_FLOAT PERFDAT_API_TEST_AST
$ LINK PERFDAT_API_TEST_EF,PERFDAT$LIBRARY:PERFDAT_API_IA64/LIB
$ LINK PERFDAT_API_TEST_AST,PERFDAT$LIBRARY:PERFDAT_API_IA64/LIB
```

3.4 Configuration instructions

1. As described in previous sections the VSI PERFDAT API tries to associate an application database with the program based on the application name passed in the API initialization routine. An application database is associated with the calling program if an application database descriptor exists in the VSI PERFDAT configuration database with the same name as the application name passed in the API initialization routines. The VSI PERFDAT installation procedure provides the database descriptor load file PERFDAT$EXAMPLES:PERFDAT_API_TEST.CFG.

   Before you run either of the example programs load this database descriptor load file containing the application database descriptor for application TEST into the VSI PERFDAT configuration database.

   ```
   $ MCR PERFDAT_MGR LOAD METRIX
                       PERFDAT$EXAMPLES:PERFDAT_API_TEST.CFG
   ```

2. Create a collection profile for the application TEST:

   ```
   $ MCR PERFDAT_MGR ADD PROFILE DEFAULT/OS=TEST

          WELCOME to TEST collection profile wizard

      Collection sample interval [120 sec]:
      Enable metrix PRCIO [Yes]:
      Enable metrix PRCMEM [Yes]:
   PERFDAT_MGR-I-CFGSUCC, Profile /DEFAULT/ added for OS Type /TEST/
   ```

3. Check if the account you are logged into has the following privileges assigned:
   - NETMBX
   - TMPMBX
   - SYSLCK

4. Check if the PERFDAT_API identifier has been granted to the account you are logged into. If not, grant the identifier to the user:

   ```
   $ MCR AUTHORIZE GRANT/IDENT PERFDAT_API user-name
   ```

3.5 Running the Example Programs

1. Create foreign commands:

   ```
   $ PERFDAT_TEST_EF :== $PERFDA$EXAMPLES:PERFDAT_API_TEST_EF.EXE
   $ PERFDAT_TEST_AST:== $PERFDA$EXAMPLES:PERFDAT_API_TEST_AST.EXE
   ```

2. Start either of the programs using the foreign commands and pass the application name in P1 to the program that it will be associated with. Since the database descriptor load file PERFDAT$EXAMPLES: PERFDAT_API_TEST. CFG contains the application database descriptor for the application TEST, set the value of P1 to TEST.

   ```
   $ PERFDAT_TEST_EF TEST
   ```
   Or
   ```
   $ PERFDAT_TEST_AST TEST
   ```

3. If the example program display the output:

   ```
   No collection automatically started, no-auto-start entry exists for application
   TEST
   ```

   start the application collection with the collection database profile previously created (DEFAULT) using the PERFDAT_MGR utility:

   ```
   $ MCR PERFDAT_MGR START COLLECTION DEFAULT/OS=TEST
   ```