Oracle Trace Collector™

User's Guide

Version 2.2

Part No. A38159-1

ORACLE®

Oracle Trace Collector User's Guide

Version 2.2

Part No. A38159-1

Contents

3 Scheduling Data Collection

4 Instrumenting Applications

5 Creating Facility Definitions

6 Oracle Trace Commands

7 Oracle Trace Service Routines

8 System Management Tasks

9 Troubleshooting Oracle Trace

A Example of the ATM-Sample Facility Instrumented

B Registering a Cross-Facility Item

Glossary

Index

Examples

Figures

Tables

Send Us Your Comments

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

You can send comments to us in the following ways:

- electronic mail — nedc_doc@us.oracle.com
- FAX — 603-897-3334 Attn: Oracle Trace Documentation
- postal service

  Oracle Corporation
  Oracle Trace Documentation
  One Oracle Drive
  Nashua, NH 03062
  USA

If you like, you can use the following questionnaire to give us feedback. (Edit the online release notes file, extract a copy of this questionnaire, and send it to us.)

Name _____ Title _____

Company _____ Department _____

Mailing Address _____ Telephone Number _____

Book Title _____ Version Number _____

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?

- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available).

Preface

This manual describes how to use Oracle Trace for OpenVMS Collector Version 2.2 to collect event-based data and instrument applications for collection. The documentation refers to Oracle Trace for OpenVMS by its abbreviated name: Oracle Trace.

Intended Audience

This manual is intended for software performance analysts, application programmers, and database administrators. Users of the Oracle Trace software fall into two distinct groups: general users and application developers. The general users gather event-based data from products and applications that are already programmed to work with Oracle Trace. Developers will add Oracle Trace service routine calls to their programs and applications to collect the event-based data from them.

General users should read this manual in the order that it is presented (Chapters 1, 2, and 3).

Application developers who want to instrument their applications should read Chapter 1, then skip to Chapter 4 and 5 and then go back and read Chapters 2 and 3.

System managers should read Chapter 8.

Chapters 6, 7, and 9 provide reference material of interest to all users.

Structure

This manual has nine chapters, two appendixes, a glossary, and an index:

Chapter 1	Provides an overview of the Oracle Trace software and describes the sample application.
Chapter 2	Describes how to create facility selections for those who instrument application source code.

Chapter 3	Describes how to schedule data collection.
Chapter 4	Describes how to instrument your application source code with Oracle Trace service routines.
Chapter 5	Describes how to create facility definitions for applications you have instrumented.
Chapter 6	Describes the Oracle Trace commands.
Chapter 7	Describes the Oracle Trace service routines.
Chapter 8	Describes system management tasks associated with the Oracle Trace software.
Chapter 9	Describes solutions for common Oracle Trace problems.
Appendix A	Contains an example of a facility written in C that is instrumented for Oracle Trace.
Glossary	Defines terms used in this manual and other manuals in the documentation set.
Index	Provides page references for topics covered in this manual.

Related Documents

The other manuals in the Oracle Trace documentation set are:

- *Oracle Trace Installation Guide*—Provides instructions for installing the Oracle Trace software on an OpenVMS system.
- *Oracle Trace Release Notes*—Provides additional information about the Oracle Trace software that was not included in the *Oracle Trace Collector User's Guide*. The release notes are located in SYS$HELP:EPC022.RELEASE_NOTES.
- *Oracle Trace Getting Started*—Provides an introduction and overview to the Oracle Trace software.
- *Oracle Trace Reporter User's Guide*—Describes how to format and generate reports from event-based data.
- *Oracle Trace Monitor User's Guide*—Describes how to display Oracle Trace data interactively.

Conventions

The special symbols used in this book are:

Symbol	Meaning
CTRL/x	This symbol tells you to press the CTRL (control) key and hold it down while pressing the specified letter key.
Bold	Bold lettering in text indicates the definition of a new term. Refer to the Glossary for definitions of new terms.
[ ]	Brackets indicate optional elements.
. . .	Horizontal ellipsis indicates that you can enter additional parameters, values, or information.
. . .	Vertical ellipsis in an example means that information not directly related to the example has been omitted.
$	The dollar sign is used to indicate the DCL prompt. This prompt may be different on your system.

References to Products

In this guide Oracle Trace for OpenVMS is more simply referred to as Oracle Trace.

The naming conventions for other Oracle products discussed in this guide are as follows:

- Oracle CDD/Repository for OpenVMS VAX and Oracle CDD/Repository for OpenVMS Alpha software are referred to as Oracle CDD/Repository or the dictionary. (Previous to Version 5.0, Oracle CDD/Repository was called VAX CDD/Plus.)
- Oracle CODASYL DBMS refers to all products previously known as: DEC DBMS for OpenVMS VAX, DEC DBMS for OpenVMS AXP, and VAX DBMS.
- Oracle Rdb refers to all products previously known as: DEC Rdb for OpenVMS VAX, DEC Rdb for OpenVMS AXP, and VAX Rdb/VMS.

The naming conventions for Digital Equipment Corporation products discussed in this guide are as follows:

- DEC ACMS is referred to as ACMS.
- DEC DATATRIEVE is referred to as DATATRIEVE.

- VAX Language-Sensitive Editor software is referred to as LSE.
- Various Digital Equipment Corporation programming language compilers are referred to by their common names, without the VAX or DEC prefix.

1
Introduction to Oracle Trace

Oracle Trace for OpenVMS is a layered product that gathers and reports event-based data from any combination of OpenVMS layered products and application programs containing Oracle Trace service routine calls. The Oracle Trace documentation refers to application programs that contain Oracle Trace service routines as **facilities**. The following products are facilities:

- DEC ACMS
- DEC ALL–IN–1
- DECforms
- Oracle CODASYL DBMS
- Oracle RALLY
- Oracle Rdb

Oracle Trace provides Oracle Expert for Rdb with data that it uses for optimizing existing Oracle Rdb databases. It also provides data to ALL–IN–1 Performance Reports for OpenVMS.

You can collect event-based data from products that contain Oracle Trace service routine call. You can also add Oracle Trace service routines to your own applications to collect data from them. The process of adding Oracle Trace service routine calls to an application is called **instrumenting** an application. The products that are instrumented for Oracle Trace provide documentation that describes details of their instrumentation.

Oracle Trace software operates with minimal performance impact on the system. It can run with both the development and production versions of your application to give you information about the behavior of your application.

Several third-party, fourth-generation language (4GL) products, contain Oracle Trace service routine calls and work with Oracle Trace and RdbExpert. Contact your 4GL vendor for more information.

1.1 What Can Oracle Trace Do for You?

You can use the event data that Oracle Trace collects from applications for many different purposes including:

- Tuning the performance of applications
- Planning hardware resources (capacity planning)
- Tuning the performance of databases
- Debugging applications
- Logging errors

1.1.1 Database Tuning

For database tuning, Oracle Trace provides request and transactional-level information from Oracle Rdb. This information allows a database administrator to examine a wide variety of performance statistics and usage information related to actual transactions and DML (Database Manipulation Language) requests.

Oracle Trace provides RdbExpert the information that it uses to produce more efficient database designs. Database administrators no longer have to guess about the database workload because Oracle Trace collects actual workload information.

1.1.2 Transaction Processing

Oracle Trace tabular reports identify occurrences such as the transaction with the highest virtual memory usage, or the 95th-percentile disk I/O for each transaction. For transaction processing, Oracle Trace gathers information for DEC ACMS events to provide task-level performance information. For DECforms, Oracle Trace provides response time information.

1.1.3 Relating Events Among Facilities

To provide better information about the context in which some events are being executed, Oracle Trace now allows users to relate items from one application or layered product to items associated with events in other applications or layered products. The Oracle Trace cross-facility item capability lets instrumenters define relationships among events in two or more facilities.

Many products instrumented with Oracle Trace routines use the cross-facility capability. For example, the cross-facility capability associates the ACMS Procedure Call event with its related Oracle Rdb transactions and requests. This relationship provides a greater understanding of the total resources the ACMS Procedure Call uses. Based on your understanding of how facilities

relate events, you can produce Oracle Trace reports that display related events and their associated items.

1.2 What Is Event-Based Data Collection?

An event-based collector gathers data at predefined locations in your program code when that code is executed. Timer-based collectors perform data collection at specified time intervals, at random places within your code. Oracle Trace differs from other collectors in that it is event-based, whereas most other collectors are timer based. Event-based collectors are advantageous because they:

- Help you to determine the actual frequency of the execution of events, rather than an average or estimated frequency.
- Provide an easy way to collect and report on the resources used by specific events in applications

An **event** is an application-defined entity. There are two kinds of events: a **duration event** that has a start and an end, and a **point event** that can simply occur. With Oracle Trace you can define events within layered products or application programs and associate data items with each event. Data items fall into three general types:

- **Standard resource utilization statistics**—are data items that Oracle Trace collects for all facilities.
- **Facility-specific items**—are data items pertaining to an application that instrumenters must explicitly define before they are collected.
- **Cross-facility items** —are data items that relate events among facilities. See the documentation associated with the facilities from which you collect data to learn about the cross-facility items they use. Programmers can specify cross-facility items in the applications they instrument.

1.3 Description of the Sample Applications

Oracle Trace provides sample applications in EPC$EXAMPLES that are instrumented with Oracle Trace service routine calls. General users will find these applications helpful because the Oracle Trace documentation provides many command and report examples based on these applications. Application programmers who want to instrument their own applications with Oracle Trace service routine calls can reference these example applications as they instrument their own code. Table 1–2 describes these applications.

These applications simulate a simple bank automated teller machine (ATM) and define Duration events for each of the basic transactions: checking a balance, depositing funds, and withdrawing funds. They also define a point event to note execution of the error-handling procedure, for example, when an overdraft occurs.

The ATM application is instrumented to gather information about the user interface. The goal is to be able to answer questions such as:

- How long does it take to complete a transaction?
- Do customers check their balance before or after every transaction?
- Could overdrafts be reduced or eliminated by displaying the balance on the withdrawal display?
- Are ATM machines used primarily for deposits or withdrawals?

1.4 How to Use the Sample Applications

You can use the sample applications in many ways. You can examine the source code versions for examples of instrumenting details. You can also modify these versions to gain a fuller understanding of the instrumenting process. Oracle Trace also provides compiled versions of both the extended and the non-extended applications that you can run.

Perform the instructions in Table 1–1 to run a sample application.

Table 1–1 Running a Sample Application

Step	Action
1	Select the application that you want to run from Table 1–2 and note the files associated with it.

2	Copy the application and its associated files from EPC$EXAMPLES into the directory from which you wish to run it.
	Example for the EPC$ATM-SAMPLE-EXTENDED.COB application:
	`$COPY EPC$EXAMPLES:EPC$ATM-SAMPLE-EXTENDED.COB *.*`
	`$COPY EPC$EXAMPLES:EPC$ATM-SAMPLE.DAT *.*`
	`$COPY EPC$EXAMPLES:EPC$ATM-FAC-DEF-EXTENDED.COM *.*`

(continued on next page)

Table 1–1 (Cont.) Running a Sample Application

Step	Action

3	Compile and link the application. If you have selected an already compiled version (.exe) proceed to the next step.
	Example for the EPC$ATM-SAMPLE-EXTENDED.COB application:
	`$ COBOL EPC$ATM-SAMPLE-EXTENDED`
	`$ LINK EPC$ATM-SAMPLE-EXTENDED`
	Example for the EPC$ATM-SAMPLE-EXTENDED.C application:
	`$ CC EPC$EXAMPLES:EPC$ATM-SAMPLE-EXTENDED`
	`$ LINK EPC$ATM-SAMPLE-EXTENDED, -`
	`_$ EPC$EXAMPLES:EPC$ATM-SAMPLE-SQL,SYS$INPUT -`
	`_$ /options SYS$SHARE:VAXCRTL/SHARE`

4	Execute the command procedure to insert the facility definition. Example for the EPC$ATM-SAMPLE-EXTENDED.COB application: `$ @EPC$ATM-FAC-DEF-EXTENDED.COM`
5	Run the application. Example for the EPC$ATM-SAMPLE-EXTENDED.COB application: `$ RUN EPC$ATM-SAMPLE-EXTENDED.EXE`

See Table 1–4 and Table 1–5 for a overview of getting started using Oracle Trace.

Table 1–2 Sample Applications in EPC$EXAMPLES and Their Associated Files

Application	Data file[1]	Facility Definition Command Procedure
EPC$ATM-SAMPLE-EXTENDED.C[2]	None, it creates its own Oracle Rdb database	EPC$ATM-FAC-DEF-EXTENDED.COM
EPC$ATM-SAMPLE-EXTENDED.COB[3]	EPC$ATM-SAMPLE.DAT	EPC$ATM-FAC-DEF-EXTENDED.COM
EPC$ATM-SAMPLE.FOR	EPC$ATM-PASCAL-FORTRAN.DAT	EPC$ATM-FAC-DEF.COM

[1]Data files contain ten account records numbered from 1 to 10.

[2]All of the sample applications collect standard resource utilization items. The EPC$ATM-SAMPLE-EXTENDED.C application collects facility-specific and cross-facility items as well.

[3]All of the sample applications collect standard resource utilization items. The EPC$ATM-SAMPLE-EXTENDED.COB application collects facility-specific items as well.

(continued on next page)

Table 1–2 (Cont.) Sample Applications in EPC$EXAMPLES and Their Associated Files

Application	Data file[1]	Facility Definition Command Procedure
EPC$ATM-SAMPLE.PAS	EPC$ATM-PASCAL-FORTRAN.DAT	EPC$ATM-FAC-DEF.COM
EPC$ATM-SAMPLE.COB	EPC$ATM-SAMPLE.DAT	EPC$ATM-FAC-DEF.COM
EPC$ATM-SAMPLE.EXE	EPC$ATM-C.DAT	EPC$ATM-FAC-DEF.COM

[1]Data files contain ten account records numbered from 1 to 10.

1.5 Overview of Oracle Trace Commands

Table 1–3 provides an overview of Oracle Trace capabilities and their associated commands. Chapter 6 describes these commands in more detail.

Table 1–3 Overview of Oracle Trace Commands

Collecting and Reporting Event-Based Data	
Task	**Command**
Execute a command procedure.	@
Exit Oracle Trace and return to the DCL command level.	EXIT
Create a facility selection to limit the scope of a collection.	CREATE SELECTION
Delete a facility selection.	DELETE SELECTION
Schedule a collection.	SCHEDULE COLLECTION
Cancel a collection.	CANCEL COLLECTION
Create an Oracle Rdb database from one or more data collection files.	FORMAT[1]
Produce a hard-copy Oracle Trace report from an Oracle Rdb database.	REPORT[1]
Tracking Oracle Trace Collections	
Task	**Command**

[1]See the *Oracle Trace Reporter User's Guide* for more information about this command.

(continued on next page)

Table 1–3 (Cont.) Overview of Oracle Trace Commands

Tracking Oracle Trace Collections	
Display the processes on your system from which you can collect data.	SHOW REGISTER
Display facility definition information from the Oracle Trace administration database.	SHOW DEFINITION
Display facility selection information from the administration database.	SHOW SELECTION
Display error and informational messages that have occurred during a collection.	SHOW HISTORY
Display the version number of Oracle Trace installed on your system.	SHOW VERSION
Create a subprocess of a current process.	SPAWN

Managing Oracle Trace	
Task	**Command**
Grant or deny users access to either the Oracle Trace administration or history database.	SET ACCESS
Display a report of users who have access to the Oracle Trace administration, history, and formatted databases.	SHOW ACCESS
Create a new history database or change to another history database. The history database contains error and informational messages from Oracle Trace collections.	SET HISTORY
Start the Oracle Trace Registrar process as part of your normal system startup procedure, *after* activating the Oracle Rdb monitor process.	Add @EPC$STARTUP to the system startup command file on each node that you want Oracle Trace to run, as described in Section 8.2.1.
Insert a facility definition from a binary file into the Oracle Trace administration database.	INSERT DEFINITION
Stop Oracle Trace and interrupt any active data collection.	STOP SYSTEM
Insert a facility definition provided by Oracle Trace.	@EPC$EXAMPLES:EPC$INSERT
Remove Oracle Trace from your system.	@SYS$UPDATE:EPC$DEINSTAL.COM

(continued on next page)

Table 1–3 (Cont.) Overview of Oracle Trace Commands

Instrumenting an Application[2]	
Task	**Command**
Specify the events and items in an application for which to collect data.	Oracle Trace service routines[3]
Creates a facility definition for your application.	CREATE DEFINITION
Extract a definition from the Oracle Trace administration database and store it in a binary file.	EXTRACT DEFINITION
Delete a facility definition from the Oracle Trace administration database.	DELETE DEFINITION

[2]Many products are already instrumented for Oracle Trace collections. Therefore, you do not have to do any instrumenting to use Oracle Trace with these products. However, Oracle Trace provides commands and routines for programmers who want to instrument their own applications for Oracle Trace collections.

[3]See the Chapter 7 for more information about the Oracle Trace service routines.

1.6 How to Use Oracle Trace

Table 1–4 summarizes the task you would perform to prepare to collect and report on event-based data using Oracle Trace. See the *Oracle Trace Getting Started* manual for more details about each of these steps.

Table 1–4 Summary of Preparation Tasks

Step	Action
1	Learn if Oracle Trace EPC$REGISTRAR process is running on your system: `$ SHOW SYSTEM`

2	Determine the version of the facility that your process runs. For example, to learn the Oracle Rdb version:
	`$ RMU/SHOW VERSION`
	`Executing RMU for Rdb/VMS V4.0-0`

(continued on next page)

Table 1–4 (Cont.) Summary of Preparation Tasks

Step	Action
3	Verify the presence of the appropriate facility definition in the administration database. For example to verify Oracle Rdb: `$ COLLECT SHOW DEFINITION RDBVMS` If the administration database does not contain the appropriate facility and version, in this case for example, Oracle Rdb Version 4.0, ask your system manager to run the EPC$INSERT command procedure in EPC$EXAMPLES to add the Oracle Rdb version you are running to the administration database.

Table 1–5 describes tasks for collecting and reporting on events using Oracle Trace, after you complete the preparation tasks. It also describes the functions Oracle Trace performs in response to the commands that you enter.

Table 1–5 Summary of Tasks to Collect Data from Facilities

Step	Action

1	Create a facility selection as described in Chapter 2. For example:
	`$ COLLECT CREATE SELECTION SAMPLE_SELECTION /OPTIONS`
	`Options> FACILITY RDBVMS/VERSION="V4.0-0"`
	`Options>` CTRL/Z
	Specify the facility and version number from Step 2, Table 1–4.

2	Schedule the collection, as described in Chapter 3. For example:
	`$ COLLECT SCHEDULE COLLECTION SAMPLE_COLLECTION SAMPLE_DATA.DAT-`
	`_$ /SELECTION=SAMPLE_SELECTION -`
	`_$ /DURATION=:30-`
	`_$ /NOCLUSTER /COLLECTION_FILES=(PROTECTION=(W:RW))`

3	Run your application. Oracle Trace performs set-up tasks, as described in Table 1–6. The facility writes event-based data to a data collection file.
4	Wait for collections to finish, or cancel them: `$ COLLECT CANCEL COLLECTION SAMPLE_COLLECTION /NOCONFIRM` Oracle Trace stops collecting data and closes the data collection file.
5	Format the data collection file, as described in the *Oracle Trace Reporter User's Guide*. For example: `$ COLLECT FORMAT SAMPLE_DATA.DAT FORMATTED_SAMPLE.RDB` Oracle Trace produces a formatted Oracle Rdb database from the data collection file.

6	Generate an Oracle Trace report, as described in *Oracle Trace Reporter User's Guide*. For example:
	`$ COLLECT REPORT FORMATTED_SAMPLE.RDB /TYPE=SUMMARY -`
	`_$ /STATISTICS=ALL /FACILITY=RDBVMS -`
	`_$ /OUTPUT=SAMPLE_REPORT.TXT`
	Oracle Trace produces a report.

1.7 Oracle Trace Internal Components

Oracle Trace has three major internal components, the Registrar process, the administration database, and the history database. This section describes these components.

1.7.1 Oracle Trace Registrar Process

The **Registrar process** is a detached process that handles all communication between the applications running on your system and the Oracle Trace software. The Registrar tells your applications to start or stop collecting data and maintains a list (visible using the Oracle Trace SHOW REGISTER command) of the processes available for data collection. When you start the Oracle Trace software on your system (by using SYS$SYSTEM:EPC$STARTUP.COM), you are starting the Registrar process.

Note that a Registrar process must exist on every node from which you want to collect data. The OpenVMS SHOW SYSTEM command confirms that the Registrar process is running on your system. For example:

```
$ SHOW SYSTEM
VAX/VMS V5.4  on node MYVAX1 14-AUG-1995 10:06:22.04   Uptime  125 23:31:29
  Pid    Process Name    State  Pri      I/O       CPU       Page flts Ph.Mem
25200021 SWAPPER         HIB     16        0   0 00:33:18.22         0      0
25200062 SMITH           CUR      4     1132   0 05:17:49.86     23591    512
25200029 OPCOM           HIB      8     2847   0 00:06:34.99       749    134
   .
   .
   .
252001DA EPC$REGISTRAR   HIB      8    10955   0 00:04:36.88     21334    512
25200588 RDMS_MONITOR    LEF     15      558   0 00:00:02.29      2720     53
```

1.7.2 Oracle Trace Administration Database

The **administration database** is an Oracle Rdb database that Oracle Trace uses to store the following:

- Facility definitions
- Facility selections
- Data collection schedule information

When you install the Oracle Trace software on your system or VMScluster, the installation procedure creates the administration database.

The database is referred to as the **Oracle Trace administration database**. A system-wide logical name, EPC$ADMIN_DB, points to the location of the database.

1.7.3 History Database

The **history database** is a single Oracle Rdb database that contains all of the informational and error messages associated with data collection. There is only one history database per system or VMScluster. You reference the history database with the logical name EPC$HISTORY_DB.

The history database is the first place you should look whenever you encounter a problem with Oracle Trace. Use the SHOW HISTORY command to display any informational or error messages associated with a collection.

1.8 Details of How Oracle Trace Sets Up Collections

Table 1–6 describes how Oracle Trace sets up collections. The items numbered in Table 1–6 relate to items with corresponding numbers in Figure 1–1.

Table 1–6 Details of Collection Setup

When the ...	Then...
User starts an application	❶ Each facility in the application sends a registration message to the Oracle Trace Registrar process by calling the EPC$INIT system service routine. The registration message contains the facility number and version and Registration_ id. The Registrar also associates the following information with each facility: • EPID • Process name • User name • Image name The Registrar stores the facility information that it receives.
User schedules a collection	❷Oracle Trace stores collection-related information from the SCHEDULE COLLECTION and the CREATE SELECTION commands in the Oracle Trace administration database. Oracle Trace sends a message to the Registrar telling it to check new collection information in the administration database.

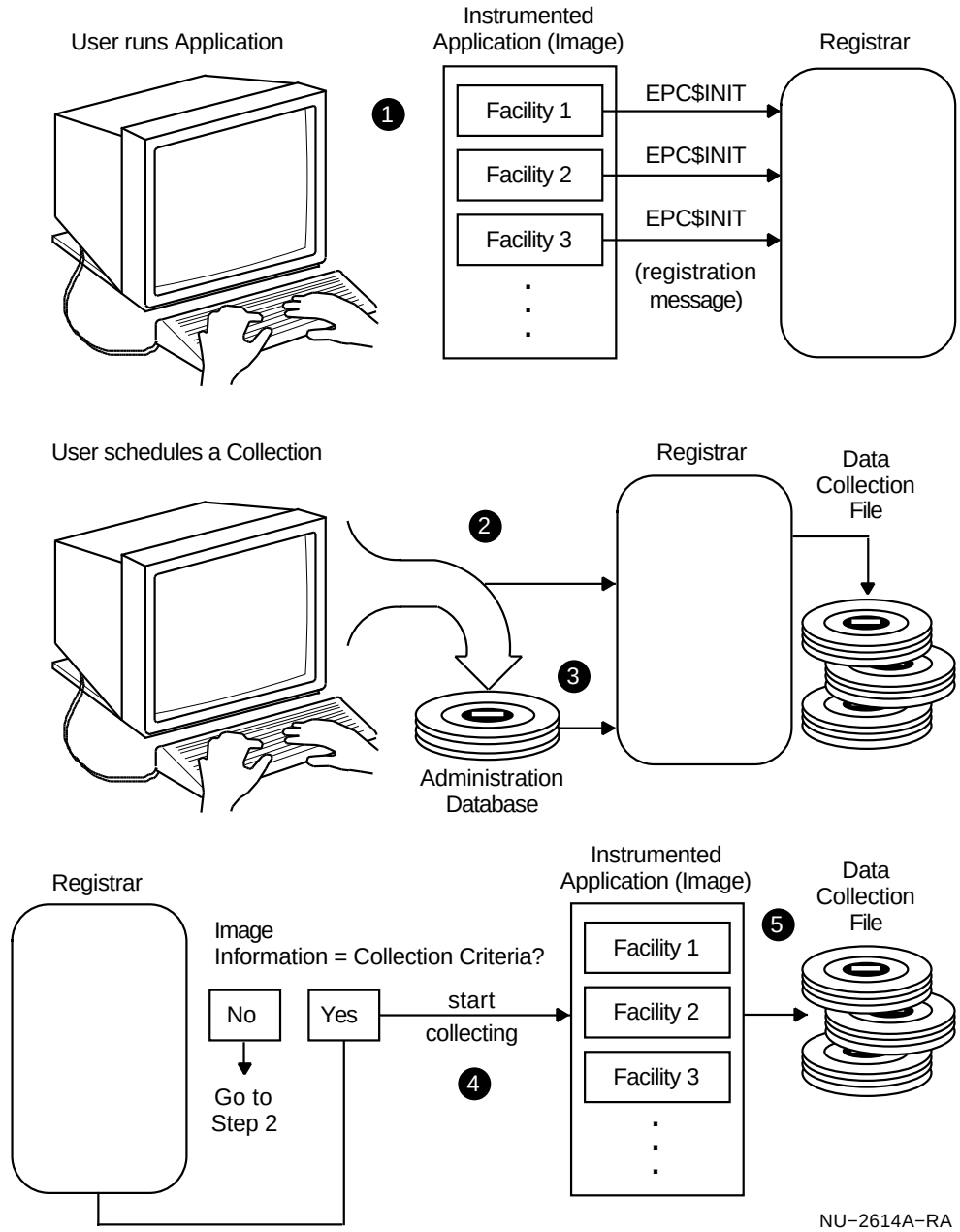
(continued on next page)

Table 1–6 (Cont.) Details of Collection Setup

When the ...	Then...
User-specified collection start time occurs	❸Registrar reads all information pertaining to this collection in the administration database and stores it in the data collection files associated with the collection. ❹Registrar compares the collection criteria for the collection with the image-related information that it has to determine if any images meet the collection criteria for the collection. When the Registrar determines that an image meets the collection criteria, it sends a "Start collecting" message to the image. ❺ The image begins recording event data to the data collection files. Collections continue until either the user enters a CANCEL COLLECTION command causing Oracle Trace to send the image a "Stop collecting" message or until the end time of the collection occurs.

Figure 1–1 illustrates details of how Oracle Trace sets up collections.

Figure 1–1 Details of Oracle Trace Collections



2
Creating a Facility Selection

A **facility selection** describes which data to collect during data collection. It specifies the individual facilities (applications and layered products) and the **class** of data to collect for each one. Creating a facility selection is a first step for general users of the Oracle Trace software who want to gather event-based data from facilities on their system. See Chapter 4 and Chapter 5 if you want to instrument your own application.

This chapter describes how to manipulate facility selections. It describes how to:

- Create selections
- Delete selections
- Display selections

Table 2–1 summarizes the Oracle Trace commands available to accomplish the functions associated with facility selections.

Table 2–1 Commands for Manipulating Collection Definitions

Command	Description
CREATE SELECTION	Creates a facility selection in the Oracle Trace administration database. Selection names must be unique in the Oracle Trace administration database. If you use the /REPLACE qualifier, your new facility selection replaces the old selection of the same name.
DELETE SELECTION	Deletes a facility selection from the Oracle Trace administration database.
SHOW SELECTION	Displays a facility selection in one of three formats: BRIEF, FULL, or NAMES_ONLY.

2.1 Choosing Which Data to Collect

The Oracle Trace software performs data collection on your system with minimal performance impact to the application and the system. However, the impact increases as you select more facilities from which to collect data, and as you specify more data to collect from each facility. The data files produced by Oracle Trace can become very large in a busy environment. If you are on a system with limited free disk space, you should carefully consider how much data you will be collecting. If you use Oracle Trace to collect all of the available data from all of the facilities on your system, you could run out of disk space very quickly.

When you decide to use Oracle Trace you probably have a specific function or area about which you want more information. For example, some of your applications might not be performing as well as you think they should, or you might need to generate statistics for a capacity planning report for the next fiscal year. Oracle Trace can collect a wide variety of data from the applications running on your system. However, you may not need to collect all of the possible data from all of the events occurring in the facilities you are using. It is possible to end up with much more information than you really need.

To make it easier to collect the data that you really need and to protect you from collecting data that you do not need, facilities can have one or more **collection classes** associated with them. These classes are subsets of all of the available events and items chosen by the facility developers for their significance to a specific function. For example, a facility might have specific collection classes defined for performance, workload analysis, capacity planning, or debugging purposes. The importance of selecting a collection class is illustrated by the following collection rates, which apply to the Debit/Credit workload for Oracle Rdb and Oracle CODASYL DBMS:

- 20,000+ blocks per 1 TPS/hour for ALL class data
- 10,000 blocks per 1 TPS/hour for RDBEXPERT class data
- 20,000 blocks per 1 TPS/hour for PERFORMANCE class data

You use the SHOW DEFINITION/FORMAT=NAMES_ONLY command to determine what classes are available for a given facility. For example, Example 2–1 shows that the NEW_FORMS facility has the collection classes ALL, CAPACITY_PLANNING, PERFORMANCE, and WORKLOAD (default). ATM_SAMPLE and MY_APPLICATION have only the default (ALL) class available.

Example 2–1 Display Format for SHOW DEFINITION /FORMAT=NAMES_ ONLY

```
16-SEP-1995 11:43       Facility Definition Information              Page 1
Names Only Report                                          Oracle Trace V2.2-0

 Facility:            Version:    Creation Date:      Class:
 -------------------  ----------  ------------------  -----------------
 ATM_SAMPLE           V1.0        25-AUG-1990 14:17   ALL               (D)
 MY_FACILITY          V2.0        12-AUG-1990 10:27   ALL               (D)
 NEW_FORMS            V4.0        13-JUN-1990 07:22   ALL
                                                      CAPACITY_PLANNING
                                                      PERFORMANCE
                                                      WORKLOAD          (D)
 .
 .
 .
```

Every facility has the ALL class, which contains all of the events and items defined for the facility. Furthermore, one of the classes is designated by the facility developers as the default class based on its predicted importance and frequency of use. Unless otherwise specified, the default is the ALL class, which is generated automatically when the developer creates a facility definition.

2.2 How to Ensure the Presence of a Facility Definition

A **facility definition** is a record in the administration database that describes the data that Oracle Trace can gather for a facility. Oracle Trace cannot collect data for a facility unless there is a facility definition for that facility in the administration database.

Installing Oracle Trace creates an administration database without any facility definitions. Before you can collect data for facilities you must learn if the administration database contains facility definitions for them. Use the following command to learn if your administration database contains a facility definition for Oracle Rdb:

```
$ COLLECT SHOW DEFINITION RDBVMS
```

If your administration database contains a facility definition for Oracle Rdb, Oracle Trace displays them as shown in Example 2–2. Note that Example 2–2 contains a definition for Oracle Rdb Versions 3.1-0, 4.0-0, and 6.0-05.

Example 2–2 Example List of Oracle Rdb Facility Definitions

```
30-AUG-1995 13:27       Facility Definition Information             Page 1
Names Only Report                                        Oracle Trace V2.2-0

 Facility:            Version:    Creation Date:      Class:
 -------------------  ----------  ------------------  -----------------
 RDBVMS               V6.0-05      9-AUG-1995 18:00   ALL
                                                      PERFORMANCE       (D)
                                                      PERFORMANCE_NO_CF
                                                      RDBEXPERT
                                                      RDBEXPERT_NO_CF
                      V3.1-0       3-JUN-1995 14:55   ALL
                                                      PERFORMANCE       (D)
                                                      PERFORMANCE_ALL_CF
                                                      PERFORMANCE_NO_CF
                                                      RDBEXPERT
                                                      RDBEXPERT_ALL_CF
                                                      RDBEXPERT_NO_CF
                      V4.0-0       3-JUN-1995 13:55   ALL
                                                      PERFORMANCE       (D)
                                                      PERFORMANCE_ALL_CF
                                                      PERFORMANCE_NO_CF
                                                      RDBEXPERT
                                                      RDBEXPERT_ALL_CF
                                                      RDBEXPERT_NO_CF
```

If your administration database does not contain a facility definition for Oracle Rdb, Oracle Trace displays a message similar to the one in Example 2–3.

Example 2–3 Example of No Facility Definition Message

```
%EPC-E-FACSHOW_FAIL, Show definition command failed
-EPC-E-FACSHO_NOEXIST, Facility definition RDBVMS * does not exist
%EPC-E-OPFAIL, Operation failed
```

If your administration database does not contain a facility definition for Oracle Rdb, ask your system manager to use the EPC$INSERT command procedure to enter its facility definition and definitions for other facilities on your system into the administration database.

2.3 Creating a Selection

An Oracle Trace facility selection consists of:

- Name of the selection
- List of facilities from which to collect data
- Classes of data to collect for each facility
- Comment describing the purpose of the selection

The format of the CREATE SELECTION command is:

CREATE SELECTION selection_name [/FACILITY=(facility_name[, . . .]) /COMMENT=" . . . " /OPTIONS[=file_spec] /REPLACE]

All facility selections are stored in the Oracle Trace administration database, and any user can reference a facility definition created by any other user. The facility selections remain in the database until they are deleted by their creator or by a user with BYPASS or SYSPRV privilege.

You use the CREATE SELECTION command to choose a subset of the available facilities from which you want to collect data. The following example defines the facility selection MY_SELECTION to collect the default class of data for ACMS:

```
$ COLLECT CREATE SELECTION MY_SELECTION /FACILITY=ACMS -
_$ /COMMENT="Collect the default DEC ACMS data"
```

To define a more detailed facility selection, you should use an options file. The /OPTIONS qualifier allows you to specify more than one facility and choose a different class of data for each facility. The qualifier takes the name of an options file as an argument. If you do not specify a file name, Oracle Trace prompts you for the options. Each facility must be defined on a separate line. The format of the facility description is:

FACILITY facility-name [/VERSION="version-code"] [/CLASS=class-name]

facility-name
The name of the facility from which to collect data.

version-code
A text string identifying the version of the facility. The string must be enclosed with quotation marks (" "). If you do not include the version code, Oracle Trace uses the most recent version of the facility stored in the Oracle Trace administration database. The SHOW DEFINITION /FORMAT=NAMES_ONLY

command lists the versions of facilities with the most recent at the top of the list. Note that you can only specify a single version of a particular facility in your selection.

class-name
The name of the class of data to collect for the facility. Note that you can only specify a single class for a particular facility in your selection.

The following example creates the facility selection COLLECT_ALL to collect all of the possible events and data items for Version 1.0 of MY_APPLICATION and the performance events and items for Oracle Rdb:

```
$ COLLECT CREATE SELECTION COLLECT_ALL /OPTIONS
Option> FACILITY MY_FACILITY /VERSION="1.0"/CLASS=ALL
Option> FACILITY RDBVMS /CLASS=PERFORMANCE
Option> CTRL/Z
%EPC-S-SELCRE, Selection COLLECT_ALL was created
$
```

Note

To avoid problems starting a collection you must always specify the correct facility version. If you are running multiple versions of a facility on your system, or if you have upgraded to a newer version, the administration database can have more than one facility definition for the facility. You must ensure that the version number you specify in your facility selection corresponds to the version of the facility that your process runs.

You can query the facility to learn the version your process is running. For example, for the Oracle Rdb facility you could use a command like this:

```
$ RMU/SHOW VERSION
Executing RMU for DEC Rdb V6.0-05
```

2.4 Deleting a Selection

You can delete a facility selection from the Oracle Trace administration database with the DELETE SELECTION command. You must be the creator of the selection or have OpenVMS BYPASS or SYSPRV privilege. Note that you cannot delete a facility selection if any data collection, either active or pending, is using that facility selection.

The following example deletes the facility selection MY_SELECTION:

```
$ COLLECT DELETE SELECTION MY_SELECTION /NOCONFIRM
%EPC-S-SELDEL_DELETED, Selection RDB_ALL_SELECTION was deleted
$
```

To delete a facility selection that is referenced by any active or pending collections, you must first cancel the data collection. See Section 3.4 for information on how to cancel all data collection using a particular facility selection.

You can use the SHOW SELECTION/FORMAT=NAMES_ONLY command to confirm the spelling of the names of facility selections that you want to delete.

2.5 Displaying Information About a Selection

You can display information about the facility selections stored in the Oracle Trace administration database using the SHOW SELECTION command. The command takes one argument: the name of a selection. If you do not specify a selection name, Oracle Trace displays information on all of the facility selections defined on the system.

You can specify the amount of information to display about a selection by using the /FORMAT qualifier to the SHOW SELECTION command. There are three valid format types:

- BRIEF (default)
- FULL
- NAMES_ONLY

2.5.1 BRIEF Format

If you specify /FORMAT=BRIEF with the SHOW SELECTION command, Oracle Trace lists the names of the facility selections in the Oracle Trace administration database together with the facilities, versions, and collection classes that each selection describes. If you did not specify a facility version when you created the facility selection, the most recently created version of the facility definition is used and "(latest)" is displayed.

The BRIEF format is useful if you want to check what facilities are in a facility selection. This will also show you if there is already a selection defined that suits your collection needs.

Example 2–4 shows a sample of the display produced with the SHOW SELECTION /FORMAT=BRIEF command.

Example 2–4 Display for SHOW SELECTION Using the Brief Format

```
9-MAY-1995 10:03         Facility Selection Information              Page 1
                                                          Oracle Trace V2.2-0

   Selection Name        Facility         Version     Class
   --------------------  ---------------  ----------  --------------------
   A1_DATA_ENTRY         OA               (latest)    ALL
                         TESTER           T4.1        ALL
                         RDBVMS           V3.1        ALL

   ACMSDBMS              ACMS             (latest)    PERFORMANCE
                         DBMS             (latest)    PERFORMANCE

   RDB_LOAD_TEST         RDBVMS           V4.0        RDBEXPERT
```

2.5.2 FULL Format

If you specify /FORMAT=FULL with the SHOW SELECTION command, Oracle Trace displays a full description of facility selections stored in the Oracle Trace administration database. You can display the description of a single selection if you include its name on the command line. For example, the following command would display the complete description of the FINANCE selection:

```
$ COLLECT SHOW SELECTION FINANCE /FORMAT=FULL
```

The FULL format display includes the following information:

- Name of the facility selection
- User name of the facility selection's creator
- Facility name, version, and collection class for each facility specified in the selection

Example 2–5 shows a sample of the display produced with the SHOW SELECTION /FORMAT=FULL command.

Example 2–5 Display for SHOW SELECTION Using the Full Format

```
9-MAY-1995 10:04         Facility Selection Information              Page 1
                                                        Oracle Trace V2.2-0

   Selection:           ACMSDBMS
   Comment:             This is the facility selection for DEC ACMS
                        and Oracle CODASYL DBMS performance data.
   Created By:          JONES

   Facility:            ACMS
   Version:             (latest)
   Collection class:    PERFORMANCE

   Facility:            DBMS
   Version:             (latest)
   Collection class:    PERFORMANCE
```

2.5.3 NAMES_ONLY Format

If you specify /FORMAT=NAMES_ONLY with the SHOW SELECTION command, Oracle Trace lists the names of the facility selections alphabetically. This format is useful to determine the correct spelling of a selection name or to determine if a particular selection name already exists in the Oracle Trace administration database.

Example 2–6 shows a sample of the display produced with the SHOW SELECTION /FORMAT=NAMES_ONLY command.

Example 2–6 Display for SHOW SELECTION Using the Names Only Format

```
9-MAY-1995 10:03         Facility Selection Information              Page 1
                                                        Oracle Trace V2.2-0

   Selection Name
   --------------------------------
   A1_DATA_ENTRY
   ACMSDBMS
   RDB_LOAD_TEST
```

3
Scheduling Data Collection

Scheduling data collection is the second step for general users of the Oracle Trace software. **Data collection** is the process of gathering event-based data from facilities and applications running on your system. You define the characteristics of a collection using the SCHEDULE COLLECTION command that allows you to specify:

- Which data to collect
- How much data to collect
- When to collect data
- Where to store collected data

This chapter describes how to work with Oracle Trace data collections including:

- Scheduling data collection
- Canceling data collection
- Displaying information about collections scheduled in the Oracle Trace administration database

Table 3–1 summarizes the Oracle Trace commands available to accomplish the functions associated with Oracle Trace data collection.

Table 3–1 Commands for Manipulating Data Collections

Command	Description
SHOW REGISTER	Shows the individual processes for which data can be collected.
SCHEDULE COLLECTION	Schedules data collection based on the specified qualifiers.
CANCEL COLLECTION	Stops data collection for an active collection. If a collection is pending, it removes the collection from the administration database.
SHOW COLLECTION	Shows data collection information in one of two formats, either BRIEF or FULL.
SHOW HISTORY	Shows all error or informational messages that have occurred during one or all data collections active on your system.

3.1 Limiting Collections to Specific Processes

You can limit the scope of data collections in two ways:

- Using a selection specification that limits collections to the facilities and data items that you specify. See Chapter 2 for more information.
- Using the /REGISTRATION_ID qualifier in the SCHEDULE COLLECTION command.

This section describes how to limit collections using the /REGISTRATION_ID qualifier. You can indicate the processes for which you want Oracle Trace to collect data by specifying any of the following information about each process with the /REGISTRATION_ID qualifier:

- Information associated with the process by the registration-id argument to the EPC$INIT routine. See Chapter 7 for more information.
- EPID
- Full image name including the device, directory specification, and version number
- User name
- Process name

You can specify this information in any order. However, when you use the /REGISTRATION_ID qualifier, Oracle Trace only collects for processes when there is a match between the information you provide in the /REGISTRATION_ID qualifier and the information the Oracle Trace associates with facilities and processes. Therefore, you must be sure that the information you provide in the /REGISTRATION_ID corresponds to the information associated with the processes you want to collect data for.

You can use the SHOW REGISTER command to learn what information is associated with each process. The SHOW REGISTER command shows all of the information you can use for the /REGISTRATION_ID qualifier except user name. See 3.1.1 to learn about using the SHOW REGISTER command.

When you specify an image name, you must either specify the full name including the device, directory specification, and version number or use a wildcard asterisk character (*).

The following command schedules a collection to gather data from any applications with "COURSES" as part of the image name:

```
$ COLLECT SCHEDULE COLLECTION ALL_COURSES COLLEGE.DAT-
_$ /SELECTION=JUST_RDB /NOCLUSTER -
_$ /BEGINNING=09:00 /ENDING=11:30 -
_$ /REGISTRATION_ID=(*COURSES*.EXE;*) -
_$ /COLLECTION_FILES=(PROTECTION=(G:RW))

%EPC_S_SCHED, Data collection ALL_COURSES is scheduled
```

See Section 3.3.1 for more information on collecting per-user data.

The following example tells Oracle Trace to select those processes on the cluster that have ORDER_ENTRY as a registration ID. Then Oracle Trace collects the DEC ACMS and Oracle Rdb data from those processes:

```
$ COLLECT SCHEDULE COLLECTION ORDER_WORK ORDERS.DAT -
_$ /SELECTION=ACMS_AND_RDB /CLUSTER -
_$ /BEGINNING=09:00 /ENDING=10:00 -
_$ /REGISTRATION_ID=ORDER_ENTRY -
_$ /COLLECTION_FILES=(PROTECTION=(W:RW))
%EPC_S_SCHED, Data collection ORDER_WORK is scheduled
```

Example 3–1 shows the Oracle Trace register after the collection ORDER_WORK activates. DEC ACMS and Oracle Rdb event data is collected from the following processes on MYVAX1: SMITH_2 and WRITER and from the following processes on MYVAX2: CONTRACT and JONES_2. Note that SMITH_2 also registered both the MY_FACILITY facility and Oracle CODASYL DBMS. Because MY_FACILITY and Oracle CODASYL DBMS are not included in the facility selection, Oracle Trace does not collect data from them. In addition, note that SECRETARY registered both DEC ACMS and

Oracle Rdb. However, the process does not have the correct registration ID, so no data is collected from it. An arrow (->) indicates that data collection is active for the facility listed on that line.

Example 3–1 Process Registration Display

```
12-JUN-1995 9:03:22.4          Register Information for node MYVAX1    Page 1
                                                          Oracle Trace V2.2-0

Registrations actively collecting

  Node:  MYVAX1

    Process   Process Name     Facility         Version   Registration ID
    --------  ---------------  ---------------  --------  ------------------
    21444556  JONES            RDBVMS           V3.1
    WORK$DISK:[FINANCE]DEBIT_CREDIT.EXE;11

    21544675  SMITH            RDBVMS           V3.1
    USER2:[GAMES]POKER.EXE;3

Registrations actively collecting

  Node:  MYVAX1       Collection:  ORDER_WORK     Selection:  ACMS_AND_RDB

    Process   Process Name     Facility         Version   Registration ID
    --------  ---------------  ---------------  --------  ------------------
->  21457890  SMITH_2          ACMS             V3.1      ORDER_ENTRY
                               DBMS             V4.1
                               MY_FACILITY      V2.0      DATA_ENTRY
    WORK$DISK:[TOOLS]INVENTORY_CHECK.EXE;5

->  21567444  WRITER           ACMS             V3.1      ORDER_ENTRY
    WORK$DISK:[TOOLS]SPELL_CHECK.EXE;5

12-JUN-1995 9:03:25.4          Register Information for node MYVAX2    Page 2
                                                          Oracle Trace V2.2-0

Registrations actively collecting

  Node:  MYVAX2

    Process   Process Name     Facility         Version   Registration ID
    --------  ---------------  ---------------  --------  ------------------
    21778900  SECRETARY        ACMS             V3.1      BUDGET_ANALYSIS
                               RDBVMS           V3.1
                               MY_FACILITY      V2.0      DATA_ENTRY
    WORK$DISK:[FINANCE]BUDGET_UPDATE.EXE;24

    21778902  JONES            ACMS             V3.1      BUDGET_ANALYSIS
    WORK$DISK:[FINANCE]DEBIT_CREDIT.EXE;11
```

(continued on next page)

Example 3–1 (Cont.) Process Registration Display

```
Registrations actively collecting

  Node:  MYVAX2       Collection:  ORDER_WORK     Selection:  ACMS_AND_RDB
    Process   Process Name     Facility         Version   Registration ID
    --------  --------------   --------------   --------  ------------------

->  21778888  CONTRACT         ACMS             V3.1      ORDER_ENTRY
->                             RDBVMS           V3.1
    WORK$DISK:[TOOLS]INVENTORY_CHECK.EXE;5

->  21778905  JONES_2          ACMS             V3.1      ORDER_ENTRY
->                             RDBVMS           V3.1
    USER1:[JONES.TESTS]ORDER_ENTRY_PROTO.EXE;1
```

3.1.1 Displaying Information About Process Registration

The SHOW REGISTER command allows you to examine the current state of process registration on your system or VMScluster. The SHOW REGISTER display is divided into the following segments:

- Registrations not collecting

  These processes do not have any active collections collecting data from them, but they are available for data collection if the proper collection is scheduled.

- Registrations actively collecting

  These processes have active data collection occurring.

If only the facility ID number is listed under the Facility heading, this represents an error condition where a facility has made a call to EPC$INIT, but there is no corresponding facility definition in the Oracle Trace administration database. No data can be collected for an undefined facility.

Example 3–2 shows a typical register display produced by the SHOW REGISTER/CLUSTER command. In the example, three separate processes are running the WEEKLY_CHECKS program which has registered three facilities: Oracle Rdb and two facilities for which no facility definitions exist in the Oracle Trace administration database. Another process is running the INVESTMENTS program which registered two facilities: Oracle Rdb and NEW_FORMS, both of which are being actively collected from. Lastly, the MORGAN process is running the MAINT_EMP program which also registered Oracle Rdb and NEW_FORMS, but data is only being collected from the NEW_FORMS facility.

Two local collections are active on the cluster: ALL_DAY is active on MYVAX1, and WED_2ND_SHIFT is active on MYVAX2. An arrow (->) indicates that data collection is active for the facility listed on that line.

Example 3–2 Display Format for SHOW REGISTER/CLUSTER

```
2-AUG-1995 21:25        Register Information for node MYVAX1            Page 1
                                                              Oracle Trace V2.2-0

Registrations not collecting

  Node: MYVAX1

    Process   Process Name     Facility  Version   Registration Id
    --------  ---------------  --------  --------  --------------------------
    00000021  Monday Payroll   2235                Payroll
                               2236                Stock Plan
                               RDBVMS    V3.1

    DISK$PAYROLL:[PAYROLL.IMAGES]WEEKLY_CHECKS.EXE;80

    00000025  Tuesday Payroll  2235                Payroll
                               2236                Stock Plan
                               RDBVMS    V3.1

    DISK$PAYROLL:[PAYROLL.IMAGES]WEEKLY_CHECKS.EXE;80

Registrations actively collecting

  Node: MYVAX1        Collection: ALL_DAY     Selection: FORMS_AND_RDB

    Process   Process Name     Facility  Version   Registration Id
    --------  ---------------  --------  --------  --------------------------
->  00000059  LINDA            NEW_FORM  T1.0-1    Investment Stream
->                             RDBVMS    V3.1

    DISK$PAYROLL:[PAYROLL.IMAGES]INVESTMENTS.EXE;3

    0000004D  Wed Payroll      2235                Payroll
                               2236                Stock Plan
->                             RDBVMS    V3.1
    DISK$PAYROLL:[PAYROLL.IMAGES]WEEKLY_CHECKS.EXE;80
```

(continued on next page)

Example 3–2 (Cont.) Display Format for SHOW REGISTER/CLUSTER

```
2-AUG-1995 21:25        Register Information for node MYVAX2           Page 2
                                                          Oracle Trace V2.2-0

Registrations actively collecting

  Node: MYVAX2  Collection: WED_2ND_SHIFT     Selection: JUST_FORMS

    Process   Process Name     Facility  Version   Registration Id
    --------  ---------------  --------  --------  --------------------------
->  00000027  MORGAN           NEW_FORM  T1.0-1    Payroll Stream
                               RDBVMS    V3.1
    DISK$PAYROLL:[PAYROLL.IMAGES]MAINT_EMP.EXE;3
```

3.2 Scheduling a Collection

You must schedule data collection on your system before Oracle Trace can begin gathering data. The SCHEDULE COLLECTION command lets you define collection characteristics. Data collection criteria include the output file for the collected data, the start and end times (or alternately, the duration), the facility selection to use, and whether to collect from your entire cluster or just the local node.

You can schedule multiple, concurrent collections, but be aware that the system overhead (especially the global sections) used by the Registrar process increases with each new collection. In addition, if a process registers and is immediately told to start collecting for more than five collections, a communications error occurs with a "mailbox full" condition and the process unregisters. If a process is actively collecting data and a new collection begins, the process can record information for that collection regardless of how many existing collections it is already recording data for.

You can specify data collection to occur either locally or cluster-wide using the /[NO]CLUSTER qualifier. By default, SCHEDULE COLLECTION schedules data collection to occur on every node in the cluster. To schedule a collection on a subset of the cluster, you must log in to each node that you want data collection to occur on and schedule a local collection on that node using the /NOCLUSTER qualifier. Note that on a standalone system, the /CLUSTER qualifier is ignored.

When you successfully schedule data collection, and again when the collection activates, Oracle Trace writes a confirmation message to the history database. You can examine the history database at any time with the SHOW HISTORY command. See Section 3.5.2 for information about the history database and the SHOW HISTORY command.

Note that you must have write access to both the administration and history databases in order to schedule a collection. See the SET ACCESS command in Chapter 6 for information on granting access to Oracle Trace databases. You must also ensure that the Oracle Trace collection process always has privileges to write to the collection file. Use the /COLLECTION_FILES qualifier of the SCHEDULE COLLECTION command to specify a protection of world read and write for the collection file.

3.2.1 Scheduling Data Collection on a Standalone System

If you have Oracle Trace installed on a standalone system, you can collect data from applications running on that system.

The following example schedules the collection MY_TEST to begin at 11:00 and end at 12:00 on the current day. The collection uses the facility selection MY_SELECTION and runs on the local node. Oracle Trace stores the collected data in the file MY_DATA.DAT in your default device and directory:

```
$ COLLECT SCHEDULE COLLECTION MY_TEST MY_DATA.DAT -
_$ /SELECTION=MY_SELECTION -
_$ /BEGINNING=11:00 /ENDING=12:00 -
_$ /COLLECTION_FILES=(PROTECTION=(WORLD:RW))
%EPC-S-SCHED, Data collection MY_TEST is scheduled
```

Alternately, you can use the /DURATION qualifier in place of the /ENDING qualifier. You must specify the duration as a relative OpenVMS time. For example:

```
$ COLLECT SCHEDULE COLLECTION MY_TEST MY_DATA.DAT -
_$ /SELECTION=MY_SELECTION -
_$ /BEGINNING=11:00 /DURATION="1:" -
_$ /NOCLUSTER -
_$ /COLLECTION_FILES=(PROTECTION=(WORLD:RW))
%EPC-S-SCHED, Data collection MY_TEST is scheduled
```

Note

The /COLLECTION_FILES qualifier specifies characteristics of the data collection files. You should usually use this parameter to specify a protection of world read and write for the collection file so that facilities always have privileges to write to the collection file.

3.2.2 Scheduling Data Collection on a Cluster

If you run Oracle Trace on a system that is a member of a VMScluster, your collections are automatically scheduled to run on all nodes in the cluster on which Oracle Trace has been started.

The following example schedules the collection MY_FULL_TEST to begin at 10:00 and end at 11:00 on the current day. The collection uses the facility selection MY_SELECTION and runs on every node in the cluster. Oracle Trace stores the collected data in the file CLUSTER_DATA.DAT in your default device and directory:

```
$ COLLECT SCHEDULE COLLECTION MY_FULL_TEST CLUSTER_DATA.DAT -
_$ /SELECTION=MY_SELECTION -
_$ /BEGINNING=10:00 /ENDING=11:00 -
_$ /COLLECTION_FILES=(PROTECTION=(WORLD:RW))
%EPC-S-SCHED, Data collection MY_FULL_TEST is scheduled
```

3.2.3 Scheduling Data Collection on Part of a Cluster

If you run Oracle Trace on a VMScluster but you do not want to collect data from every node, use the /NOCLUSTER qualifier when you schedule your data collection. This will schedule a **local collection** (data collection that occurs only on your local node).

The following example schedules the collection MY_LOCAL_TEST to begin at 11:00 and end at 12:00 on the current day. The collection uses the facility selection MY_SELECTION and runs on the local node (that is, the node you are currently logged in to). Oracle Trace stores the collected data in the file LOCAL_DATA.DAT in your default device and directory:

```
$ COLLECT SCHEDULE COLLECTION MY_LOCAL_TEST LOCAL_DATA.DAT -
_$ /SELECTION=MY_SELECTION -
_$ /BEGINNING=11:00 /ENDING=12:00 -
_$ /NOCLUSTER
_$ /COLLECTION_FILES=(PROTECTION=(WORLD:RW))
%EPC-S-SCHED, Data collection MY_LOCAL_TEST is scheduled
```

To schedule data collection on a subset of the cluster, you must log in to each node and schedule a local collection on that node using the /NOCLUSTER qualifier. Note that you should use a different name for the data files produced by each of these collections. One suggestion is to include the node name as part of the data file name. If you do not use different names for the data files, Oracle Trace uses the file version number to distinguish each file. The individual data files can be combined later using the FORMAT command. See the *Oracle Trace Reporter User's Guide* for information on combining multiple data files.

Example 3–3 shows a sample command procedure that schedules a local collection with the node name included as part of both the collection and data file names.

Example 3–3 Sample Command Procedure for Starting a Local Collection

```
$!
$!                        LOCAL_COLLECTION.COM
$!
$! This procedure starts Oracle Trace data collection using the
$! node name as part of both the collection name and the data file
$! name.
$!
$! Find the name of the local node and append it to the collection
$! and data file names
$!
$ node_name  = F$GETSYI("NODENAME")
$ collection_name  = "MY_COLL_''node_name'"
$ data_file  = "DATA_''node_name'.DAT"
$!
$! Put your facility selection name and start and end times here:
$!
$ selection_name = "MY_SELECTION"
$ start_time = "12:00"
$ end_time   = "13:00"
$!
$! Schedule your local data collection
$!
$ COLLECT SCHEDULE COLLECTION  'collection_name' 'data_file' -
     /SELECTION='selection_name' -
     /BEGIN='start_time' /END='end_time' -
     /NOCLUSTER /COLLECTION_FILES=(PROTECTION=(W:RW))
$ EXIT
```

If you have the OpenVMS OPER privilege, you can use the OpenVMS System Management (SYSMAN) utility to schedule local data collection on nodes in your cluster without logging into each node.[1] SYSMAN is a utility that centralizes the management of nodes and clusters by allowing you to define an environment that can be a particular node, a group of nodes, or a cluster. You can perform tasks on all nodes in the environment from your local node. The following example shows how you can schedule local data collection on nodes MYVAX1 and MYVAX2 from your local node:

[1] If OPER is not a default privilege for your process, you must use the SET PROFILE /PRIVILEGE=OPER command in SYSMAN. Refer to the OpenVMS documentation for information about the SYSMAN utility.

```
$ RUN SYS$SYSTEM:SYSMAN
SYSMAN> SET ENVIRONMENT/NODE=(MYVAX1,MYVAX2)
%SYSMAN-I-ENV, Current Command Environment:
        Individual nodes: MYVAX1,MYVAX2
        Username SMITH will be used on nonlocal nodes
SYSMAN> DO @LOCAL_COLLECTION.COM
%SYSMAN-I-OUTPUT, Command execution on node MYVAX1
%SYSMAN-I-OUTPUT, Command execution on node MYVAX2
SYSMAN> EXIT
$
```

3.3 Data Collection Files

When you schedule data collection, you can specify one or many data files to store the event data (refer to the SCHEDULE COLLECTION command in Chapter 6 for details). In addition, one or many different executable images (running in the context of one or many processes) that are collecting data on the local system or VMScluster can record the data in a common data collection file. By default, there is one data collection file for all of the data related to a particular collection. However, if you anticipate that the amount of data to be collected is too great for the I/O bandwidth of the disk where the data file exists or that the data could exceed the capacity of the disk, you can specify more than one file when scheduling data collection. In this case, each data file would be on a different disk device. See the *Oracle Trace Reporter User's Guide* for information on how to combine multiple data files after data collection has ended.

Note that when you specify more than one file, Oracle Trace performs load balancing among all processes gathering data for that collection. An individual process only connects to one data collection file and does not fail-over to the next file when the file is full or when any other exception occurs. If all of the data collection files fill up before the scheduled end of the collection, the collection aborts so that new processes do not attempt to record data to the already-full files.

The following example schedules data collection using three data collection files, each on a separate device:

```
$ COLLECT SCHEDULE COLLECTION TSTVAX_WORKLOAD -
_$ USER1:[SMITH.DATA]TST1.DAT,USER2:[DATA]TST2.DAT, -
_$ USER3:[DATA]TST3.DAT /SELECTION=ACMS_AND_RDB -
_$ /BEGIN=09:00 /END=17:00 /CLUSTER -
_$ /COLLECTION_FILES=(PROTECTION=(WORLD:RW))
```

Alternately, you could use a **file list**, which is a file containing a list of file specifications to use as output files. Note that the /FILELIST qualifier is position-dependent; you must specify it on the second parameter to the command. For example:

```
$ COLLECT SCHEDULE COLLECTION TSTVAX_WORKLOAD -
_$ DATA_LIST.TXT /FILELIST -
_$ /COLLECTION_FILES=(PROTECTION=(WORLD:RW)) -
_$ /SELECTION=ACMS_AND_RDB /BEGIN=09:00 /END=17:00 /CLUSTER
```

Each file specification must be on a separate line within the file list. For example:

```
USER1:[SMITH.DATA]TST1.DAT
USER2:[DATA]TST2.DAT
USER3:[DATA]TST3.DAT
```

The VAX Distributed File Service (DFS) is an OpenVMS layered product that lets you access remote files as if they were local. Any number of users can use DFS to read files, but only one user at a time can write to or update any one file. Do not put your data collection files on a device that is served by the VAX Distributed File Service. An RMS exception is returned when both the Oracle Rdb monitor and the Oracle Trace Registrar attempt to access a data collection file simultaneously.

3.3.1 File Protection Schemes

Oracle Trace uses the default file protection for your process when creating the data collection files for a collection. For a process to record event data to your file, it must have both read and write access to that file. Use the PROTECTION parameter to the /COLLECTION_FILES qualifier to the SCHEDULE COLLECTION command to automatically set the protection on your data collection files. Note that the data collection files are created immediately when the collection is scheduled, not when the collection becomes active.

The format of the parameter is:

PROTECTION=(ownership:access[, . . .])

Where:

- OWNERSHIP is one of the following: (S)ystem, (O)wner, (G)roup, or (W)orld.
- ACCESS is any combination of the following: (R)ead, (W)rite, (E)xecute, or (D)elete.

If you do not specify a value for each ownership category, or if you omit the PROTECTION parameter, the Oracle Trace software applies the current default protection for each unspecified category. If the data collection file replaces a previous version, then the protection on the old file is used on the new one.

The following example schedules a collection where all members of the user's UIC-based group have read and write access to the data collection file:

```
$ COLLECT SCHEDULE COLLECTION ALL_DAY MONDAY.DAT -
_$ /SELECTION=EVERYTHING /BEGIN=9:00 /END=17:00 -
_$ /COLLECTION_FILES=(PROTECTION=(GROUP:RW))
```

3.4 Canceling Data Collection

You can cancel data collection that is active or pending with the CANCEL COLLECTION command. When you cancel an active data collection, Oracle Trace stops recording data for the collection but does not delete the collection's data files. You can format and create reports from the data files as if the data collection had run to completion. Oracle Trace also writes a message to the history database indicating that the data collection has been cancelled. You can examine the history database at any time with the SHOW HISTORY command. See Section 3.5.2 for information about the history database and the SHOW HISTORY command.

When you cancel a collection it does not terminate immediately, but is set to an aborting state. For an active collection, the Registrar process (or processes in a VMScluster environment) sends a "Stop collecting" message to each process that is recording event data. This communication takes place very quickly, but if you cancel a collection and immediately enter the SHOW COLLECTION command, your collection will still exist in an aborting state. A collection that is aborting is indicated in the SHOW COLLECTION display by two asterisks (**) next to the entry. Whenever you cancel an active collection you receive an informational message from the Oracle Trace Registrar stating that the collection has been set to the aborting state:

```
$ COLLECT CANCEL COLLECTION TEMP /NOCONFIRM
%EPC-S-SCHED_ABTNG, Data collection TEMP has been set to aborting
```

In a cluster, CANCEL COLLECTION cancels data collection either locally or cluster-wide depending on how the collection was originally scheduled. If you scheduled data collection with the /NOCLUSTER qualifier, CANCEL COLLECTION cancels the collection at the local node. If you scheduled data collection with the /CLUSTER qualifier, CANCEL COLLECTION cancels the collection on all nodes in the cluster. Note that you cannot cancel collections

from individual nodes in a cluster environment if you scheduled the collection using /CLUSTER.

The following example cancels the collection MYTEST, which was originally schedule to run on the local node:

```
$ COLLECT CANCEL COLLECTION MY_TEST /NOCONFIRM
%EPC_S_SCHED_CANCEL, Collection MY_TEST is cancelled.
```

You can cancel all of the collections (active or pending) that use a particular facility selection if you specify the /SELECTION qualifier. This option is useful if you typically schedule multiple data collections referencing the same facility selection. It is also needed when you want to delete a facility selection. To do this, you must first cancel all collections using that selection.

The following example cancels all data collections that are scheduled with the facility selection TEMP_SELECTION:

```
$ COLLECT CANCEL COLLECTION /SELECTION=TEMP_SELECTION /NOCONFIRM
%EPC_S_SCHED_CANCEL, Collection MY_COLL_MYVAX1 is cancelled.
%EPC_S_SCHED_CANCEL, Collection MY_COLL_MYVAX2 is cancelled.
%EPC_S_SCHED_CANCEL, Collection TEMP_TEST is cancelled.
```

3.5 Displaying Information About Data Collection

This section describes the commands that allow you to examine the status of data collection on your system. The actual recording of event data is the third step, in the Oracle Trace collection process. You can examine the schedule of active and pending collections, and you can display any errors, warnings, or informational messages encountered by the collections.

3.5.1 Displaying the Schedule for Data Collection

Collections exist on your system in either an active or a pending state.

The Oracle Trace administration database maintains the schedule of data collection on your VMScluster. You can display information about scheduled collections with the SHOW COLLECTION command.

Example 3–4 shows a sample of the display produced with the SHOW COLLECTION/FORMAT=BRIEF command. The arrow (->) next to the line describing the MY_FULL_TEST and PAYROLL collections indicate that the collections are active.

Example 3–4 Display for SHOW COLLECTION Using the Brief Format

```
9-MAY-1995 10:09            Scheduled Collections                    Page 1
Brief Report                                              Oracle Trace V2.2-0

Collections scheduled for the entire cluster

    Selection Name   Collection Name  Start            End
    --------------   --------------  --------------   --------------
->  MY_SELECTION     MY_FULL_TEST     9-MAY-95 10:00   9-MAY-95 11:00
->  RDB_WORKLOAD     PAYROLL          9-MAY-95 10:00   9-MAY-95 17:00

Oracle Trace Collection Schedule for node MYVAX1

    Selection Name   Collection Name  Start            End
    --------------   --------------  --------------   --------------
    MY_SELECTION     MY_COLL_MYVAX1   9-MAY-95 10:00   9-MAY-95 13:00

Oracle Trace Collection Schedule for node MYVAX2

    Selection Name   Collection Name  Start            End
    --------------   --------------  --------------   --------------
    MY_SELECTION     MY_COLL_MYVAX2   9-MAY-95 12:00   9-MAY-95 14:00


Oracle Trace Collection Schedule for node SMTHVX

    Selection Name   Collection Name  Start            End
    --------------   --------------  --------------   --------------
    MY_SELECTION     MY_LOCAL_TEST    9-MAY-95 11:00   9-MAY-95 12:00
```

3.5.2 Displaying the History for Collections

The Oracle Trace **history database** contains a record of the informational and error messages that are encountered during data collection. The SHOW HISTORY command allows you to discover if any errors occurred during a collection (or many collections). You can also display any informational messages that occurred during data collection. An example of an informational message is a confirmation message sent back to Oracle Trace by a process that has actually begun data collection, or a message that a process has registered with Oracle Trace and is available for collection. Displaying informational messages is most useful for facility developers and support personnel, while displaying errors is useful for general users.

The format of the SHOW HISTORY command is:

SHOW HISTORY collection-name [/BEFORE="time" /[NO]CLUSTER /FORMAT=type /NODE=node-name /OUTPUT=file-spec /SINCE="time"]

Note that a process must have write access to the history database in order to log scheduling errors to it. However, for only read access is required for run-time collection errors. See the SET ACCESS and SHOW ACCESS commands in Chapter 6 for information.

Valid format types are ERROR, INFORMATIONAL, and ALL which includes both errors and informational messages.

You should examine the history database during and after data collection to verify that the collection started successfully and did not encounter any serious errors. Note that when a process registers or begins collecting data, or when a collection begins, the informational message might not appear in the history database for 10-15 seconds. When the message does appear, it has the correct timestamp.

You should also always examine the history before formatting your collected data. Note that you can format data files from collections that failed during active data collection, but you might not achieve the full results that you wanted.

An error encountered by a process which has registered with Oracle Trace is not related to any collection for which the process might be recording event data. If you use the COLLECTION-NAME parameter to the SHOW HISTORY command, you do not see errors or messages generated by the individual processes. For example, if a process is unable to record data due to a file protection violation on the data collection file, the message is on the SHOW HISTORY /FORMAT=ALL report; not the report for a specific collection. Example 3–5 shows the display for the SHOW HISTORY/FORMAT=ERROR command and Example 3–6 shows the display for the SHOW HISTORY /FORMAT=INFORMATIONAL command.

Example 3–5 Display for SHOW HISTORY /FORMAT=ERROR

```
04-APR-1995 10:03           Data Collection History                    Page 1
Oracle Trace error history for cluster                    Oracle Trace V2.2-0

For database:  EPC$HISTORY_DB

Collection:

Date and Time            EPID      Process Name     Registration Id
-----------------------  --------  ---------------  --------------------
03-APR-1995 12:34:16.48  31600062  ACMS_USER1       Personnel
   %SYSTEM-W-DEVICEFULL, device full - allocation failure
   %EPC-E-HST_PRCERR, Error received from collecting process

04-APR-1995 09:47:42.73  316000D6  DB_MANAGER
   %EPC-E-OPEDCF, Error opening data collection file
   %EPC-E-HST_PRCERR, Error received from collecting process

04-APR-1995 09:47:47.07  316000D6  DB_MANAGER
   %EPC-E-HST_PRCERR, Error received from collecting process
   %SYSTEM-F-NOPRIV, no privilege for attempted operation
```

Example 3–6 Display for SHOW HISTORY /FORMAT=INFORMATIONAL

```
04-APR-1995 10:05           Data Collection History                    Page 1
Oracle Trace informational history for cluster             Oracle Trace V2.2-0

For database:  EPC$HISTORY_DB

Collection:  EPC$IVP_COLLECTION
Node:  MYVAX1

Date and Time            EPID      Process Name     Registration Id
-----------------------  --------  ---------------  --------------------
03-APR-1995 15:03:24.74  38A00B79  _RTA16:
   %EPC-S-HST_SCHED, Data collection scheduled

03-APR-1995 15:03:29.38  38A0124E  EPC$REGISTRAR    EPC$IVP_SELECTION
   %EPC-S-HST_START, Collecting started

03-APR-1995 15:03:53.55  38A00B79  _RTA16:
   %EPC-S-HST_START_COLL, Process started collecting data

03-APR-1995 15:04:55.20  38A00B79  _RTA16:
   %EPC-S-HST_ABORT, Collection aborting

03-APR-1995 15:04:58.04  38A0124E  EPC$REGISTRAR
   %EPC-S-HST_END, Collecting ended

03-APR-1995 15:05:01.50  38A0124E  EPC$REGISTRAR
   %EPC-S-HST_DELETED, Collection deleted
```

4
Instrumenting Applications

This chapter describes how to instrument your applications. Instrumenting source code is the first step, for application programmers using Oracle Trace. General users do not need to add anything to their applications to collect data, if these applications call products such as Oracle Rdb that are already instrumented with Oracle Trace routines. You will need to instrument your application source code routines if:

- You want to collect event-based data for your application.
- You want to relate data between your application and other facilities using the cross-facility capability. With this capability, you can gain a better understanding about the context in which some events are being executed and distinguish between the resources your application uses and the resources other facilities use on its behalf.
- You want to collect event-based data for individual threads of a multithreaded application.

Note

You do not have to instrument your application with Oracle Trace service routine calls if your application calls a facility and you only want to collect the event-based data for that facility. An example would be if your application uses Oracle Rdb, DEC ACMS, or Oracle RALLY, and you are interested only in the events and items of these facilities.

Instrumenting is the process of:

1. Modifying your application by adding Oracle Trace service routine calls that log the events and items for collection.
2. Compiling and linking your application.
3. Creating a facility definition and entering it into the administration database, as described in Chapter 5.

4. Creating a binary version of your facility definition, as described in Section 5.4.
5. Transporting facility definitions, if you want to run your facility at other sites, as described in Section 5.4.

This chapter focuses on steps 1 and 2. It presents information in the order you will use it, and it also presents tasks in the order you might perform them.

4.1 The Oracle Trace Service Routines

Oracle Trace provides routines that allow you to record events that occur during the execution of your application program (executable image.) These routines collect data for each instrumented facility in an application program. Oracle Trace records the data in one or more data collection files that can be common to one or many different processes that are also collecting data on your system or VMScluster. Table 4–1 lists all of the Oracle Trace service routine calls for instrumenting applications. Chapter 7 describes each service routine in detail.

Table 4–1 Oracle Trace Service Routines

For All Applications	
Use this Routine	**If you want to . . .**
EPC$INIT	register the facility with Oracle Trace. You must use this routine before your application executes any event-logging calls. This routine registers your specific application with Oracle Trace.

(continued on next page)

Table 4–1 (Cont.) Oracle Trace Service Routines

For All Applications Collecting Duration and Point Events	
Use this Routine	**If you want to . . .**
EPC$START_EVENT	record the start of a duration event.
EPC$START_EVENTW	record the start of a duration event and wait for processing to complete before returning.
EPC$END_EVENT	record the end of a duration event.
EPC$END_EVENTW	record the end of a duration event and wait for processing to complete before returning.
EPC$EVENT	record the execution of a point event.
EPC$EVENTW	record the execution of a point event and wait for it to complete before returning.

For Applications Collecting Cross-Facility Items	
Use this Routine	**If you want to . . .**
EPC$SET_CF_VALUE	assign a value to a single cross-facility item.
EPC$SET_CF_ITEMS	assign a value to all cross-facility items.
EPC_GET_CF_ITEMS	retrieve to all cross-facility items to a buffer. This routine pertains to client/server or multithreaded applications and relies on the EPC$SET_CONTEXT routine to change the context for cross-facility items.

For Multithreaded Applications Collecting Resource or Cross-Facility Items on a per Thread Basis	
Use this Routine	**If you want to . . .**
EPC$SET_CONTEXT	denote the start of a new thread.
EPC$DELETE_ CONTEXT	denote the end of a thread.

An application program can be made up of one or many facilities. For example, an application can consist of its own server code, ACMS, and an application database (probably an Oracle Rdb or an Oracle CODASYL DBMS database). Each of these component facilities is responsible for issuing calls to the Oracle Trace service routines to collect data. The service routines record data in data collection files.

Each facility in the application image must contain a call to EPC$INIT to register with Oracle Trace. Note that each facility needs to call EPC$INIT only once for each image activation. The call to EPC$INIT must precede any other calls to Oracle Trace. Oracle Corporation recommends placing the EPC$INIT call early in the execution of your facility or application. Facilities use the EPC$START_EVENT(W) and EPC$END_EVENT(W) calls to collect data for duration events and the EPC$EVENT(W) call to collect data for point events.

For programmers who want better information about the interfacility context in which some events are being executed, Oracle Trace allows them to relate event items from one application to items associated with one or more events in other facilities or applications. Related items are called **cross-facility items**. Oracle Trace provides the EPC$SET_CF_ITEMS routine to assign values to all cross-facility items in an application, the EPC$SET_CF_VALUE routine to assign a value to a single cross-facility item, and the EPC$GET_CF_ITEMS routine to retrieve all cross-facility items into a buffer. The EPC$GET_CF_ITEMS routine mostly pertains to client/server or multithreaded applications and relies on the EPC$SET_CONTEXT routine to change the context for cross-facility items. Oracle Trace can automatically log cross-facility items for any other facility, as long as its facility definition defines these items.

Oracle Trace can collect two other kinds of items for events in a facility: standard resource utilization items that Oracle Trace automatically gathers for each event and facility-specific items that you define for your application.

With Oracle Trace, a multithreaded facility (a multithreaded server, for example) can collect resource utilization items on a per-thread basis using the EPC$SET_CONTEXT and EPC$DELETE_CONTEXT calls to denote when a thread starts and completes executing.

4.2 How Oracle Trace Interacts with a Facility

An understanding of how Oracle Trace interacts with a facility can help you to instrument your application effectively. In general, Oracle Trace keeps track of the information to be collected based on the facility definition and the facility selection associated with each collection. The facility definition is a general description of all of the data that can possibly be collected for a facility. It describes all of the events that are instrumented in an application and their associated items. The facility definition can specify many classes of items for each event. On the other hand, the facility selection is a more limited description of the data to be collected for a specific collection. It usually specifies only one class of items for an event, and might specify only a subset of the application's events.

Table 4–2 shows how Oracle Trace interacts and shares information about what it is collecting with a facility.

Table 4–2 Detail of Oracle Trace and Facility Interaction

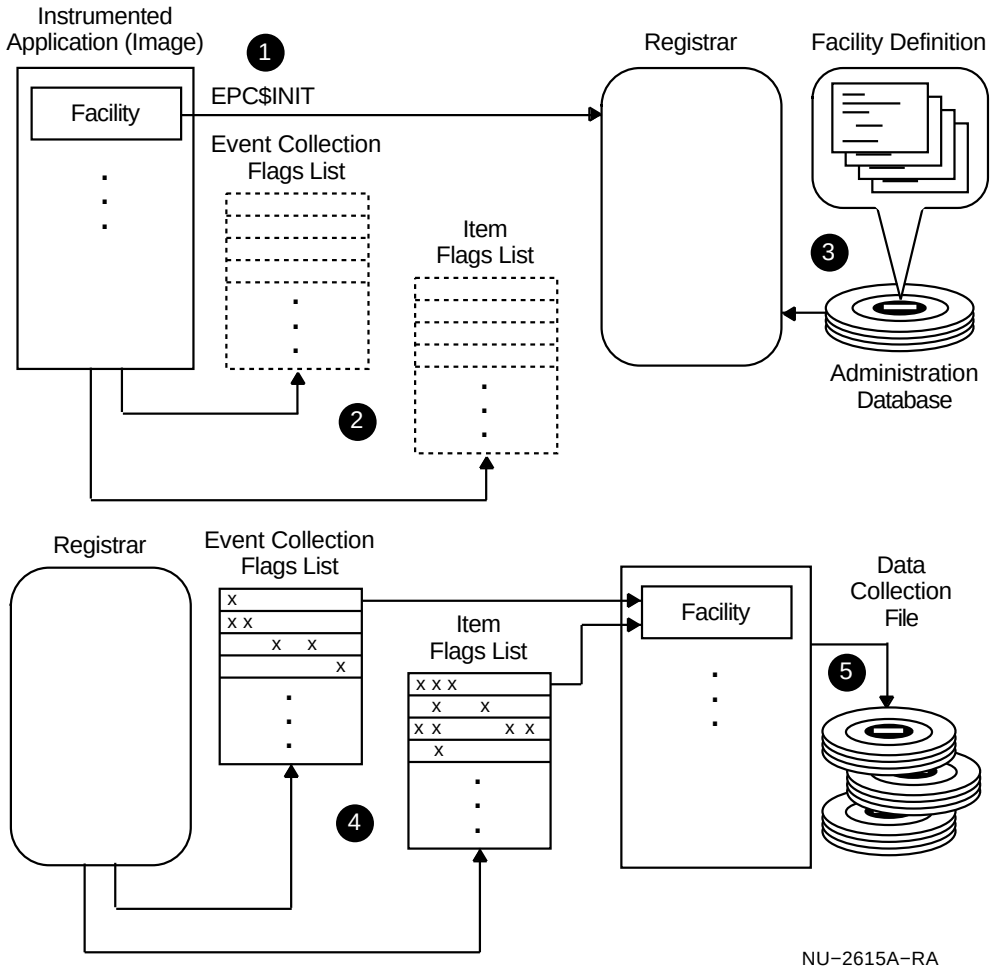
When . . .	Then . . .
The user runs an image	❶Each facility in the image registers with Oracle Trace by sending a registration message via an EPC$INIT call. The registration message contains the facility name, facility version, and optionally a REGISTRATION_ID value.
	❷The facility can optionally pass the address of the event collection and the item flags lists, so that it can perform an efficient check to learn if Oracle Trace is collecting data for an event before it calls an Oracle Trace service routine.
	❸The Registrar examines the facility definition associated with any active processes to determine what events and items to collect.
	❹If a collection is scheduled while the facility is running, the Registrar sets bits in the event collection flags list based on the facility selection associated with the collection[1].
The application executes an EPC$START_EVENT(W), EPC$END_EVENT(W), or an EPC$EVENT(W) call.	❺Oracle Trace and optionally the facility code examines the event collection and the item flags lists. If the bit corresponding to an event is set, Oracle Trace collects data for the event. Likewise, if the bit is not set, Oracle Trace does not collect data for the event.

[1] If the collection is running before the facility starts, the Registrar examines the facility selection associated with the collection and examines the facility definition after the facility registers. In either case, the Registrar determines what to collect based on both the facility definition and selection, and sets the appropriate bits in the event collection flags list.

Figure 4–1 shows how Oracle Trace interacts with facilities.

Oracle Trace's dependence on the information in the facility selection to determine the events to collect gives programmers flexibility when they instrument an application and users flexibility when they collect data for an application. Programmers can instrument an application to collect data for many kinds of events for various purposes. Users can collect data for a subset of these events, or avoid collecting data altogether. For example, users can create a facility selection for a collection that focuses on application-related events to evaluate the application's performance. If they want to focus on how people use the application, for example to learn how many deposits, withdrawals, or requests for balance that customers might make in

Figure 4–1 Oracle Trace and Facility Interaction



an Automated Teller Machine (ATM) application, they can create a different facility selection.

Although this arrangement provides users maximum flexibility when they collect events, it could negatively impact application performance if the application made calls to Oracle Trace whether or not Oracle Trace was collecting for that event. That is why applications should test the event collection flags list before executing each call to Oracle Trace to be sure that collections are set for that event. In this way, applications do not make unnecessary calls to Oracle Trace, and users can alter the events and items

collected by specifying different classes of events and items in the facility selection, rather than modifying their applications.

4.3 Oracle Trace Buffering

Oracle Trace stores all of the data that it collects in private buffers. Under normal circumstances, Oracle Trace writes the contents of these buffers out to a data collection file when either of the following occurs:

- A buffer becomes full.
- A flush interval occurs, if one is specified in the SCHEDULE COLLECTION command.

When a process that is collecting data terminates, Oracle Trace writes the data that has accumulated in its buffers to the data collection file when:

- The application performs a normal exit.
- The application terminates abnormally (for example, producing an Oracle Rdb bugcheck).
- The application is stopped using Ctrl/C or Ctrl/Y and STOP.
- The collection is cancelled (using the CANCEL COLLECTION command) or the end time of the collection is reached.
- The $FORCEX system service is used to delete the process.

Note that data in the collection buffer is lost if STOP/ID is used to stop the process. STOP/ID calls the $DELPRC service which does not call the EPC$SHR.EXE shareable image exit handler.

Note

Oracle Corporation recommends that to avoid losing collection data, you never stop collections using the STOP/ID command. Instead, use the other methods described in this section.

See the /COLLECTION_FILES qualifier to the SCHEDULE COLLECTION command in Chapter 6 to learn how to set the size of the collection buffer. See the /FLUSH_INTERVAL qualifier of the SCHEDULE COLLECTION command in Chapter 6 to learn how to control how often Oracle Trace writes its buffer to the data collection file.

4.4 Aids to Instrumenting

Oracle Corporation provides a number of aids to help you instrument your application including:

- Examples of instrumented applications
- A Language-Sensitive Editor (LSE) environment
- Language definition files for a number of programming languages

This section describes each of these aids.

4.4.1 Examples of Instrumented Applications

Oracle Corporation provides examples of instrumented applications written in FORTRAN, C, COBOL, and Pascal. Table 4–3 describes these applications. You can find these examples in EPC$EXAMPLES.

Table 4–3 Sample Applications in EPC$EXAMPLES and Their Associated Files

Application	Data file[1]	Facility Definition Command Procedure
EPC$ATM-SAMPLE-EXTENDED.C[2]	None, it creates its own Oracle Rdb database	EPC$ATM-FAC-DEF-EXTENDED.COM
EPC$ATM-SAMPLE-EXTENDED.COB[3]	EPC$ATM-SAMPLE.DAT	EPC$ATM-FAC-DEF-EXTENDED.COM
EPC$ATM-SAMPLE-EXTENDED.EXE[2]	None, it creates its own Oracle Rdb database	EPC$ATM-FAC-DEF-EXTENDED.COM
EPC$ATM-SAMPLE.FOR	EPC$ATM-PASCAL-FORTRAN.DAT	EPC$ATM-FAC-DEF.COM
EPC$ATM-SAMPLE.PAS	EPC$ATM-PASCAL-FORTRAN.DAT	EPC$ATM-FAC-DEF.COM
EPC$ATM-SAMPLE.COB	EPC$ATM-SAMPLE.DAT	EPC$ATM-FAC-DEF.COM
EPC$ATM-SAMPLE.EXE	EPC$ATM-C.DAT	EPC$ATM-FAC-DEF.COM

[1]Data files contain ten account records numbered from 1 to 10.

[2]All of the sample applications are instrumented to collect standard resource data items. The EPC$ATM-SAMPLE-EXTENDED.C application is instrumented to collect facility-specific and cross-facility data items as well.

[3]All of the sample applications are instrumented to collect standard resource data items. The EPC$ATM-SAMPLE-EXTENDED.COB application is instrumented to collect facility-specific data items as well.

See Section 1.4 for instructions if you want to run or modify these applications. Appendix A contains an example of an instrumented application.

4.4.2 LSE Environment Files

An LSE environment file resides in SYS$SHARE. The file contains all Oracle Trace service routine call templates. To use this environment file, specify /ENVIRONMENT=EPC$LSE_ENVIRONMENT.ENV on the LSEDIT DCL command. For example:

```
$ LSEDIT /ENVIRONMENT=EPC$LSE_ENVIRONMENT MY_PROGRAM.C
```

The environment file contains support for BASIC, BLISS, C, COBOL, FORTRAN, Pascal, and PL/I.

4.4.3 Language Definition Files

Oracle Trace provides definition files in SYS$SHARE. The Oracle Trace definition files include external routine definitions for the Oracle Trace service routines. The files also provide constants that are needed when calling the service routines and when generating your own reports against formatted Oracle Rdb databases and RMS files. The definition files are:

Ada	EPC$DEFINITIONS.ADA
BASIC	EPC$DEFINITIONS.BAS
BLISS	EPC$DEFINITIONS.R32
C	EPC$DEFINITIONS.H
COBOL	EPC$DEFINITIONS.LIB
FORTRAN	EPC$DEFINITIONS.FOR
MACRO	EPC$DEFINITIONS.MAR
Pascal	EPC$DEFINITIONS.PAS
PL/I	EPC$DEFINITIONS.PLI

4.5 Using the Flags Argument to Instrument Efficiently

All of the Oracle Trace service routine calls contain an optional argument, called **flags**. This argument uses bit flags to convey special instructions to Oracle Trace. Do not confuse the service routine arguments called flags, with OpenVMS event flags. The flags argument is Oracle Trace specific. Judicious use of the Oracle Trace flags argument can ensure that collections have minimum impact on your application's performance.

Table 4–4 describes how each service routine uses the flags argument:

Table 4–4 Oracle Trace Flags Argument

Flag Argument	Associated Service Routines	Setting this bit . . .
EPC$M_BITMAP	EPC$INIT	specifies a bitmap rather than an array of long words for the event collection flags to determine if Oracle Trace will collect for each event. This argument allows applications using a language that supports bit testing to use less memory while not adversely affecting users of languages not supporting bit tests.
EPC$M_NOEF	All routines	eliminates clearing the OpenVMS event flag on service initiation and setting it on completion. This can reduce the overhead associated with these tasks, if your application does not need to wait on the OpenVMS event flag.
EPC$M_SYNCSTS	All routines except EPC$INIT	eliminates an AST routine call upon service completion. Oracle Trace calls the AST routine only if the Oracle Trace command does not complete synchronously (for example, if it had to perform some I/O). If your application has no need of being notified of a synchronous completion, you can eliminate the overhead associated with this task.
EPC$M_LONGREC	EPC$START_ EVENT(W), EPC$END_ EVENT(W), and EPC$EVENT(W)	indicates that the value specified by the user-defined-buffer argument is the address of a longword buffer, rather than a descriptor. The longword buffer contains the address of a descriptor followed by a pointer to the actual string. Set this bit when the user-defined buffer (for defining facility-specific items) contains more than 32K bytes.

Note

You must set the values for the EPC$M_xxx bits in the flags argument. Oracle Trace provides include files (see Section 4.4.3) for many languages containing the values of these bits. If no EPC$DEFINITIONS file exists for your language, you need to define the following constants in your instrumented code:

- EPC$M_SYNCSTS = 1
- EPC$M_NOEF = 2
- EPC$M_BITMAP = 4
- EPC$M_LONGREC = 8

4.6 Identifying Events and Data Items of Interest

To use Oracle Trace efficiently, you must be clear about your purpose and goals for using Oracle Trace. Clarity of purpose will help you identify the events and items you want to instrument for collection.

Typically, you should focus on business-level and functional-level events. An example of a business-level event might be individual transactions such as making, changing, canceling, and confirming reservations in an airline reservation application. An example of a functional-level event might be a routine that reads and writes information to and from a disk.

The programmer instrumenting the automated teller machine (ATM) sample application (EPC$EXAMPLES:EPC$ATM-SAMPLE.*) chose events based on how people used the program. In the sample, a duration event occurs when a customer checks the account balance or makes a withdrawal or deposit to an account.

The instrumentation of the ATM application is designed to gather information about the user interface. The goal is to be able to answer questions such as:

- How long does it take to complete a transaction?
- Do customers check their balances before or after every transaction?
- Could overdrafts be reduced or eliminated by displaying the balance on the withdrawal display?
- Are ATM machines used primarily for deposits or withdrawals?

Similar function level events which could be developed for the ATM application include:

- Which ATMs are used most often?
- What are the most common withdrawal amounts on each day of the week? $10 on Mondays? $25 on Thursdays? $100 on Fridays?
- Does anyone ever make deposits in the ATM machine in the shopping mall?

If the goal of instrumenting the ATM application is to improve the performance of the program, then a different set of events could be defined. The events based on functionality could be broken down into more specific events. For example, the withdrawal event consists of three main steps:

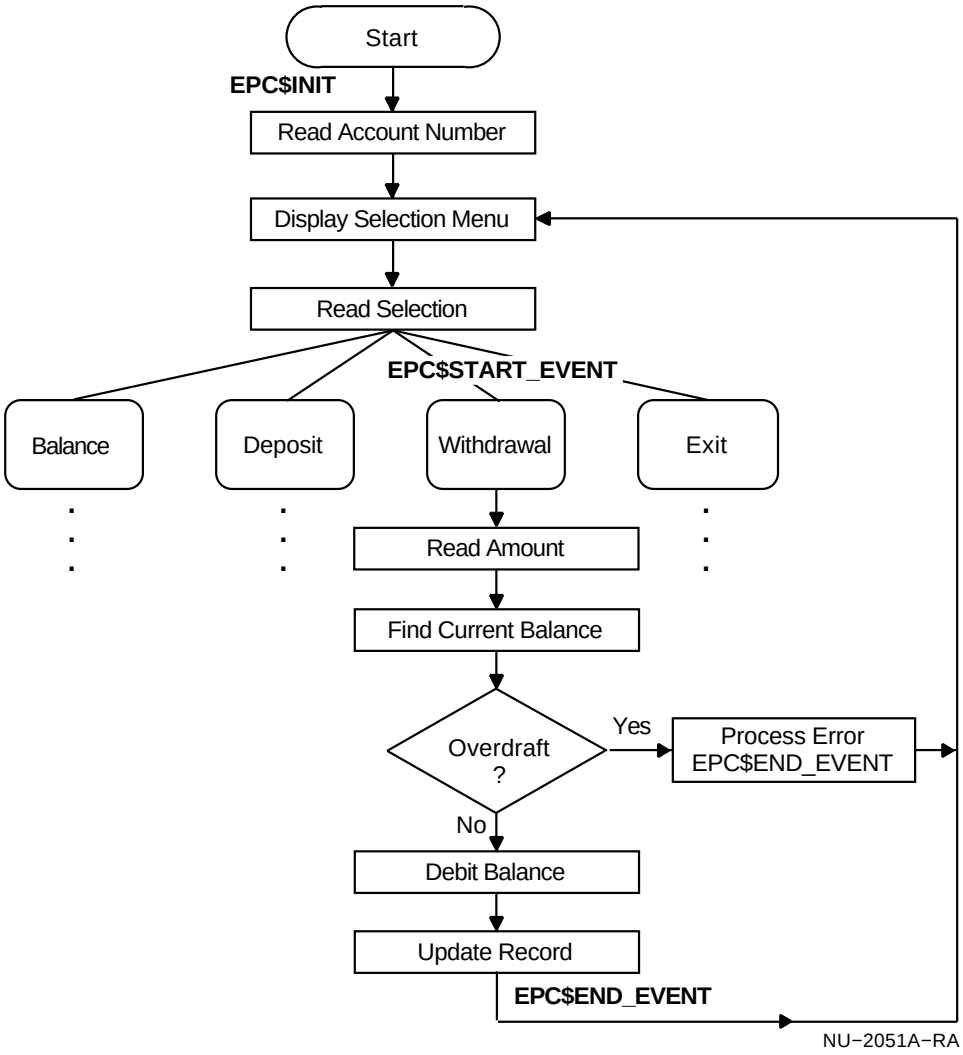
1. Get the withdrawal amount.
2. Subtract the amount from the customer's balance.
3. Update the database with the new balance.

You can further divide the withdrawal event into steps that detail the actual execution sequence of the program. In fact, the following list of events is similar to the pseudo-code and flowchart used to create the original, non-instrumented, application program:

1. Display the withdrawal screen.
2. Read the withdrawal amount.
3. Find the customer's current balance in the database.
4. Check for a possible overdraft.
5. Subtract the withdrawal amount from the customer's balance.
6. Lock and then update the customer's account record.

After you have determined what events you want to define for your application, you need to locate the events in a functional diagram of the application. This helps you to locate the actual location of the event in your source code. Figure 4–2 shows a flowchart of the sample ATM application with one event outlined in detail.

Figure 4–2 Flowchart of the Sample Application Showing One Event



4.7 Defining Data Structures for Your Application

After you have identified the events in your application you want to collect data for, you need to think about the kinds of items (data) you want to collect for these events. The kinds of items you plan to collect determine the data structures you need to define for your application.

Oracle Trace allows you to select three kinds of items as described in Table 4–5.

Table 4–5 Data Items

Item	Description
Standard resource utilization items	A set of data items that Oracle Trace collects by default. Table 4–6 describes these items. Although Oracle Trace collects them by default, they are optional.
Cross-facility item	A set of data items that relate events among applications. You can designate cross-facility items if you cannot change or pass data items using the inter-application interface.
Facility-specific items	A set of data items that you define for your application. You can assign names to these items to identify them in the Oracle Trace report.

Table 4–6 describes the standard resource utilization items collected by Oracle Trace.

Table 4–6 Resource Utilization Items

Item Name	Item ID Number	Description	Data Type	Usage
BIO	101	Number of buffered I/O operations	Longword	Counter
DIO	102	Number of direct I/O operations	Longword	Counter
PAGEFAULTS	103	Total number of hard and soft page faults	Longword	Counter
PAGEFAULT_IO	104	Number of hard page faults (that is, page faults to or from the disk)	Longword	Counter
CPU	105	Total amount of CPU time in tens of milliseconds	Longword	Counter
CURRENT_PRIO	106	Current priority of the process	Word	Level
VIRTUAL_SIZE	107	Number of virtual pages that are currently mapped for the process	Longword	Level
WS_SIZE	108	Current working set size of the process	Longword	Level
WS_PRIVATE	109	Number of pages in the working set that are private to the process	Longword	Level
WS_GLOBAL	110	Number of pages in the working set that are globally shared among processes on the system	Longword	Level

Note

Oracle Trace stamps each data collection record with the current date and time, process identification number (EPID), facility number, and event identifier. Therefore, the elapsed time information for the event, EPID, the facility that logged the event, and the event identifier are always collected.

Table 4–7 lists the 14 cross-facility items available to programmers who are instrumenting their application for Oracle Trace applications. Note that only these items can be used as cross-facility items; all other items are accessible only within the context of a single facility.

Table 4–7 Cross-Facility Items

Cross-Facility Item Name	Item Number	Data Type	Facility that Sets the Value
CROSS_FAC_1	115	Longword	ALL–IN–1 and other OA software products
CROSS_FAC_2	116	Longword	ACMS, DECintact, other TP monitor products, collected by Oracle Rdb, Oracle CODASYL DBMS
CROSS_FAC_3	117	Longword	RALLY and possibly other 4GL products, collected by Oracle Rdb
CROSS_FAC_4	118	Longword	Reserved
CROSS_FAC_5	119	Longword	Reserved
CROSS_FAC_6	120	Longword	Reserved
CROSS_FAC_7	121	Longword	Set by DEC ACMS, collected by Oracle Rdb, Oracle CODASYL DBMS
CROSS_FAC_8	122	Longword	Reserved
CROSS_FAC_9	123	Longword	Reserved
CROSS_FAC_10	124	Longword	Reserved
CROSS_FAC_11	125	Longword	Reserved
CROSS_FAC_12	126	Longword	Reserved
CROSS_FAC_13	127	Longword	Reserved
CROSS_FAC_14	128	Longword	General customer facility use, unrestricted

Refer to the documentation that accompanies the facility for which you are collecting data to learn how it uses the Oracle Trace cross-facility capability. Table 4–8 describes the Oracle Trace data structures and shows when you should use them in the facility you are instrumenting.

Table 4–8 Data Structures

Data Structure Name	Purpose	Use for Collecting Standard Resource Utilization Items?	Use for Collecting Facility-Specific Items?
Event collection flags list	Contains the events Oracle Trace collects data for	Yes	Yes
Item flags list	Contains items collected for each event	Optional	Yes
Use-defined buffer	Describes facility-specific items	No	Yes

Your facility must allocate virtual memory (either statically or dynamically) for each data structure it uses and pass the address of these structures in the call to the EPC$INIT service routine.

Note

If your application is in C or any other programming language that starts array offsets at zero, you must take care when you reference items in the event collection flags list. Oracle Trace event ID numbering begins at 1, whereas in C and some other languages, array element numbering begins at 0. Therefore, for these languages, you must adjust the array subscript when you test an event, as shown in Example 4–7.

The following sections describe the data structures and provide examples of how to define them.

4.7.1 Event Collection Flags List

The event collection flags list is a list of up to 128 longword Boolean values. As described in Table 4–2 and shown in Figure 4–1, Oracle Trace designates events for collection by setting bits in the event collection flags list. Applications can minimize overhead by verifying in the event collection flags list that collections are active for an event and by calling Oracle Trace event service routines only when collections are active for the event.

Although the event collection flags list is comprised of longword Boolean values, you can alternatively implement it as a bitmap of up to 128 bit elements. To implement the event collection flags list as a bitmap, you need to set the EPCM_BITMAP bit of the flag argument as described in Section 4.5. See the EPC$INIT routine description in Chapter 7 for a diagram of the event collection flags list.

Each element of the list corresponds to a particular event in your application. As described in Figure 4–1, your application can allocate the event collection flags list and pass Oracle Trace the address of the event collection flags list using the EPC$INIT routine.

Example 4–1 is an example from EPC$ATM-SAMPLE-EXTENDED.C of allocating the event collection flags list. Appendix A contains a complete version of the EPC$ATM-SAMPLE-EXTENDED facility.

Example 4–1 Allocating the Event Collection Flags List

```
  .
  .
  .
/* Note: I am using the max events and item flags.  The item flags needs to */
/* be max because we are collecting resource items, to use VM more     */
/* efficiently the event_flags array should only be MAX-MIN+1 long     */
static int s_event_flags[128];
static ITEM_FLAGS_T s_event_item_flags[128];
  .
  .
  .
```

Example 4–2 is an example from the EPC$ATM-SAMPLE-EXTENDED.C application of sending Oracle Trace the address of the event collection flags list.

Example 4–2 Sending Oracle Trace the Address of the Event Collection Flags List

```
/* Call EPC$INIT to register the ATM_SAMPLE facility with Oracle Trace*/
  cond_status = epc$init(0,      /* No VMS efn used      */
    FACILITY_NUMBER,   /* facility number REQUIRED     */
    &s_fac_ver,     /* Facility version REQUIRED    */
                        &s_reg_id,     /* registration ID      */
                        &s_event_flags,    /* Oracle Trace event flags     */
    &s_event_item_flags,/* Item flags buffer      */
    0,      /* No Status block needed     */
    0,      /* No AST       */
    0,      /* No AST parameter      */
    EPC$M_NOEF,     /* Don't bother to clear and    */
         /* set efn, I'm not using it    */
    MIN_EVENT,     /* Min. event for this faciity  */
    MAX_EVENT,     /* Max. event for this facility */
    MIN_ITEM,     /* Min. item for this facility  */
    MAX_ITEM     /* Max. item for this facility  */
    );
```

The Registrar sets event collection flags corresponding to events to be collected, based on the facility definition and selection. Your application then tests the longword (or bit) associated with each event before calling an Oracle Trace event service routine for that event.

4.7.2 Item Flags List

The item flags list contains items to collect for each event. It is a two-dimensional (128 by 128 element) bit array. See the EPC$INIT routine description in Chapter 7 for a diagram of the item flags list. Each bit in the list element corresponds to an item identifier. Oracle Trace designates bits 0 through 100 in the item flags list for facility-specific items and bits 101 through 128 as reserved for Oracle Trace.

If you plan to collect only standard resource utilization items, you do not need to use the item flags list because Oracle Trace collects these items by default for each event.

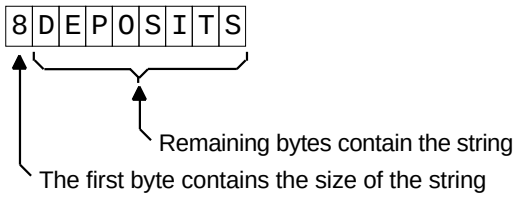
4.7.3 User-Defined Buffer

The **user-defined buffer** is a data structure that contains descriptions of facility-specific items. The user-defined-buffer argument of the Oracle Trace event service routines (see Table 4–9) specifies the address of the user-defined buffer. A facility uses the user-defined buffer to pass descriptions of facility-specific items to Oracle Trace. When a facility specifies the layout of this buffer, it is the same whether the facility collects one item or all items listed in the EVENT option in the facility definition for an event. Any extra items passed by the facility for an event are removed from the data collection files during the merge and format operations, so that only those items that were specified in the facility selection appear in the formatted database.

You must pass items to Oracle Trace in the user-defined buffer as counted strings. A **counted string** is a data structure in which a count of the number of characters in the string precedes the string. Oracle Trace supports three kinds of counted strings:

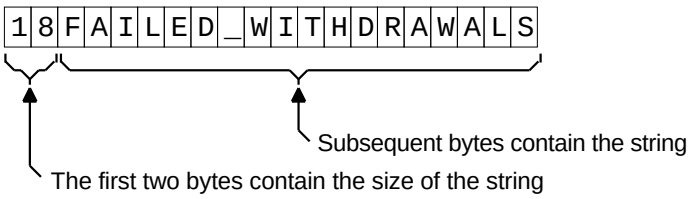
- ASCIC
- ASCIW
- FIXED_ASCIC

An ASCIC string is a variable length string of a predetermined size. The first byte contains the size of the string.



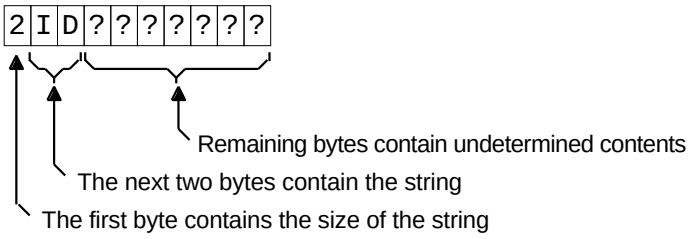
NU–2638A–RA

An ASCIW string is also variable length string of a predetermined size. The first two bytes contain the size of the string.



NU–2639A–RA

A FIXED_ASCIC string, unlike the ASCIC and ASCIW, is a fixed length string. However, like ASCIC and ASCIW, its first two bytes contain the size of the string. Most programmers use the FIXED_ASCIC data structure because the user-defined buffer passed to Oracle Trace does not need to be constructed for each event occurrence. However, records in the FIXED_ASCIC structure take up more space if your item is less than the maximum size, as shown below:



NU–2640A–RA

4.7.3.1 How to Define a User-Defined Buffer in C

Example 4–3 is an example from EPC$EXAMPLES:EPC$ATM-SAMPLE-EXTENDED.C showing how to define a user-defined buffer in C.

Example 4–3 C Example of a User-Defined Buffer

```
/*****************************************************************************/
main()
 {
   .
   .
   .
 char              *record_buffer;
```

(continued on next page)

Example 4–3 (Cont.) C Example of a User-Defined Buffer

```
.
.
.
        /*****************************************************************/
        /* Sample layout of user defined buffer                          */
        /* The user defined items are:                                   */
        /*    signin_id:          FIXED_ASCIC 9 characters               */
        /*    machine_number:     longword                               */
        /*    machine_location:   ASCIC                                  */
        /* Buffer looks like:                                            */
        /*  +---------------------------+                                */
        /*  |a|    b    | c  |d|     e...                                 */
        /*  +---------------------------+                                */
        /*  where:                                                       */
        /*    a = fixed_ascic length (1 byte)                            */
        /*    b = actual fixed_ascic data (9 bytes)                      */
        /*    c = longword (4 bytes)                                     */
        /*    d = ascic length (1 byte)                                  */
        /*    e = actual ascic data (c bytes)                            */
        /*****************************************************************/
        record_buffer = (char *) malloc(
               FIXED_ASCIC_LENGTH_BYTES + ACCOUNT_ID_LENGTH +
         LONGWORD_LENGTH_BYTES +
               ASCIC_LENGTH_BYTES + sizeof(atm_location));

        record_buffer[0] = ACCOUNT_ID_LENGTH;
        memcpy(&(record_buffer[FIXED_ASCIC_LENGTH_BYTES]),
              account_id, ACCOUNT_ID_LENGTH);

        record_buffer[FIXED_ASCIC_LENGTH_BYTES+ACCOUNT_ID_LENGTH] =
          (long) THIS_MACHINE;
```

(continued on next page)

Example 4–3 (Cont.) C Example of a User-Defined Buffer

```
        record_buffer[FIXED_ASCIC_LENGTH_BYTES+ACCOUNT_ID_LENGTH
               +LONGWORD_LENGTH_BYTES]=
          strlen(atm_location);
        memcpy
    (&(record_buffer[FIXED_ASCIC_LENGTH_BYTES+ACCOUNT_ID_LENGTH
                         +LONGWORD_LENGTH_BYTES+ASCIC_LENGTH_BYTES]);
                 atm_location, strlen(atm_location));
        /* length = signin id size (1byte) + signin id + machine no. +    */
        /*          machine loc size (1 byte) + machine loc str size      */
        record_desc.dsc$w_length = FIXED_ASCIC_LENGTH_BYTES+ACCOUNT_ID_LENGTH+
                                   LONGWORD_LENGTH_BYTES+ASCIC_LENGTH_BYTES+
       sizeof(atm_location);
        record_desc.dsc$b_class = DSC$K_CLASS_S;
        record_desc.dsc$a_pointer = record_buffer;
        cond_status = EPC$START_EVENT(0,  /* No efn used                  */
                        FACILITY_NUMBER,  /* Facility number REQUIRED     */
                        SIGNIN_EVENT,     /* Event ID REQUIRED            */
                        &si_event_handle, /* Event occurence handle REQ   */
                        0,                /* No Context, single threaded  */
                        &record_desc,     /* Data buffer                  */
                        0,                /* No status required           */
                        0,                /* No AST                       */
                        0,                /* No AST parameter             */
                        EPC$M_NOEF        /* Don't clear/set efn          */
                        );
      }
```

4.7.3.2 How to Define a User-Defined Buffer in COBOL

Example 4–4 is an example from EPC$EXAMPLES:EPC$ATM-SAMPLE-EXTENDED.COB showing how to define a user-defined buffer in COBOL.

Example 4–4 COBOL Example of a User-Defined Buffer

```
*
* This is the event record that will be passed to
* EPC$START/END-EVENTfor the SignIn event.  The facility
* definition in EPC$EXAMPLES:EPC$ATM-FAC-DEF-EXTENDED.COM
* defines the items as:
*
* ITEM  ACCOUNT_ID FIXED_ASCIC /SIZE=9 /ID=1 ...
* ITEM  MACHINE_NUMBER  LONGWORD /ID=2 ...
* ITEM  MACHINE_LOC     ASCIC /SIZE=40 /ID=3 ...
*
*
* Sample layout of the data buffer is
*           +----------------------------+
*           |a|    b    | c  |d|     e...
*           +----------------------------+
*           where:
*            Account_ID =
*               a = fixed_ascic length (1 byte)
*               b = actual fixed_ascic data (9 bytes)
*
*            Machine Number =
*               c = longword (4 bytes)
*
*            Machine Location =
*               d = ascic length (1 byte)
*               e = actual ascic data (c bytes)
*
* The above data types are represented in the SignIn_Record
* definition below. Note that the FIXED_ASCIC and ASCIC items
* are prefixed by a single byte that contains the length of
* the following character string. Since COBOL does not support
* a one byte computational field, we have to trick COBOL
* by assigning a hexadecimal value to a PIC X field.  In the
* example below, the 9 byte SignIn_ID (i.e., ACCOUNT_ID) is
* prefixed by the hex value "09"; the 40 byte
* Machine_Location is prefixed by the hex value
* "28" which is equivalent to 40 decimal.
*
* ASCIW uses a 2 byte count field which can be represented in
* COBOL with a PIC 9(4) COMP description.
*
* Dynamic memory allocation in COBOL is relatively difficult to do.
* One technique is to preallocate the maximum size buffer that
* would be used by an event and then move into the buffer the
* data as necessary.
*
```

(continued on next page)

Example 4–4 (Cont.) COBOL Example of a User-Defined Buffer

```
* A sample of this is the Signout_Record.
*
* ITEM  BANK_NAME           ASCIW (40 bytes max.)
* ITEM  BANK_HQ_LOCATION    ASCIW (40 bytes max.)
*
*
* Sample layout of the data buffer is
*           +---------------------------+
*           |a |   b...    |c |   d...   |
*           +---------------------------+
*           where:
*            Bank_Name =
*               a = asciw length (2 bytes)
*               b = actual asciw data (upto 40 bytes)
*
*            Machine Location =
*               c = asciw length (2 bytes)
*               d = actual asciw data (upto 40 bytes)
*
* The Signout_char array is 84 bytes total.
*
****************************************************************

01 SignIn_Record.
   02 SignIn_ID_Length  PIC X value X"09".
   02 SignIn_ID         PIC 9(9).
   02 Machine_Number    PIC 9(9) comp.
   02 Machine_Location_Length PIC X value X"10".
   02 Machine_Location  PIC X(16).

01 Dectrace_Data_Descriptor.
    02  Descriptor_Length   PIC 9(4) comp.
    02  Descriptor_Class    PIC 9(4) comp.
    02  Descriptor_Ptr      USAGE IS POINTER.

* This is the maximum size of the signout record
* (2 40-byte ASCIW items with 2 2-byte lengths each)
01 Signout_char PIC X(84).
   .
   .
   .
*
* Exit if the account id is zero.
*
* Otherwise, validate the account number and proceed to
* the main menu. If it isn't a valid number, display an
* error message and let the customer try again.
*
```

(continued on next page)

Example 4–4 (Cont.) COBOL Example of a User-Defined Buffer

```
IF Account-ID = 0
THEN GO TO Close-and-Exit-Program.

MOVE No-such-account TO error-message.

READ DATA-FILE KEY IS Account-ID
    INVALID KEY
            DISPLAY no-such-account AT LINE 20 COLUMN 1
                                    ERASE TO END OF LINE
            GO TO ACCEPT-ACCOUNT-ID.

MOVE Account-ID to SignIn_ID.
MOVE 2 to Machine_Number.
MOVE "Colorado Springs" to Machine_Location.
.
.
.
```

4.8 Using EPC$INIT to Register a Facility

The instrumentation of a program begins with an EPC$INIT call. This call should be done early in the execution of a program, because no event data can be collected from the application until the Oracle Trace Registrar process receives this message. As shown in Figure 4–1, The EPC$INIT call basically alerts Oracle Trace of the application's readiness and provides specific information about the events and items the application makes available for collection.

The EPC$INIT call contains the following information:

- Optional OpenVMS event flag (for synchronization)
- Unique facility ID number
- Version of the facility
- Optional facility-specific registration ID value to distinguish at collection time between images using the same facilities
- Address of the event collection flags list (optional)
- Address of the items flags list (optional)
- Optional arguments that describe characteristics of the event and item information.

Note

If you specify the **max_event** and **max_item** arguments to the EPC$INIT routine, the user-defined buffer that you use to define facility-specific items must be large enough to support that maximum number of events and items, regardless of whether all the events and items are actually used. See Section 4.7.3 for more information about the user-defined buffer.

Example 4–5 shows the implementation of the EPC$INIT call in the COBOL version of the ATM sample application. The COBOL source file of the ATM application is available in EPC$EXAMPLES:EPC$ATM-SAMPLE.COB. Additional versions of the ATM program, written in C, FORTRAN and Pascal, are also available in the EPC$EXAMPLES directory.

Example 4–5 Instrumentation of EPC$INIT in the COBOL ATM Application

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ATM-SAMPLE.
AUTHOR. Oracle Corporation
*++
   .
   .
   .
*
*The following variables are used for the Oracle Trace service routines.
*

77 Max_events           PIC 9(9) comp value 128.
77 Facility_number      PIC 9(9) comp value 4094.
77 Errors_event_id      PIC 9(9) comp value 1.
77 Balance_event_id     PIC 9(9) comp value 2.
77 Deposit_event_id     PIC 9(9) comp value 3.
77 Withdrawal_event_id  PIC 9(9) comp value 4.

01 Facility_version     PIC X(4) value "V1.0".
01 Event_flags_list.
02 Event_flags          PIC 9(9) comp occurs 128 times.
01 Event_handle         PIC 9(9) comp.
01 Cond_status          PIC 9(9) comp.
01 Registration_id      PIC X(15) value "ATM APPLICATION".
```

(continued on next page)

Example 4–5 (Cont.) Instrumentation of EPC$INIT in the COBOL ATM Application

```
   .
   .
   .
PROCEDURE DIVISION.

INITIALIZE-DECTRACE.
*
* Perform facility initialization tasks, including the EPC$INIT call.
*
    CALL "EPC$INIT" USING BY VALUE 0,
                          BY VALUE FACILITY_NUMBER,
                          BY DESCRIPTOR FACILITY_VERSION,
                          BY DESCRIPTOR REGISTRATION_ID,
                          BY REFERENCE EVENT_FLAGS_LIST
                    GIVING COND_STATUS.
```

See Chapter 7 for a detailed description of the format of the EPC$INIT call.

4.8.1 Waiting for the Registrar to Respond

Two timers control the length of time a user or process waits while trying to communicate with the Oracle Trace Registrar process. You can change the default timeout values by setting the logicals EPC$BE_REGTIMEOUT and EPC$FE_REGTIMEOUT.

You can define the EPC$BE_REGTIMEOUT logical to the maximum number of seconds that you want a process to wait for the Registrar during a call to EPC$INIT. If the Registrar process does not respond within that time, the condition EPC$_REGTIMOUT is returned through the EPC$INIT call and the process is not registered with Oracle Trace. If you want to guarantee process registration, your application code must check for this status and resubmit the EPC$INIT call if necessary.

Note that the typical Registrar response rate is less than one second, and the only time the response should degrade is during collection activation. To determine the proper timer value, use the SHOW HISTORY /FORMAT=ALL command after scheduling a cluster-wide collection. Examining this history report will help you to determine the time required to activate a large collection. To ensure that all processes are available for collection, set the EPC$BE_REGTIMEOUT logical greater than the longest process activation time.

Define the EPC$FE_REGTIMEOUT logical to the maximum number of seconds that a user should wait for the Registrar to respond to a SCHEDULE or CANCEL COLLECTION, or CREATE DEFINITION command. If the Registrar process does not respond within that time period, an error occurs and the user must resubmit the command.

The default for these timers is 90 seconds. You can redefine them to any whole number of seconds greater than or equal to 1.

4.8.2 Waiting for EPC$INIT to Complete

When the EPC$INIT call registers a facility with the Oracle Trace Registrar process, the Registrar records the registration and compares it to the criteria associated with any active collections. Although this processing completes quickly, the application program does continue to execute, and some events might not get collected.

Oracle Trace does not cause the application to stop and wait for the Registrar process to completely finish processing. As a result, some events immediately following the EPC$INIT call might not be collected even though they are designated for collection. However, the application programmer can make the program wait until data collection is enabled for the process, by passing the address of the event collection flags list in the EPC$INIT routine. The event collection flags list is filled asynchronously when the process receives the "Start collecting" message from the Registrar process.

If you do not want to miss any events, poll the event collection flags list and wait until they get written before continuing program execution. Note that you should set a timeout or retry limit, because if no active collections are defined to collect data from that image, or if the Oracle Trace Registrar process is not active, the program will wait indefinitely.

Example 4–6 shows a C code segment from the Oracle Trace IVP program that waits until the process has registered before continuing program execution.

Example 4–6 Code to Guarantee Collection of First Events

```
/* Wait for the collection to be activated for this     */
/* process. Keep checking the 1st event collection flag */
/* to see if it's set. */
 while((loop_count++ < RETRY_TIMES) && (evntflgs[0] != 1))
  {
    status = lib$wait(&delay_time);
    if (status != SS$_NORMAL)
      {
        printf("Error returned from synchronization after
        call to EPC$INIT\n");
        return (status);
      }
  }
```

4.9 Checklist for Instrumenting Events

You must perform these tasks to instrument each event for collection:

1. Perform a Boolean test of the element corresponding to the event in the collection flags list prior to calling each event service routine to ensure that Oracle Trace is collecting for the event, as described in Section 4.9.1. Note that this test is optional, but strongly recommended.
2. Implement the calls to the appropriate event service routines for each event, as described in Section 4.9.2.
3. Designate any non Oracle Trace cross-facility items you want to collect, as described in Section 4.9.3.
4. Specify the events in the facility definition using the /EVENT qualifier to the CREATE DEFINITION command, as described in Chapter 5.
5. Specify any facility-specific items associated with events in your application. Also specify the /ITEMS qualifier to the EVENT option of the CREATE DEFINITION command as described in Chapter 5.

Note

Oracle Trace uses the facility definition to map the types of collected data to the event for both collection and formatting operations. Therefore, every event in the instrumented application must have a corresponding description in the facility definition. To avoid problems, be sure to perform tasks 4 and 5.

Appendix A contains an example of an instrumented facility called EPC$ATM-SAMPLE-EXTENDED.C that the following sections reference.

4.9.1 Verifying That Oracle Trace Is Collecting an Event

As described in Figure 4–1, instrumenting the code with the appropriate event service routines does not guarantee that Oracle Trace will log it. For example, Oracle Trace will not collect for an event if the collection for the facility is not currently active, or if the particular event is not part of the class of events being collected.

Therefore, to avoid unnecessary calls to Oracle Trace, your application should determine if Oracle Trace is collecting for the event, before executing each EPC$START_EVENT(W), EPC$END_EVENT(W), and EPC$EVENT(W) routine call. This makes your application more efficient because this test reduces the overhead of Oracle Trace routine calls on your system when the event does not need to be collected.

Applications can test whether or not Oracle Trace is collecting for an event by determining if the longword (or bit) associated with the event is set (has a value of 1) in the event collection flags list. A set element indicates that data should be collected for the associated event.

The following source code defines events for the EPC$ATM-SAMPLE-EXTENDED facility. See Appendix A for all of the source code of this instrumented facility.

```
#define BALANCE_EVENT  2  /* value for the balance event */
#define DEPOSIT_EVENT  3  /* value for the deposit event */
#define WITHDRAW_EVENT  4  /* value for the withdraw event */
#define SIGNIN_EVENT  5  /* value for the signin event */
```

If data is being collected for the Balance event, Oracle Trace sets a flag in the event collection flags list. For example:

```
static int s_event_flags[128];        /* Event Flags list */
```

Note

If your application is in C or any other programming language that starts array offsets at zero, you must take care when you reference items in the event collection flags list. Oracle Trace event ID numbering begins at 1, whereas in C and some other languages, array element numbering begins at 0. Therefore, for these languages, you must adjust the array subscript when you test an event, as shown in Example 4–7.

The EPC$ATM-SAMPLE-EXTENDED.C facility uses the event ID as an offset to test the appropriate element of the event collection flags list before calling EPC$EVENT(W), EPC$START_EVENT(W) or EPC$END_EVENT(W), as shown for C example in Example 4–7.

Example 4–7 Testing for Collections on an Event Showing Array Adjustment

```
          /* balance event; check event first */
                if (s_event_flags[BALANCE_EVENT - 1] != FALSE)
    {
.
.
.
                    epc$start_event(0,  /* No efn used     */
.
.
.
    }
```

4.9.2 Calling Event Service Routines

Table 4–9 lists the Oracle Trace event service routines.

Table 4–9 Oracle Trace Event Service Routines

Routine Name[1]	Use for a . . .
EPC$START_EVENT	duration event to record its start to the data collection file.
EPC$START_EVENTW	duration event to records its start to the data collection file and wait for processing to complete before returning.
EPC$END_EVENT	duration event to record its end to the data collection file.
EPC$END_EVENTW	duration event to record its end to the data collection file and wait for processing to complete before returning.
EPC$EVENT	point event to record its occurrence to the data collection file.
EPC$EVENTW	point event to record its occurrence to the data collection file and wait for it to complete before returning

[1]The service routines are prefaced with "EPC" because that is the registered facility name of Oracle Trace.

4.9.2.1 Designating Events to Collect Standard Resource Items

Oracle Trace can collect resource utilization items for each event on behalf of the facility by default. To specify resource items for collection for an event, list them in the EVENT option of the facility definition. However, if you collect any resource items for a duration event, you must specify them on both the /START_EVENT and /END_EVENT qualifiers of the EVENT option.

An item identifier set in the item flags list indicates that the corresponding item is collected for a particular event. The event identifier is used as the offset into the item flags list to obtain the corresponding item flags for an event.

Figure 4–2 illustrates the ATM WITHDRAWAL event. Instrumenting the event consists of testing to determine if Oracle Trace is collecting for the event, and adding an EPC$START_EVENT call to the beginning of the function, and an EPC$END_EVENT call to the end of the function. Oracle Trace collects resource items on behalf of this event. The EPC$START_EVENT and EPC$END_EVENT calls in Figure 4–2 do not specify a user-defined buffer, therefore, Oracle Trace does not collect any other items.

Example 4–8 shows the COBOL code required to implement the WITHDRAWAL event.

Example 4–8 Instrumentation of the WITHDRAWAL Event in the ATM Application

```
* The Withdrawal transaction begins here.
*
START-WITHDRAWAL.
*
* Start of event4: WITHDRAW. For efficiency, test to see if the event is to
* be collected prior to calling EPC$START_EVENTW.
*
    IF EVENT_FLAGS(WITHDRAW_EVENT) = 1
    THEN
      CALL "EPC$START_EVENTW" USING BY VALUE 0,
                                    BY VALUE FACILITY_NUMBER,
                                    BY VALUE WITHDRAW_EVENT,
                                    BY REFERENCE EVENT_HANDLE
                              GIVING COND_STATUS
    END-IF.
*
* Clear the lower portion of the screen and display a withdrawal form.
*
    DISPLAY Withdrawal_heading reversed AT LINE 5 COLUMN 32
                 ERASE TO END OF SCREEN.

    DISPLAY "Amount of withdrawal:" AT LINE 12 COLUMN 15.

    DISPLAY withdrawal-instruction AT LINE 22 COLUMN 1.

PROMPT-FOR-WITHDRAWAL.
*
* Clear the message line and then prompt for the user's input.
*
    DISPLAY SPACE       AT LINE 20 COLUMN 1
                        ERASE TO END OF LINE.

    ACCEPT transaction-amount   BOLD
                                PROTECTED
                                WITH CONVERSION
                                FROM LINE 12 COLUMN 40
                                ERASE TO END OF LINE.
```

(continued on next page)

Example 4–8 (Cont.) Instrumentation of the WITHDRAWAL Event in the ATM Application

```
*
* Find the customer's record and make the withdrawal.
*
    MOVE transaction-failed TO error-message.
    READ DATA-FILE KEY IS Account-ID
        INVALID KEY
                PERFORM Invalid-input THROUGH End-Invalid-Input
                GO TO END-WITHDRAWAL.
    SUBTRACT Transaction-Amount FROM Balance GIVING Balance.
*
* If the withdrawal is larger than the current balance,
* issue an error message.
* Otherwise, update the file.
*
    IF Balance < 0
    THEN MOVE overdraft TO error-message
         PERFORM Invalid-input THROUGH End-Invalid-Input
    ELSE REWRITE Account-record
            INVALID KEY
                PERFORM Invalid-input THROUGH End-Invalid-Input
                GO TO END-WITHDRAWAL
    END-IF.
*
* End of event4: WITHDRAW. For efficiency, test to see if the event
* is to be collected prior to calling EPC$END_EVENTW.
*
    IF EVENT_FLAGS(WITHDRAW_EVENT) = 1
    THEN
      CALL "EPC$END_EVENTW" USING BY VALUE 0,
                                  BY VALUE FACILITY_NUMBER,
                                  BY VALUE WITHDRAW_EVENT,
                                  BY REFERENCE EVENT_HANDLE
                              GIVING COND_STATUS
    END-IF.
END-WITHDRAWAL.
    EXIT.
```

4.9.2.2 Defining Events in Both the Application and the Facility Definition

If you want to collect data for events you define in your application, you must provide corresponding definitions of these events in the facility definition, as described in Chapter 5. You can use a facility definition to tailor a collection. For example, you can limit collections to certain groups of events and data items using the CLASS option to the CREATE DEFINITION command to create collection classes, as described in Section 5.1.2. Example 4–9 shows how you might define events in your COBOL application.

Example 4–9 Example of Defining Events in a COBOL Application

```
 *
 *The following variables are used for the Oracle Trace service routines.
 *

77 Max_events           PIC 9(9) comp value 128.
77 Facility_number      PIC 9(9) comp value 4094.
77 Error_display        PIC 9(9) comp value 1.
77 Balance_event        PIC 9(9) comp value 2.
77 Deposit_event        PIC 9(9) comp value 3.
77 Withdraw_event       PIC 9(9) comp value 4.

01 Facility_version     PIC X(4) value "V1.0".
01 Event_flags_LIST.
```

Example 4–10 shows how you might define the events in the facility definition for your application.

Example 4–10 Example of Defining Events in a Facility Definition

```
!
! COBOL ATM Facility Definition (Three Events)
!
CREATE DEFINITION COBOL-ATM 4094 /VERSION="V1.0"-
    /EVENTS=(Error_display,Balance_event,Deposit_event,Withdraw_event)
```

These three events are defined as duration events and have *only* the standard resource utilization items collected for them by Oracle Trace. Every event type in the instrumented application must have a corresponding description in the facility definition because Oracle Trace uses the facility definition to map the types of collected data to the event for both collection and formatting

operations. In this example, Balance_event has an event ID of 2, Deposit_event has an event ID of 3, and so forth.

4.9.2.3 Defining Facility-Specific Items in Both the Application and the Facility Definition

In the following example, the EPC$ATM-SAMPLE-EXTENDED facility uses a record buffer with fields for each facility-specific item:

```
typedef  struct     record_buffer_s
    {
 SIGNIN_ID_T    signin_id;      /* fixed ascic     */
 short        machine_number;      /* word       */
 char           machine_location[MACHINE_LOC_LENGTH+
                       ASCIC_LENGTH_BYTES]; /* ascic */
   } RECORD_BUFFER_T;
```

An application must pass a user-defined buffer variable by descriptor. Thus the EPC$ATM-SAMPLE-EXTENDED facility contains:

```
#include <descrip.h>            /* VMS Descriptors */
        .
        .
        .
    struct     dsc$descriptor_s record_desc;
     /* Descriptor for record buffer */

            .
            .
            .
          record_desc.dsc$w_length = FIXED_ASCIC_LENGTH_BYTES+
                                ACCOUNT_ID_LENGTH+LONGWORD_LENGTH_BYTES+
                                ASCIC_LENGTH_BYTES+sizeof(atm_location);
   record_desc.dsc$b_class = DSC$K_CLASS_S;
   record_desc.dsc$a_pointer = record_buffer;
```

The facility writes the data into the record buffer and calls the appropriate Oracle Trace service routine to execute the Signin event, passing the address of the descriptor. For example:

```
            record_buffer = (char *) malloc(
ASCIW_LENGTH_BYTES +
sizeof(bank_name)  +
ASCIW_LENGTH_BYTES +
sizeof(bank_location));

     record_buffer[0] = sizeof(bank_name);
     record_buffer[1] = 0;
     memcpy(&(record_buffer[ASCIW_LENGTH_BYTES]),
    bank_name, sizeof(bank_name));
     record_buffer[ASCIW_LENGTH_BYTES+sizeof(bank_name)] =
sizeof(bank_location);
     record_buffer[ASCIW_LENGTH_BYTES+sizeof(bank_name)+1] =
0;
     memcpy(&(record_buffer[ASCIW_LENGTH_BYTES +
       sizeof(bank_name)  +
       ASCIW_LENGTH_BYTES]),
    bank_location, strlen(bank_location));
     record_desc.dsc$w_length = ASCIW_LENGTH_BYTES +
    sizeof(bank_name)  +
    ASCIW_LENGTH_BYTES +
    sizeof(bank_location);
     record_desc.dsc$b_class = DSC$K_CLASS_S;
     record_desc.dsc$a_pointer = record_buffer;
                  cond_status = epc$start_event(0,
  . . .
      &record_desc, /* Data buffer     */
  . . .
      );
     }
```

See Appendix A for the complete version of this sample facility.

You might specify a facility definition for this event and items as follows:

```
!
CREATE DEFINITION ATM_SAMPLE 4094 /REPLACE /VERSION="V1.2-0" /OPTIONS
!
! Items specific to ATM_SAMPLE facility version V1.1-0
!
! We will always pass a 10 byte buffer to Oracle Trace, 1 for the length of
! the data and the other 9 bytes = the data
ITEM  ACCOUNT_ID FIXED_ASCIC /SIZE=9 /ID=1 /REPORT_HEADER="Account Number" -
        /CHARACTERISTICS=PRINTABLE /REPORT_WIDTH=9 /USAGE_TYPE=TEXT

ITEM  MACHINE_NUMBER    LONGWORD /ID=2 /REPORT_HEADER="Machine Number" -
        /CHARACTERISTICS=PRINTABLE /REPORT_WIDTH=5 /USAGE_TYPE=LEVEL

! ASCIC may be upto 255 characters however we force max to be 40 here
ITEM  MACHINE_LOC       ASCIC /SIZE=40 /ID=3 /REPORT_HEADER="ATM Location" -
        /CHARACTERISTICS=PRINTABLE /REPORT_WIDTH=20 /USAGE_TYPE=TEXT
!
ITEM  BANK_NAME ASCIW /SIZE=40 /ID=4 /REPORT_HEADER="Bank Name" -
        /CHARACTERISTICS=PRINTABLE /REPORT_WIDTH=20 /USAGE_TYPE=TEXT
!
ITEM  BANK_LOCATION ASCIW /SIZE=40 /ID=5 /REPORT_HEADER="Bank Location" -
        /CHARACTERISTICS=PRINTABLE /REPORT_WIDTH=20 /USAGE_TYPE=TEXT

GROUP ACCOUNT_ITEMS /ITEMS=(ACCOUNT_ID,MACHINE_NUMBER,MACHINE_LOC,BANK_NAME,-
                            BANK_LOCATION)
!
    . . .
!
EVENT SIGNIN_EVENT /ID=5 /REPORT_HEADER="Signins" -
        /ITEMS=(ACCOUNT_ITEMS,RESOURCE_ITEMS, CROSS_FAC_14) -
        /START_EVENT=(ACCOUNT_ID, MACHINE_NUMBER, MACHINE_LOC, RESOURCE_ITEMS,
        /END_EVENT=(BANK_NAME, BANK_LOCATION, RESOURCE_ITEMS, CROSS_FAC_14)
!
```

This facility definition specifies that the Signin event collects three facility-specific items (ACCOUNT_ITEMS such as ACCOUNT_ID, MACHINE_ NUMBER, MACHINE_NUMBER, MACHINE_LOC, BANK_NAME, and BANK_LOCATION), along with the resource utilization items and CROSS_ FAC_14.

Oracle Trace provides a number of options for associating facility-specific items with events, grouping them, and organizing them into classes. See Section 5.6 for more information about the facility-specific options.

4.9.2.4 Collecting Facility-Specific Items More Efficiently

As previously stated, not all of the items associated with an event are necessarily being collected at any given time. A facility selection can specify a class that collects only a subset of the items in a facility definition. For example, the DB facility definition might contain the following CLASS options where WORKLOAD collects only one facility-specific item:

```
!
! DB Facility Definition (Two Items, One Event, and Two Classes)
!
CREATE DEFINITION DB 2048 /VERSION="V1.0-0" /OPTIONS
!
ITEM TRANSACTION_ID LONGWORD /ID=1 /REPORT_HEADER="Transaction ID"-
                    /USAGE_TYPE=LEVEL /CHARACTERISTICS=PRINTABLE
!
ITEM PERF_ITEM LONGWORD /ID=2 /REPORT_HEADER="Perf item"-
               /USAGE_TYPE=COUNTER /CHARACTERISTICS=PRINTABLE
!
EVENT TRANSACTION /ID=1 /REPORT_HEADER="Transaction"-
    /ITEMS=(TRANSACTION_ID, PERF_ITEM, RESOURCE_ITEMS)-
!
CLASS WORKLOAD TRANSACTION /ITEMS=(TRANSACTION_ID)
!
CLASS PERFORMANCE TRANSACTION /ITEMS=*
!
DEFAULT_CLASS PERFORMANCE
```

Thus, for efficiency, a facility can check each item and pass dummy data for the items not collected. Oracle Trace provides a list of items to be collected in the item flags list. Like the event collection flags list, the address of the item flags structure is one of the parameters the application passes to Oracle Trace using EPC$INIT:

```
typedef struct item_flags_s
  {
    int db_trans_id_item : 1;
    int db_perf_item : 1;
    int offset1 : 30;
    int offset2 : 32;
    int offset3 : 32;
    int offset4 : 32;
  } ITEM_FLAGS_T;

static ITEM_FLAGS_T s_event_item_flags[128];
```

Using the event ID as an offset, the facility can optionally test the appropriate bits in the items flags structure to test the flag for each facility-specific item. If the flag is set, the facility writes the data into the record buffer. For example:

```
if (s_event_flags[TRANS_EVENT_ID-1] != FALSE)
  {
            .
            .
            .
    /* set up descriptor */
           .
           .
           .
  if (s_event_item_flags[TRANS_EVENT_ID-1].db_trans_id_item != FALSE)
     {
     record_buffer.transaction_id =  . . .  ;
     }
  if (s_event_item_flags[TRANS_EVENT_ID-1].db_perf_item != FALSE)
     {
     record_buffer.perf_item =  . . .  ;
     }
           .
           .
           .
  status = epc$event( . . .  &record_desc . . .  )
           .
           .
           .
  }
```

Another optimization involves using classes more efficiently. When a class is declared, it sets the appropriate bits in the EVENTFLG and ITEMFLG structures of an instrumented facility. Typical instrumentations only check the EVENTFLG for an event and then build a record buffer that contains the ALL class of items for that event. Oracle Trace provides an optimization which can greatly reduce the size of the resulting data file. If a class specification for an event contains only Oracle Trace resource utilization items and cross-facility items, or a subset thereof, then none of the extraneous facility-specific items will be logged to the data file. If it is possible to create meaningful output using only these items, Oracle Corporation strongly recommends using this approach to greatly reduce overhead and disk space usage.

4.9.3 Relating Events Among Applications

Programmers often relate events among two or more facilities to gain an understanding of the full context in which the events occurred. There are two ways that programmers can relate events among applications. The most common way is to explicitly pass data items through the application programming interface (API). When instrumenting an application that relates events in this way, you can ensure that Oracle Trace collects these items by designating them in the user-defined buffer passed in the Oracle Trace service routine calls.

When facilities cannot change the existing application programming interface, or cannot pass data items through it, as is the case with SQL (DML) Oracle Trace provides the ability to relate these events using the cross-facility feature. You must perform two steps to relate events using Oracle Trace cross-facility feature:

1. Assign the appropriate values to cross-facility items using Oracle Trace cross-facility service routines described in Table 4–10.
2. Define cross-facility items in the facility definition as described in Chapter 5.

Consider this example of the cross-facility feature. Within Oracle Rdb, the Transaction Sequence Number (TSN) item relates the events TRANSACTION and REQUEST_ACTUAL. However, these events and items are only accessible from within Oracle Rdb. Instrumenting using the cross-facility feature, allows other facilities to relate their events to the Oracle Rdb events. For example, the CROSS_FAC_7 item relates the ACMS PROCEDURE_CALL event to the Oracle Rdb TRANSACTION event. As a result, you can look at DEC ACMS procedure call events with the same CROSS_FAC_7 value as that found in the Oracle Rdb transaction event to learn about all of the Oracle Rdb transaction events that occurred within the same PROCEDURE_CALL.

See Table 5–3 for a list of the 14 cross-facility items available to instrumenters. Note, you can use only those cross-facility items designated in Table 5–3; all other items are accessible only within the context of a single facility.

Table 4–10 describes the Oracle Trace service routines for instrumenting cross-facility items.

Table 4–10 Oracle Trace Cross-Facility Service Routines

Routine Name[1]	Use to
EPC$SET_CF_ITEMS	Assign values to all cross-facility items.
EPC$SET_CF_VALUE	Assign a values to one cross-facility item.
EPC$GET_CF_ITEMS	Retrieve all cross-facility items into a buffer. This routine applies to client/server and multithreaded applications and depends on the EPC$SET_ CONTEXT routine to change contexts for cross-facility items.

[1]The service routines are prefaced with "EPC" because that is the registered facility name of Oracle Trace.

Example 4–11 is an example from EPC$EXAMPLES:EPC$ATM-SAMPLE-EXTENDED.C showing how to define a cross-facility item in C.

Example 4–11 Instrumenting a Cross-Facility Item in C

```
/*
* The following constants are used for the Oracle Trace service routines
*                                                                      */
#define CROSS_FAC_14            128

   .
   .
   .

/*********************************************************************/
main()
 {
 int               account_id_number;
 int               *cf_14_loc;


   .
   .
   .
  /*********************************************************************/
  /* The design of this sample can assume that signin_event will      */
  /* always be logged if any ATM Oracle Trace collection is active.   */
  /* To guarantee this, the signin_event must be part of ALL classes  */
  /* of events in the ATM facility definition.  If we are assured of  */
  /* this then only set a cf_item if a collection is active.          */
  /*********************************************************************/

      /* By setting the CF item, it can be logged by Rdb as well       */
      cond_status = epc$set_cf_value(
                        0,              /* No efn used                */
                        CROSS_FAC_14,   /* define CF 14               */
                        SIGNIN_EVENT,   /* set its value to the act id */
                        0,              /* Not on a thread            */
                        &cf_14_loc,     /* the location of the cf item */
                        EPC$M_NOEF);    /* Don't clear/set efn        */
```

EPC$EXAMPLES:EPC$ATM-SAMPLE-EXTENDED.C avoids the overhead of calling EPC$SET_CF_VALUE more than once by getting the address of the CROSS_FAC_14 item, and saving it in CF_14_loc. On subsequent event calls, the example code writes the value it wants to record in that location, as shown in Example 4–12. In this example, it writes a value (event_id) corresponding to each event.

Example 4–12 Short-Cut for Setting a Cross-Facility Item Value in C

```
case 1:
            /*  withdrawal; first check if event should be collected  */
            if (s_event_flags[WITHDRAW_EVENT - 1] != FALSE)
              {
                /* change the cf_14 value to be logged by Trace       */
                /* to signify the Rdb events occurred during an ATM   */
                /* WITHDRAWAL event.  This is cheaper than calling    */
                /* EPC$SET_CF_VALUE.                                  */
                *cf_14_loc = WITHDRAW_EVENT;

                EPC$START_EVENT(0,               /* No efn used       */
                                FACILITY_NUMBER, /* facility number   */
                                WITHDRAW_EVENT,  /* Event ID          */
                                &event_handle,   /* Event handle      */
                                0,               /* No Context,       */
                                                 /* single threaded   */
                                0,               /* No user Data buffer*/
                                0,               /* No status required*/
                                0,               /* No AST            */
                                0,               /* No AST parameter  */

                    EPC$M_NOEF                 /* Don't clear/set efn */
                                );
              };
            withdrawal(account_id_number);
            if (s_event_flags[WITHDRAW_EVENT - 1] != FALSE)
              {
                EPC$END_EVENT(0,                 /* No efn used       */
                              FACILITY_NUMBER,   /* Facility Number   */
                              WITHDRAW_EVENT,    /* Event ID          */
                              &event_handle,     /* Distinguish this  */
                                                 /* occurrence with   */
                                                 /* the above start   */
                              0,                 /* No Context,       */
                                                 /* single threaded   */
                              0,                 /* No user Data buffer*/
                              0,                 /* No status required*/
                              0,                 /* No AST            */
                              0,                 /* No AST parameter  */
                              EPC$M_NOEF         /* Don't clear/set efn*/
                              );
              };
            break;
```

Example 4–13 shows a sample C code segment that sets the value of CROSS_FAC_2. Note that like the Oracle Trace resource utilization items, you must collect these cross-facility items on both the start and end of duration events.

Example 4–13 Cross-Facility Item

```
#include <epc$definitions.h>;   /* Oracle Trace definitions */
int *cross_fac2_addr;
int flags;
int procedure_index_value;
        .
        .
        .
/*********************************************************************
** Set the value of CROSS_FAC_2, if it's the first time we set it,
** call EPC$SET_CF_VALUE to set it, otherwise set the value directly.
** We always set the value of a cross-facility item, even if we're
** not currently collecting data, in case another facility is collecting
** and relies on the context that we are providing.
*********************************************************************/

if (first_time == TRUE)
  {
    flags = EPC$M_NOEF;
    status = epc$set_cf_value (0,
                               EPC$K_CROSS_FAC_2,
                               procedure_index_value,
                               0,
                               &cross_fac2_addr,
                               flags);
    first_time = FALSE;
  }
else
  {
    *cross_fac2_addr = procedure_index_value;
  }
        .
        .
        .
status = EPC$START_EVENT (0,
                          fac_number,
                          procedure_call_event_id,
        .
        .
        .
```

4.10 Instrumenting a Multithreaded Facility

You instrument a multithreaded application in the same way you instrument an application that is not multithreaded, as described in the previous sections of this chapter. However, Oracle Trace provides two service routines for collecting resource utilization and cross-facility items on a per-thread basis in a multithreaded application. Table 4–11 describes these routines.

Table 4–11 Oracle Trace Service Routines for Multithreaded Applications

Routine Name[1]	Use to . . .
EPC$SET_CONTEXT	denote the start of a new thread.
EPC$DELETE_CONTEXT	denote the end of of a thread.

[1]The service routines are prefaced with "EPC" because that is the registered facility name of Oracle Trace.

This section contains an example of the code modifications required for an event called TRANSACTION for the multithreaded facility NEW_TOOL. The following examples contain code segments written in C from an application composed of four routines:

- NEWT$$INITIALIZE — Represents a place in the source code for the facility code where the first code execution takes place. This example does not include this routine because the EPC$INIT routine is the same in both multithreaded and non-multithreaded applications. See Section 4.8 for an example.
- NEWT$$THREAD_START_RESUME — Represents a place in the code where new threads begin code execution or existing threads resume code execution, as shown in Section 4.10.1.
- NEWT$$THREAD_DELETE—Represents a place in the code where threads have completed code execution, as shown in Section 4.10.2.
- NEWT$$TRANSACTION — Represents a place in the code that comprises a transaction event. This example does not include this routine because instrumenting events is the same for both multithreaded and non-multithreaded applications. See Section 4.9 for an example.

The following sections describe the steps required to instrument a thread in the multithreaded application NEW_TOOL.

4.10.1 Setting Thread Context

The application issues a call to the EPC$SET_CONTEXT service routine where the thread resumes execution or immediately following the place where the thread block is allocated for a new thread. This call tells Oracle Trace to keep some new context (resource utilization and cross-facility items) for this new thread and also passes back a new context variable to the NEW_TOOL facility. When a thread is resuming execution, this call merely tells Oracle Trace that the thread context should be updated (that is, resource utilization items should be charged to this thread).

```
!
!----------------------------------------------------------------------
!
!  Global routine called NEWT$$THREAD_START_RESUME
!

!  When creating a new thread, allocate a thread block (called
!  thread_block) for storage local to this thread only.  In addition,
!  initialize the thread_id field to zero, so you pass a zero value
!  context variable argument to EPC$SET_CONTEXT.  EPC$SET_CONTEXT
!  then fills in the thread_id field with a unique value associated
!  with this thread.

!  Call EPC$SET_CONTEXT to tell Oracle Trace that a new thread is
!  executing or that an existing thread is resuming execution.

   cond_status = EPC$SET_CONTEXT (0,thread_block[thread_id]);

           .
           .
           .

!  Call the NEWT$$TRANSACTION routine to process the database
!  transaction.

   cond_status = NEWT$$TRANSACTION (...);

           .
           .
           .
```

4.10.2 Deleting the Thread Context

The application issues a call to the EPC$DELETE_CONTEXT service routine when the thread has completed execution, prior to where the NEW_TOOL facility deallocates the thread block for this thread. The call tells Oracle Trace to no longer keep track of the resource utilization and cross-facility items for this thread.

```
!
!----------------------------------------------------------------------
!
!  Global routine called NEWT$$THREAD_DELETE
!

!  Call EPC$DELETE_CONTEXT to tell Oracle Trace not to keep resource
!  utilization items for this thread any longer.

   status = EPC$DELETE_CONTEXT (0,thread_block[thread_id]);

!  Deallocate the thread_block.
```

4.11 Linking an Instrumented Program

There are two methods for linking an instrumented program. The one you use depends on whether or not Oracle Trace is installed on the target system (the system where you will be running the application) and its OpenVMS version.

A stub Oracle Trace shareable image (SYS$SHARE:EPC$SHR.EXE) is included with OpenVMS Version 5.2 and higher. It will return a status of EPC$_NOTINSTALL to all service routines.

Note

On the OpenVMS Alpha operating system Version 1.0 and higher, the stub Oracle Trace shareable image resides in SYS$SHARE.

The active Oracle Trace image is present on all systems where the Oracle Trace software is installed. To link an instrumented program on these systems, simply use the OpenVMS LINK command. The resulting executable image will run on any system with OpenVMS Version 5.2 or higher, or on any system where Oracle Trace is installed. The following example shows the command to link the ATM sample application:

```
$ LINK EPC$ATM-SAMPLE.OBJ
```

If you anticipate that the Oracle Trace image might be deleted from the system or that your application might be installed on a system without Oracle Trace, (that is, OpenVMS Version 5.1 system) you should use LIB$FIND_IMAGE_SYMBOL in your instrumentation to resolve symbolic references to the Oracle Trace service routines. Conditionalize each Oracle Trace service routine call using LIB$FIND_IMAGE_SYMBOL for cases when the Oracle Trace active image is not present on a system. Thus you can avoid having your application make unnecessary Oracle Trace routine calls.

Example 4–14 shows a code segment of an instrumented program that uses LIB$FIND_IMAGE_SYMBOL calls. The example is written in C.

When you have completed the tasks described in this chapter, you can proceed to Chapter 5 to specify a facility definition for your application.

Example 4–14 Instrumentation Using LIB$FIND_IMAGE_SYMBOL

```
extern unsigned long int (*MYFAC$EPC$DELETE_CONTEXT)();
extern unsigned long int (*MYFAC$EPC$END_EVENT)();
extern unsigned long int (*MYFAC$EPC$END_EVENTW)();
extern unsigned long int (*MYFAC$EPC$EVENT)();
extern unsigned long int (*MYFAC$EPC$EVENTW)();
extern unsigned long int (*MYFAC$EPC$INIT)();
extern unsigned long int (*MYFAC$EPC$SET_CONTEXT)();
extern unsigned long int (*MYFAC$EPC$START_EVENT)();
extern unsigned long int (*MYFAC$EPC$START_EVENTW)();
   .
   .
   .
$DESCRIPTOR(facility_ver, "V1.0-0"); /* Facility Version */
$DESCRIPTOR (epc_lib_desc , "EPC$SHR");

$DESCRIPTOR (delete_context_desc , "EPC$DELETE_CONTEXT");
$DESCRIPTOR (end_event_desc , "EPC$END_EVENT");
$DESCRIPTOR (end_eventw_desc , "EPC$END_EVENTW");
$DESCRIPTOR (event_desc , "EPC$EVENT");
$DESCRIPTOR (eventw_desc , "EPC$EVENTW");
$DESCRIPTOR (init_desc , "EPC$INIT");
$DESCRIPTOR (set_context_desc , "EPC$SET_CONTEXT");
$DESCRIPTOR (start_event_desc , "EPC$START_EVENT");
$DESCRIPTOR (start_eventw_desc , "EPC$START_EVENTW");

/* Get entry points into Oracle Trace */
status = LIB$FIND_IMAGE_SYMBOL (&epc_lib_desc ,
                  &delete_context_desc , &MYFAC$EPC$DELETE_CONTEXT);
/* Error check */
status = LIB$FIND_IMAGE_SYMBOL (&epc_lib_desc ,
                  &end_event_desc , &MYFAC$EPC$END_EVENT);
/* Error check */
status = LIB$FIND_IMAGE_SYMBOL (&epc_lib_desc ,
                  &end_eventw_desc , &MYFAC$EPC$END_EVENTW);
/* Error check */
status = LIB$FIND_IMAGE_SYMBOL (&epc_lib_desc ,
                  &event_desc , &MYFAC$EPC$EVENT);
/* Error check */
status = LIB$FIND_IMAGE_SYMBOL (&epc_lib_desc ,
                  &eventw_desc , &MYFAC$EPC$EVENTW);
```

(continued on next page)

Example 4–14 (Cont.) Instrumentation Using LIB$FIND_IMAGE_SYMBOL

```
/* Error check */
status = LIB$FIND_IMAGE_SYMBOL (&epc_lib_desc ,
                  &init_desc , &MYFAC$EPC$INIT);

/* Error check */
status = LIB$FIND_IMAGE_SYMBOL (&epc_lib_desc ,
                  &set_context_desc , &MYFAC$EPC$SET_CONTEXT);

/* Error check */
status = LIB$FIND_IMAGE_SYMBOL (&epc_lib_desc ,
                  &start_event_desc , &MYFAC$EPC$START_EVENT);

/* Error check */
status = LIB$FIND_IMAGE_SYMBOL (&epc_lib_desc ,
                  &start_eventw_desc , &MYFAC$EPC$START_EVENTW);

/* Error check */
/* If all calls to LIB$FIND_IMAGE_SYMBOL returned success */
if ((status & 1) == 1)
  {
     /* Issue initialize call to Oracle Trace Registrar */

        status = MYFAC$EPC$INIT(event_flag,  /* Event Flag Number      */
                    MYFAC$_FACILITY,         /* Facility number        */
                    &facility_ver,           /* Facility version string*/
                    &registration_id,        /* Registration ID string */
                    &MYFAC$_EPC_EVENT_FLAGS, /* Event flags structure  */
                    &MYFAC$_EPC_ITEM_FLAGS,  /* Event item flags struc */
                    &init_iosb,              /* IOSB                   */
                    0,                       /* AST address            */
                    0);                      /* AST parameter          */
  }
```

5
Creating Facility Definitions

Each application program that is instrumented with Oracle Trace service routine calls must provide a **facility definition**, which describes the *events* in the application and the *items* to collect for each event. A facility definition also provides useful information for reporting purposes, such as header information for the events and items.

This chapter describes how to work with Oracle Trace facility definitions including:

- Creating facility definitions
- Deleting facility definitions
- Transporting facility definitions between systems
- Displaying information about facility definitions stored in the Oracle Trace administration database

Table 5–1 summarizes the Oracle Trace commands available to manipulate facility definitions. See Chapter 6 for a description of the commands and their syntax.

Table 5–1 Facility Definition Commands

Command	Description
CREATE DEFINITION	Creates a facility definition and stores it in the Oracle Trace administration database.
DELETE DEFINITION	Deletes a facility definition from the Oracle Trace administration database.

(continued on next page)

Table 5–1 (Cont.) Facility Definition Commands

Command	Description
EXTRACT DEFINITION	Extracts a facility definition from the Oracle Trace administration database and stores it in a binary file.
INSERT DEFINITION	Inserts a facility definition into the Oracle Trace administration database from a binary Oracle Trace facility definition file.
SHOW DEFINITION	Displays information about facility definitions in the Oracle Trace administration database.

5.1 Creating a Facility Definition

Oracle Trace facility definitions consist of:

- Name of the facility.
- ID number of the facility. Numbers in the range 1 to 2047 are registered with Oracle Corporation and are referred to as **registered facilities**. Numbers in the range 2048 to 4095 specify **non-registered facilities**.
- Version of the facility (up to 10 characters). Some typical formats of version strings are:
  - Vnn.nn (versions of software)
  - Vnn.nn-nn (versions and baselevels or revision levels of software)
  - Tnn.nn-nn (field test versions of software and baselevel)
- List of the events that occur within the facility.
- List of facility-specific items associated with the events.

The format of the CREATE DEFINITION command is:

```
CREATE DEFINITION facility_name facility_id [ /EVENTS=(event_name[, . . . ])
                                              /OPTIONS[=file_spec]
                                              /REPLACE
                                              /VERSION="version_code" ]
```

There are two methods for creating a facility definition. You can create a simple definition that associates only the default resource utilization items (shown in Table 5–2) with your events, or you can create an advanced definition with items that are specific to your application.

To collect the default data, you can create the definition interactively. For example, the following command creates the facility definition for the DB facility which has two duration events:

```
$ COLLECT CREATE DEFINITION DB 2048 /VERSION="V1.0-0" -
_$ /EVENTS=(EVENT_1, EVENT_2)
```

To create a more detailed definition including facility-specific items and classes, you should use the /OPTIONS qualifier to the CREATE DEFINITION command and specify an options file. There are several different ways to use the /OPTIONS qualifier:

- Enter the /OPTIONS qualifier interactively without supplying a file specification; Oracle Trace prompts you for the options. For example:

  ```
  $ COLLECT
  Trace> CREATE DEFINITION DB 2048 /VERSION="V1.0-0" /OPTIONS
  Option> ITEM TRANSACTION_ID LONGWORD /ID=1  . . .
     .
     .
     .
  ```

- Enter the /OPTIONS qualifier interactively and supply a file specification that Oracle Trace uses as a source for facility definition options. For example:

  ```
  $ COLLECT CREATE DEFINITION DB 2048 /VERSION="V1.0-0" -
  _$ /OPTIONS=DB_OPTIONS.OPT
  ```

- Invoke Oracle Trace to execute a file that contains the CREATE DEFINITION command as well as the options:

  ```
  $ COLLECT @DEFINE_DB.COM
  ```

- Execute an OpenVMS command procedure that invokes Oracle Trace, enters the CREATE DEFINITION command, and so forth. For example:

  ```
  $ @DB.COM
  ```

5.1.1 Creating and Defining Events

Oracle Trace collects data for **events** which occur during run time. In the automated teller machine (ATM) sample application, the defined events are:

- Customer checks account balance.
- Customer deposits funds into account.
- Customer withdraws funds from account.
- Customer makes an error and the application displays an error message.

The balance, deposit, and withdrawal events are **duration events**, which begin when the customer selects the appropriate menu option and end when the transaction is complete. The error message event is a **point event**, which has no logical start or end and simply occurs.

Once you have determined the events that your facility contains, you must instrument your source code with Oracle Trace service routine calls. Chapter 4 describes how to instrument your code, and Chapter 7 describes the formats of the service routines.

After instrumenting your code, you must create a facility definition that includes the events you have implemented. The /EVENTS qualifier or EVENTS option to the CREATE DEFINITION command specifies a list of events that occur within the facility. Note that in a simple facility definition, only duration events are generated and only the resource utilization items listed in Table 5–2 can be collected for them. To collect facility-specific items or to collect a subset of the resource items, you must use the /OPTIONS qualifier instead of the /EVENTS qualifier. The /OPTIONS qualifier allows you to specify the following EVENT characteristics:

- Name of the event.
- ID number of the event. The default is the next unused event ID between 1 and 128.
- Report header to use for the event.
- List of all items associated with the event.
- Items to collect at the start of duration events.
- Items to collect at the end of duration events.
- Items to collect for point events

The format of the EVENT option is:

```
EVENT event_name [ /IDENTIFIER=event_id
                   /ITEMS=(item_name[, . . . ])
                   /START_EVENT=(item_name[, . . . ])
                   /END_EVENT=(item_name[, . . . ])
                   /POINT_EVENT=(item_name[, . . . ])
                   /REPORT_HEADER="text" ]
```

The following example creates the ATM_SAMPLE facility definition using the /OPTIONS qualifier:

```
$ COLLECT CREATE DEFINITION ATM_SAMPLE 4094 /VERSION="V1.0" /OPTIONS
Option> EVENT ERROR_DISPLAY /ID=1 /REPORT_HEADER="Errors" -
_Option> ITEMS=(RESOURCE_ITEMS) /POINT_EVENT=(RESOURCE_ITEMS)
Option> EVENT BALANCE_EVENT /ID=2 /REPORT_HEADER="Balance" -
_Option> /ITEMS=(RESOURCE_ITEMS) /START_EVENT=(RESOURCE_ITEMS) -
_Option> /END_EVENT=(RESOURCE_ITEMS)
Option> EVENT DEPOSIT_EVENT /ID=3 /REPORT_HEADER="Deposits" -
_Option> /ITEMS=(RESOURCE_ITEMS) /START_EVENT=(RESOURCE_ITEMS) -
_Option> /END_EVENT=(RESOURCE_ITEMS)
Option> EVENT WITHDRAW_EVENT /ID=4 /REPORT_HEADER="Withdrawals" -
_Option> /ITEMS=(RESOURCE_ITEMS) /START_EVENT=(RESOURCE_ITEMS) -
_Option> /END_EVENT=(RESOURCE_ITEMS)
Option> EXIT
```

Alternately, you could use a command file to create the ATM_SAMPLE facility definition[1]. For example:

```
$ COLLECT @EPC$ATM-FAC-DEF.COM
```

EPC$ATM-FAC-DEF.COM contains the following lines:

```
CREATE DEFINITION ATM_SAMPLE 4094 /VERSION="V1.0" /OPTIONS
!
EVENT ERROR_DISPLAY /ID=1 /REPORT_HEADER="Errors" /ITEMS=(RESOURCE_ITEMS) -
      /POINT_EVENT=(RESOURCE_ITEMS)
EVENT BALANCE_EVENT /ID=2 /REPORT_HEADER="Balance" /ITEMS=(RESOURCE_ITEMS) -
      /START_EVENT=(RESOURCE_ITEMS) /END_EVENT=(RESOURCE_ITEMS)
EVENT DEPOSIT_EVENT /ID=3 /REPORT_HEADER="Deposits" /ITEMS=(RESOURCE_ITEMS)-
      /START_EVENT=(RESOURCE_ITEMS) /END_EVENT=(RESOURCE_ITEMS)
EVENT WITHDRAW_EVENT /ID=4 /REPORT_HEADER="Withdrawals" -
      /ITEMS=(RESOURCE_ITEMS) -
      /START_EVENT=(RESOURCE_ITEMS) /END_EVENT=(RESOURCE_ITEMS)
!
EXIT
```

See Section 5.6 for a full description of the facility definition options.

[1] You can find EPC$ATM-FAC-DEF.COM and EPC$ATM-FAC-DEF-EXTENDED.COM in the EPC$EXAMPLES directory.

5.1.2 Creating and Defining Items

Items are elements of data associated with each event. They describe *what* is collected for the event. Table 5–2 describes the set of standard Oracle Trace resource utilization items. These items are often taken as a whole, and to facilitate this, they can be referred to by the group name: RESOURCE_ ITEMS. If disk space is not a concern, collecting all of the resource items requires less CPU overhead than collecting a subset of them. Note that unlike user-defined items, if you collect the resource items for a duration event, you must collect them on both the start and end events.

Table 5–2 Standard Resource Utilization Items

Item Name	Item ID Number	Description	Data Type	Usage
BIO	101	Number of buffered I/O operations	Longword	Counter
DIO	102	Number of direct I/O operations	Longword	Counter
PAGEFAULTS	103	Total number of hard and soft page faults	Longword	Counter
PAGEFAULT_IO	104	Number of hard page faults (that is, page faults to or from the disk)	Longword	Counter
CPU	105	Total amount of CPU time in tens of milliseconds	Longword	Counter
CURRENT_PRIO	106	Current priority of the process	Word	Level
VIRTUAL_SIZE	107	Number of virtual pages that are currently mapped for the process	Longword	Level
WS_SIZE	108	Current working set size of the process	Longword	Level
WS_PRIVATE	109	Number of pages in the working set that are private to the process	Longword	Level
WS_GLOBAL	110	Number of pages in the working set that are globally shared among processes on the system	Longword	Level

Note

Oracle Trace stamps each data collection record with the current date and time, extended process identification number (EPID), facility number, and event identifier. Therefore, the elapsed time information for the EPID, the facility that logged the event, and the event identifier are always collected.

You can collect information in addition to (or instead of) that provided by the resource utilization items by defining facility-specific items and associating them with your events. You use the /OPTIONS qualifier to the CREATE DEFINITION command to create facility-specific items. The /OPTIONS qualifier allows you to specify the following ITEM characteristics:

- Name of the item.
- Data type of the item.
- ID number of the item. The default is the next unused item ID between 1 and 100.
- Maximum size (in bytes) of the item.
- Report header for the item.
- Width of the column when displaying the item in an Oracle Trace report.
- Usage type of the item. Valid types are: LEVEL, COUNTER, PERCENT, TEXT, and PRIVATE.
- Characteristics of the item. Items can be either printable or nonprintable.

The format of the ITEM option is:

ITEM item_name datatype [/IDENTIFIER=item_id | /SIZE=n-bytes | /REPORT_HEADER="text" | /REPORT_WIDTH=number-of-spaces | /USAGE_TYPE=usage-type | /CHARACTERISTICS=[non]printable]

The following example creates a facility definition for the ATM facility with five items and five events:

```
CREATE DEFINITION ATM_SAMPLE 4094 /REPLACE /VERSION="V1.2-0" /OPTIONS
!
! Items specific to ATM_SAMPLE facility version V1.1-0
!
! We will always pass a 10 byte buffer to Trace, 1 for the length of
! the data and the other 9 bytes = the data
ITEM  ACCOUNT_ID FIXED_ASCIC /SIZE=9 /ID=1 /REPORT_HEADER="Account Number" -
        /CHARACTERISTICS=PRINTABLE /REPORT_WIDTH=9 /USAGE_TYPE=TEXT

ITEM  MACHINE_NUMBER    LONGWORD /ID=2 /REPORT_HEADER="Machine Number" -
        /CHARACTERISTICS=PRINTABLE /REPORT_WIDTH=5 /USAGE_TYPE=LEVEL
```

```
! ASCIC may be up to 255 characters however we force max to be 40 here
ITEM  MACHINE_LOC       ASCIC /SIZE=40 /ID=3 /REPORT_HEADER="ATM Location" -
        /CHARACTERISTICS=PRINTABLE /REPORT_WIDTH=20 /USAGE_TYPE=TEXT
!
ITEM  BANK_NAME ASCIW /SIZE=40 /ID=4 /REPORT_HEADER="Bank Name" -
        /CHARACTERISTICS=PRINTABLE /REPORT_WIDTH=20 /USAGE_TYPE=TEXT
!
ITEM  BANK_LOCATION ASCIW /SIZE=40 /ID=5 /REPORT_HEADER="Bank Location" -
        /CHARACTERISTICS=PRINTABLE /REPORT_WIDTH=20 -
        /USAGE_TYPE=TEXT GROUP ACCOUNT_ITEMS -
        /ITEMS=(ACCOUNT_ID,MACHINE_NUMBER,MACHINE_LOC,BANK_NAME,-
                          BANK_LOCATION)
!
EVENT ERROR_DISPLAY /ID=1 /REPORT_HEADER="Errors" /ITEMS=(RESOURCE_ITEMS) -
      /POINT_EVENT=(RESOURCE_ITEMS)
EVENT BALANCE_EVENT /ID=2 /REPORT_HEADER="Balance" -
        /ITEMS=(RESOURCE_ITEMS,CROSS_FAC_14) -
        /START_EVENT=(RESOURCE_ITEMS,CROSS_FAC_14) -
        /END_EVENT=(RESOURCE_ITEMS,CROSS_FAC_14)
EVENT DEPOSIT_EVENT /ID=3 /REPORT_HEADER="Deposits" -
        /ITEMS=(RESOURCE_ITEMS,CROSS_FAC_14) -
        /START_EVENT=(RESOURCE_ITEMS,CROSS_FAC_14) -
        /END_EVENT=(RESOURCE_ITEMS,CROSS_FAC_14)
EVENT WITHDRAW_EVENT /ID=4 /REPORT_HEADER="Withdrawals" -
        /ITEMS=(RESOURCE_ITEMS,CROSS_FAC_14) -
        /START_EVENT=(RESOURCE_ITEMS, CROSS_FAC_14) -
        /END_EVENT=(RESOURCE_ITEMS, CROSS_FAC_14)
!
! Note, can use group items or list the items, both are shown here
!
EVENT SIGNIN_EVENT /ID=5 /REPORT_HEADER="Signins" -
        /ITEMS=(ACCOUNT_ITEMS,RESOURCE_ITEMS, CROSS_FAC_14) -
        /START_EVENT=(ACCOUNT_ID, MACHINE_NUMBER, MACHINE_LOC, RESOURCE_ITEMS,
        /END_EVENT=(BANK_NAME, BANK_LOCATION, RESOURCE_ITEMS, CROSS_FAC_14)
!
```

Note that the /OPTIONS and /EVENTS (as described in Section 5.1.1) qualifiers are mutually exclusive. See Section 5.6 for a full description of the facility definition options. Section 4.9.3 describes how to instrument items into your application source code.

5.1.3 Creating and Defining Collection Classes

Oracle Trace can collect data from all of the events and items defined for your facility. However, you can reduce overhead in terms of both CPU and disk space utilization by collecting only data that pertains to your current needs. You can change your facility definition without changing your instrumented application.

You can create subsets of your events and items that are relative to specific collection purposes. A facility selection (as described in Chapter 2) can specify a **collection class** to limit data collection to only that data that is important to the user. You can create a class for any usage. However, Oracle Corporation recommends that application product developers define one or more of the following standard classes:

- CAPACITY_PLANNING—Includes those events and items that are useful for capacity planning purposes.
- DEBUGGING—Includes those events and items that are useful for tracing the execution of your application.
- ERROR_LOGGING—Includes those events and items that are associated with error handling routines in your application.
- PERFORMANCE—Includes those events and items that are useful for application and/or database tuning.
- WORKLOAD—Includes those events and items that are useful for gathering information for tracing the actual workload of the application or the database management system.

Use the /OPTIONS qualifier to the CREATE DEFINITION command to create collection classes. The format of the CLASS option is:

```
CLASS class_name event_name     /ITEMS=(item_name[, . . . ])
```

To define a collection class, list each event you want to include in the class and specify the items that you want to collect for that event. Later, you can specify that class on the CREATE SELECTION command (see Section 2.3). If you do not include an event in the class definition, no data is collected for that event when the application executes. The following example defines WORKLOAD and PERFORMANCE classes for the NEW_TOOL application:

```
! NEW_TOOL facility definition
CREATE DEFINITION NEW_TOOL 2049 /VERSION="T1.0-1" /OPTIONS
!
ITEM STREAM_ID LONGWORD /ID=1 /REPORT_HEADER="Stream Id" /REPORT_WIDTH=11 -
  /USAGE_TYPE=LEVEL /CHARACTERISTICS=PRINTABLE /RADIX=HEXADECIMAL
ITEM DBS_READS LONGWORD /ID=2 /REPORT_HEADER="Data File Reads" -
  /REPORT_WIDTH=10 /USAGE_TYPE=COUNTER /CHARACTERISTICS=PRINTABLE
ITEM DBS_WRITES LONGWORD /ID=3 /REPORT_HEADER="Data File Writes" -
  /REPORT_WIDTH=10 /USAGE_TYPE=COUNTER /CHARACTERISTICS=PRINTABLE
ITEM DB_NAME ASCIC /ID=4 /REPORT_HEADER="DB Name" /REPORT_WIDTH=25 -
  /USAGE_TYPE=TEXT /CHARACTERISTICS=PRINTABLE /SIZE=255
ITEM LOCK_MODE BYTE /ID=5 /REPORT_HEADER="Lock Mode" /REPORT_WIDTH=4 -
  /USAGE_TYPE=LEVEL /CHARACTERISTICS=PRINTABLE
```

```
!
EVENT TRANSACTION /ID=1 /REPORT_HEADER="Transaction" -
  /ITEMS=(RESOURCE_ITEMS, STREAM_ID, DB_NAME, DBS_READS, DBS_WRITES) -
  /START_EVENT=(RESOURCE_ITEMS, STREAM_ID, DB_NAME, DBS_READS, DBS_WRITES) -
  /END_EVENT=(RESOURCE_ITEMS, STREAM_ID, DB_NAME, DBS_READS, DBS_WRITES)
EVENT DATABASE /ID=2 /REPORT_HEADER="Database" -
  /ITEMS=(RESOURCE_ITEMS, STREAM_ID, DB_NAME, DBS_READS, DBS_WRITES) -
  /START_EVENT=(RESOURCE_ITEMS, STREAM_ID, DB_NAME, DBS_READS, DBS_WRITES) -
  /END_EVENT=(RESOURCE_ITEMS, STREAM_ID, DB_NAME, DBS_READS, DBS_WRITES)
!
CLASS WORKLOAD DATABASE /ITEMS=(STREAM_ID, DBS_READS, DBS_WRITES)
CLASS WORKLOAD TRANSACTION /ITEMS=(STREAM_ID, DBS_READS, DBS_WRITES)
!
CLASS PERFORMANCE TRANSACTION /ITEMS=*
!
DEFAULT_CLASS WORKLOAD
!
EXIT
```

Note that the ALL class is created by default when you create a facility definition and is composed of all events and associated items in your facility. The ALL class is also the default class unless you specify another with the DEFAULT_CLASS option.

See Section 5.6 for a full description of the facility definition options. Section 4.9.2.4 describes how to use classes efficiently and how to instrument the relevant events and items into your application source code.

5.2 Relating Events Among Facilities

To gain an understanding of the full context in which the events occur when they track application performance and resource use, programmers often relate events among two or more facilities. There are two ways that programmers can relate events among applications. The most common way is to explicitly pass data items through the application programming interface (API). When instrumenting an application that relates events in this way, you can ensure that Oracle Trace will collect these items by designating them in the user-defined buffer passed in the Oracle Trace service routine calls.

When facilities cannot change the existing API, or cannot pass data items through it, as is the case with SQL (DML), Oracle Trace provides the ability to relate these events using the cross-facility feature. You must perform two steps to relate events using the Oracle Trace cross-facility feature:

1. Assign the appropriate values to cross-facility items using Oracle Trace cross-facility service routines described in Section 4.9.3.

2. Define cross-facility items in the facility definition as described in this chapter.

Oracle Trace transparently collects the relevant cross-facility items for the facilities as the events occur. However, each facility needs to include the relevant cross-facility items in the item list of the event option of their facility definition. This relating of items allows a better understanding of the context in which certain events execute.

For example, two related events are the PROCEDURE_CALL in ACMS and the TRANSACTION event in Oracle Rdb. Because both of these facilities log the CROSS_FAC_2 item on their events, Oracle Trace is able to relate the events in the data collection file and can later generate reports using the relation (see the *Oracle Trace Reporter User's Guide*).

Table 5–3 lists the 14 cross-facility items available to programmers who are instrumenting their application for Oracle Trace applications. Note that only these items can be used as cross-facility items; all other items are accessible only within the context of a single facility.

Table 5–3 Cross-Facility Items

Cross-Facility Item Name	Item Number	Data Type	Facility that Sets the Value
CROSS_FAC_1	115	Longword	ALL–IN–1 and other OA software products
CROSS_FAC_2	116	Longword	ACMS, DECintact, other TP monitor products, collected by Oracle Rdb, Oracle CODASYL DBMS
CROSS_FAC_3	117	Longword	Oracle RALLY and possibly other 4GL products, collected by Oracle Rdb
CROSS_FAC_4	118	Longword	Reserved
CROSS_FAC_5	119	Longword	Reserved
CROSS_FAC_6	120	Longword	Reserved

(continued on next page)

Table 5–3 (Cont.) Cross-Facility Items

Cross-Facility Item Name	Item Number	Data Type	Facility that Sets the Value
CROSS_FAC_7	121	Longword	Set by ACMS, collected by Oracle Rdb, Oracle CODASYL DBMS
CROSS_FAC_8	122	Longword	Reserved
CROSS_FAC_9	123	Longword	Reserved
CROSS_FAC_10	124	Longword	Reserved
CROSS_FAC_11	125	Longword	Reserved
CROSS_FAC_12	126	Longword	Reserved
CROSS_FAC_13	127	Longword	Reserved
CROSS_FAC_14	128	Longword	General customer facility use, unrestricted

See the documentation of each facility to learn how they use the cross-facility items that are assigned to them.

Any application can add cross-facility items to its facility definition. However, if the application is going to set one of the cross-facility items, Oracle Corporation recommends that you register this usage with the Oracle Trace development group. See Appendix B for information on how to do this.

5.2.1 Defining Relationships Among Events

Use the RELATE option to the CREATE DEFINITION command to define a relationship between events and items in different facilities. When you define a relationship, Oracle Trace stores the relationship information in the formatted database making it available to facilities that rely on that information, such as Oracle Expert for Rdb.

The RELATE option has the following format:

```
RELATE  relationship-name table-name source-facility-name source-event-name
        source-item-name target-facility-name target-event-name
        target-item-name
          [ /[NO]COMMENT="string"
            /HIERARCHY_LEVEL=level
            /[NO]S1="string"        ]
```

The relationship-name and table-name are simply names you create to describe the relationship between the related pair. The other names must match the exact names of the facilities, events, and items in the two facilities.

The following code example shows the parts of the facility definitions for DEC ACMS and Oracle Rdb, relating through CROSS_FAC_2:

```
CREATE DEFINITION ACMS 253 /VERSION="V3.3-0"/REPLACE/OPTIONS

   .
   .
   .
   EVENT PROCEDURE_CALL /ID=5-
   /ITEMS=( . . .  ,CROSS_FAC_2)
   /START_EVENT=( . . .  ,CROSS_FAC_2)
   /END_EVENT= . . .  )

   .
   .
   .

!---------------------------------------
! Relationships for Oracle Expert for Rdb
!---------------------------------------
!
RELATE RDBEXPERT_APPLICATION RDBX$WORKLOAD/NOS1 -
 ACMS PROCEDURE_CALL PROCEDURE_INDEX -
 ACMS PROCEDURE_CALL PROCEDURE_INDEX
!
RELATE RDBEXPERT_APPLICATION_NAME RDBX$WORKLOAD/NOS1 -
 ACMS PROCEDURE_CALL PROCEDURE_INDEX -
 ACMS PROCEDURE_CALL APPL_SPEC
!
RELATE RDBEXPERT_PROGRAM RDBX$WORKLOAD/NOS1 -
 RDBVMS TRANSACTION CROSS_FAC_7 -
 ACMS PROCEDURE_CALL PROCEDURE_INDEX
!
RELATE RDBEXPERT_PROGRAM_NAME RDBX$WORKLOAD/NOS1 -
 ACMS PROCEDURE_CALL PROCEDURE_INDEX -
 ACMS PROCEDURE_CALL PROCEDURE_NAME
!
```

```
CREATE DEFINITION RDBVMS 221/VERSION="V4.1-0"/REPLACE/OPTIONS
   .
   .
   .
   EVENT TRANSACTION/ID=2-
   /ITEMS=( . . .  ,CROSS_FAC_2,CROSS_FAC_3,CROSS_FAC_14)
   /START_EVENT=( . . .  ,CROSS_FAC_2,CROSS_FAC_3,CROSS_FAC_14)
   /END_EVENT=( . . .  ,CROSS_FAC_2,CROSS_FAC_3,CROSS_FAC_14)
   .
   .
   .
```

Note that the RELATE option does not need to be present in both facility definitions. Also, just like the Oracle Trace resource utilization items, the cross-facility items must be collected on both the start and end of duration events.

5.3 Deleting Facility Definitions

You can delete facility definitions from the Oracle Trace administration database with the DELETE DEFINITION command. Note that you do not delete the actual facility images from your system, but merely the Oracle Trace facility definitions. To delete definitions of registered facilities (numbered 1 to 2047), you must have OpenVMS BYPASS or SYSPRV privilege. To delete definitions of non-registered facilities (numbered 2048 to 4095), you must either be the creator of the definition or have OpenVMS BYPASS or SYSPRV privilege.

The following example deletes the facility definition for version T1.0 of the TEMP_PROG facility:

```
$ COLLECT DELETE DEFINITION TEMP_PROG /VERSION="T1.0"
Delete TEMP_PROG T1.0 [N]: YES
%EPC-S-FACDEL_DELETED, Facility definition TEMP_PROG T1.0 was deleted
```

You cannot delete a facility definition if any active or pending collections are collecting data from that facility. Use the SHOW SELECTION /FORMAT=BRIEF command to determine if a facility has active or pending data collection associated with it. If it does, you must use the CANCEL COLLECTION command to stop data collection before you can delete the facility definition. In addition, you have to delete any facility selections that explicitly specify the version of the facility whose definition you want to delete. If a facility selection does not specify the version of the facility and another version exists on the system, you can delete the facility definition without canceling scheduled data collection or deleting the facility selection.

The following example shows how to delete a facility definition that has active data collection associated with it:

```
$ COLLECT DELETE DEFINITION TEMP_PROG /VERSION="T1.0" /NOCONFIRM
%EPC-F-FACDEL_NOTDELETED, Facility is referenced by active data collection
$ COLLECT
Trace> SHOW SELECTION/FORMAT=BRIEF

22-AUG-1995 12:11       Facility Selection Information            Page 1
                                                        Oracle Trace V2.2-0

   Selection Name        Facility         Version     Class
   --------------------  ---------------  ----------  --------------------
   A1_DATA_ENTRY         OA               (latest)    ALL
                         TESTER           T4.1        ALL
                         RDBVMS           V3.1        ALL

   TEST_SELECT           TEMP_PROG        T1.0        PERFORMANCE

Trace> CANCEL COLLECTION /SELECTION=TEST_SELECT /NOCONFIRM
%EPC-S-SCHED_CANCEL, Collection QUICK_TEST is cancelled

Trace> DELETE SELECTION TEST_SELECT /NOCONFIRM
%EPC-S-SELDEL_DELETED, Facility selection TEST_SELECT was deleted
Trace> DELETE DEFINITION TEMP_PROG /VERSION="T1.0"
Delete TEMP_PROG T1.0 [N]: YES
%EPC-S-FACDEL_DELETED, Facility definition TEMP_PROG T1.0 was deleted
Trace> EXIT
$
```

You can use the SHOW DEFINITION /FORMAT=NAMES_ONLY command to confirm the spelling of the names and the version codes for the facility definitions that you want to delete.

5.4 Transporting Facility Definitions

You can transport a facility definition to another system or VMScluster with the EXTRACT DEFINITION and INSERT DEFINITION commands. This feature is useful in creating installation kits for your applications. Your installation procedure can automatically add a facility definition to the Oracle Trace administration database.

Note that you cannot simply back up the Oracle Trace administration database and then restore it on the new system or VMScluster. The Oracle Trace Registrar on the target system would attempt to start collections on nodes that are not part of the new system or VMScluster. See the *Oracle Trace Installation Guide* for information on dividing a VMScluster and preserving the contents of the original administration database.

5.4.1 Extracting Definitions

You can extract a facility definition from the Oracle Trace administration database with the EXTRACT DEFINITION command. The command creates a binary file containing the facility definition information for the specified facility. You must have OpenVMS BYPASS, SYSPRV, or READALL privilege to extract registered facility definitions or non-registered facility definitions that were created by another user. The files are stored in binary format to prevent tampering with the facility definitions.

The following command extracts the facility definition for V1.0 of the ATM_SAMPLE facility. The facility definition is stored in the file ATM_FAC_DEF.EPC$DEF:

```
$ COLLECT EXTRACT DEFINITION ATM_SAMPLE ATM_FAC_DEF /VERSION="V1.0"
%EPC-S-FACEXT, Facility definition(s) was successfully extracted
```

5.4.2 Inserting Definitions Using a KITINSTAL.COM Procedure

Oracle Trace provides a callback procedure compliant with the VMSINSTAL procedure that any product can use to automatically insert a binary facility definition into the Oracle Trace administration database. The procedure is called INSERT_DEF, and is located in SYS$UPDATE:EPC$VMSINSTAL_CALLBACK.COM. If Oracle Trace is not installed on the system, the procedure inserts the facility definition into the Oracle Trace facility library: SYS$COMMON:[SYSLIB]EPC$FACILITY.TLB, which is shipped with Version 5.2 or higher of the OpenVMS operating system.

Note

Oracle Corporation recommends that you include EPC$VMSINSTAL_CALLBACK.COM in your application's installation kit. This ensures that the facility definition can be inserted into the administration database even if Oracle Trace is not installed on the target node.

During installations of Oracle Trace, the system manager is asked to run EPC$INSERT.COM (in EPC$EXAMPLES:), which checks the facility library and moves any facility definitions found there into the Oracle Trace administration database.

Example 5–1 shows a segment of a KITINSTAL.COM procedure which includes the callback. The facility definition for MY_APPLICATION V1.0 is inserted into the Oracle Trace administration database.

Example 5–1 Sample Procedure to Insert Binary Facility Definitions

```
$ INSERT_DEF:
$ !+
$ !  Attempt to insert the facility definition.  On error (Oracle Trace not
$ !  installed) insert it into sys$share:epc$facility.tlb
$ !
$ !  Parameters:
$ !
$ !  P2 : Return value (see below)
$ !  P3 : File specification for the binary facility definition file.
$ !  P4 : The facility name, up to 27 characters in length
$ !  P5 : The facility version, string up to 10 characters in length.
$ !
$ !  Return values in P2:
$ !
$ !   I : Successfully inserted into Oracle Trace admin db.
$ !   L : Successfully inserted into SYS$SHARE:EPC$FACILITY.TLB.
$ !   N : No operations were performed.
$ !
$ !-
$ !+
$ !     Example of how to call INSERT_DEF callback.
$ !-
$ !+
$ ! Set up debug mode (VMSINSTAL OPTION K)
$ !-
$     _o = "!"                  ! default is nodebug
$     _x = "!"                  ! default is nodebug
$     IF .NOT. P2 THEN GOTO START
$     _o = "SET VERIFY"         ! ON around OUR code
$     _x = "SET NOVERIFY"       ! OFF around Callbacks
$_o
$
$ START:
$ !+
$ ! Copy the callback procedures to sys$update.
$ !-
$   COPY/NOLOG VMI$KWD:EPC$VMSINSTAL_CALLBACK.COM VMI$ROOT:[SYSUPD]*.*;
$ !
$ INSERT_FACDEFS:
$_x
$   VMI$CALLBACK PRODUCT EPC$VMSINSTAL_CALLBACK:INSERT_DEF -
          RETURN_SYMBOL VMI$KWD:MY_FAC_DEF.EPC$DEF -
          MY_APPLICATION V1.0-0
```

(continued on next page)

Example 5–1 (Cont.) Sample Procedure to Insert Binary Facility Definitions

```
$_o
$   IF RETURN_SYMBOL .EQS. "N" -
    THEN WRITE SYS$OUTPUT "Oracle Trace facility definition could not
    be installed"
$
$ !+
$ ! Delete the callback procedures from sys$update.
$ !-
$   DELETE/NOLOG VMI$ROOT:[SYSUPD]EPC$VMSINSTAL_CALLBACK.COM;*
$ !
```

5.4.3 Inserting Definitions Without KITINSTAL.COM

If your installation procedure is not VMSINSTAL compliant, you can use the INSERT DEFINITION command to insert a facility definition (previously created with the EXTRACT DEFINITION command) into the Oracle Trace administration database on a target system or VMScluster. You must have OpenVMS BYPASS or SYSPRV privilege to insert a facility definition for a facility that already exists in the target Oracle Trace administration database and that was created by another user.

The following command inserts the facility definition for the ATM_SAMPLE facility into the Oracle Trace administration database on the target system:

```
$ COLLECT INSERT DEFINITION ATM_FAC_DEF.EPC$DEF
%EPC-S-FACCRE, Facility definition ATM_SAMPLE was created
```

Registered facilities insert their facility definitions into the Oracle Trace administration database as part of their normal installation procedure. However, if Oracle Trace does not exist on the target system (the INSERT DEFINITION command fails), the facility definitions can be stored in the **Oracle Trace facility library**. The installation procedure must perform the following steps:

1. Create the facility library SYS$COMMON:[SYSLIB]EPC$FACILITY.TLB if it does not already exist[3]:

   ```
   $ LIBRARY/TEXT/CREATE=(BLOCK=3,HISTORY=32767,KEYSIZE=39) -
   _$ SYS$COMMON:[SYSLIB]EPC$FACILITY.TLB
   ```

[3] EPC$FACILITY.TLB is shipped with Version 5.2 or higher of the OpenVMS operating system.

2. Use the OpenVMS librarian to insert the binary facility definition file into the Oracle Trace facility library (note that the module name is a concatenation of the facility name and the version string):

```
$ LIBRARY/TEXT/INSERT/REPLACE/MODULE=ATM_SAMPLEV1.0 -
_$SYS$COMMON:[SYSLIB]EPC$FACILITY.TLB ATM_FAC_DEF.EPC$DEF
```

During the Oracle Trace installation, the system manager can run EPC$INSERT.COM (in EPC$EXAMPLES:) to automatically insert all of the facility definitions that exist in SYS$SHARE:EPC$FACILITY.TLB into the administration database.

If you plan to remove Oracle Trace from your system but want to retain the facility definitions stored in your Oracle Trace administration database (in case you later decide to reinstall Oracle Trace), you can extract the facility definitions from your Oracle Trace administration database and store them in the Oracle Trace facility library.

Use the /LIBRARY qualifier to the INSERT DEFINITION command to insert a binary facility definition file into the Oracle Trace facility library. The following example extracts the facility definition for the ATM_SAMPLE facility from the Oracle Trace administration database and stores it in the facility library:

```
$ COLLECT EXTRACT DEFINITION ATM_SAMPLE ATM_FAC.DEF /VERSION="V1.0"
$ COLLECT INSERT DEFINITION ATM_FAC_DEF.EPC$DEF /LIBRARY
```

5.5 Displaying Facility Definitions

You can display information about the facility definitions stored in the Oracle Trace administration database using the SHOW DEFINITION command. The command takes one argument: the name of a facility. If you do not specify a facility name, Oracle Trace displays information on all of the facilities defined on the system.

You can specify the amount of information to display about a facility definition by using the FORMAT qualifier to the SHOW DEFINITION command. There are two valid format types:

- FULL
- NAMES_ONLY (default)

The following example writes information in the NAMES_ONLY format about all of the facility definitions on the system to the file FAC_NAMES.DAT:

```
$ COLLECT SHOW SELECTION /FORMAT=NAMES_ONLY /OUTPUT=FAC_NAMES.DAT
```

5.5.1 FULL Format

If you specify /FORMAT=FULL with the SHOW DEFINITION command, Oracle Trace displays a full description of facility definitions stored in the Oracle Trace administration database. You can display the description of a single definition if you include its name and version code on the command line. For example, the following command displays the complete description of the ATM_SAMPLE definition:

```
$ COLLECT SHOW DEFINITION ATM_SAMPLE /VERSION="V1.0" /FORMAT=FULL
```

The FULL format display includes the following information:

- Name and version of the facility
- Facility ID number
- Creation date of the facility definition
- User name of the facility definition's creator
- Description of events, items, groups, and classes defined for the facility
- Relationships to events and items in other facilities

Example 5–2 shows a segment of the DEC ACMS facility definition including its relationship to an Oracle Rdb transaction event.

Example 5–2 Display for SHOW DEFINITION Including a Related Facility Item

```
24-OCT-1995 17:06       Facility Definition Information                Page 1
Full Report                                                 Oracle Trace V2.2-0
Facility:      ACMS
Number:        253
Version:       V3.3-0      DEFAULT
Creation Date: 12-AUG-1995 12:18
Created By:    Oracle Trace

Events:

  Event Name       Event ID  Report Header
  ---------------  --------  ----------------
  TASK             1         TASK
  EXCHANGE_STEP    2         EXCHANGE_STEP
  PROCESSING_STEP  3         PROCESSING_STEP
  REMOTE_REQUEST   4         REMOTE_REQUEST
  PROCEDURE_CALL   5         PROCEDURE_CALL
  TASK_WAIT        7         TASK_WAIT
  FORMS_REQUEST    8         FORMS_REQUEST
  FORMS_ENABLE     9         FORMS_ENABLE
  TRANSACTION      10        TRANSACTION
  SUB_RESPONSE     11        SUB_RESPONSE
  APL_RESPONSE     12        APL_RESPONSE
  COMPRESSED_MSG   13        COMPRESSED_MSG
Relationships:

 Relation  Table     Source    Source    Source    Target    Target    Target
 ship      Name      Facility  Event     Item      Facility  Event     Item
 Name                Name      Name      Name      Name      Name      Name
 --------  --------  --------  --------  --------  --------  --------  --------
 RDBEXPER  RDBX$WOR  ACMS      PROCEDUR  PROCEDUR  ACMS      PROCEDUR  PROCEDUR
 T         KLOAD               E_CALL    E_INDEX             E_CALL    E_INDEX
 _APPLICA
 TION
```

(continued on next page)

Example 5–2 (Cont.) Display for SHOW DEFINITION Including a Related Facility Item

```
   Comment                        Hierarchy  S1
                                  Level
   -----------------------------  ---------  --------------------
                                  0
Relation  Table     Source    Source    Source    Target    Target    Target
ship      Name      Facility  Event     Item      Facility  Event     Item
Name                Name      Name      Name      Name      Name      Name
--------  --------  --------  --------  --------  --------  --------  -------
RDBEXPER  RDBX$WOR  ACMS      PROCEDUR  PROCEDUR  ACMS      PROCEDUR  APPL
T         KLOAD               E_CALL    E_INDEX             E_CALL    _SPEC
_APPLICA
TION
_NAME
  .
  .
  .
```

Example 5–3 shows a complete sample of the display produced with the SHOW DEFINITION /FORMAT=FULL command. The "x" in the Position field indicates that the items are default item collected by Oracle Trace and that the image does not need to allocate space for the item in its event record buffer. A zero (0) in the maximum size column indicates that the item is the maximum supported by the data type. See Chapter 4 for descriptions of Oracle Trace data structures.

Example 5–3 Display for SHOW DEFINITION Using the Full Format

```
28-AUG-1995 17:02       Facility Definition Information                Page 1
Full Report                                                 Oracle Trace V2.2-0

Facility:      ATM_SAMPLE
Number:        4094
Version:       V1.0   DEFAULT
Creation Date: 25-AUG-89 14:17
Created By:    SYSTEM

Events:

  Event Name       Event ID  Report Header
  --------------   --------  ---------------
  ERROR_DISPLAY    1         ERROR_DISPLAY
  BALANCE_EVENT    2         BALANCE_EVENT
  DEPOSIT_EVENT    3         DEPOSIT_EVENT
  WITHDRAW_EVENT   4         WITHDRAW_EVENT
```

(continued on next page)

Example 5–3 (Cont.) Display for SHOW DEFINITION Using the Full Format

```
Items:

  Item Name       Item  Datatype  Max.   Usage    Item Report   Report  Char Rad
                    ID            Size   Type     Header        Width
  --------------  ----  --------  -----  -------  ------------  ------  ---- ---
  BIO              101  LONGWORD      4  COUNTER  BUFFERED IO     11    PRT  DEC
  DIO              102  LONGWORD      4  COUNTER  DIRECT IO       11    PRT  DEC
  PAGEFAULTS       103  LONGWORD      4  COUNTER  PAGEFAULTS      11    PRT  DEC
  PAGEFAULT_IO     104  LONGWORD      4  COUNTER  PAGEFAULT       11    PRT  DEC
                                                  IOs
  CPU              105  LONGWORD      4  COUNTER  CPU TIME        11    PRT  DEC
  CURRENT_PRIO     106  WORD          2  LEVEL    CURRENT PRIO    6     PRT  DEC
  VIRTUAL_SIZE     107  LONGWORD      4  LEVEL    VIRTUAL SIZE    11    PRT  DEC
  WS_SIZE          108  LONGWORD      4  LEVEL    WORKING SET     11    PRT  DEC
                                                  SIZ
  WS_PRIVATE       109  LONGWORD      4  LEVEL    PRIVATE WS      11    PRT  DEC
  WS_GLOBAL        110  LONGWORD      4  LEVEL    GLOBAL WS       11    PRT  DEC

Item Groups:

  Item Group Name:  RESOURCE_ITEMS

  Item Name
  -----------------
  BIO
  DIO
  PAGEFAULTS
  PAGEFAULT_IO
  CPU
  CURRENT_PRIO
  VIRTUAL_SIZE
  WS_SIZE
  WS_PRIVATE
  WS_GLOBAL

  Class:  ALL

Event Name:    BALANCE_EVENT

 Record Type       Item Name        Position
 ---------------  --------------  --------
 START EVENT       BIO                x
```

(continued on next page)

Example 5–3 (Cont.) Display for SHOW DEFINITION Using the Full Format

```
28-AUG-1995 17:02       Facility Definition Information                 Page 2
Full Report                                                 Oracle Trace V2.2-0

 Record Type       Item Name        Position
 ---------------   --------------   --------
                   DIO                 x
                   PAGEFAULTS          x
                   PAGEFAULT_IO        x
                   CPU                 x
                   CURRENT_PRIO        x
                   VIRTUAL_SIZE        x
                   WS_SIZE             x
                   WS_PRIVATE          x
                   WS_GLOBAL           x

 END EVENT         BIO                 x
                   DIO                 x
                   PAGEFAULTS          x
                   PAGEFAULT_IO        x
                   CPU                 x
                   CURRENT_PRIO        x
                   VIRTUAL_SIZE        x
                   WS_SIZE             x
                   WS_PRIVATE          x
                   WS_GLOBAL           x

Event Name:    DEPOSIT_EVENT

 Record Type       Item Name        Position
 ---------------   --------------   --------
 START EVENT       BIO                 x
                   DIO                 x
                   PAGEFAULTS          x
                   PAGEFAULT_IO        x
                   CPU                 x
                   CURRENT_PRIO        x
                   VIRTUAL_SIZE        x
                   WS_SIZE             x
                   WS_PRIVATE          x
                   WS_GLOBAL           x
```

(continued on next page)

Example 5–3 (Cont.) Display for SHOW DEFINITION Using the Full Format

```
 END EVENT         BIO                x
                   DIO                x
                   PAGEFAULTS         x
                   PAGEFAULT_IO       x
                   CPU                x
                   CURRENT_PRIO       x
                   VIRTUAL_SIZE       x
                   WS_SIZE            x
                   WS_PRIVATE         x
                   WS_GLOBAL          x


28-AUG-1995 17:02       Facility Definition Information                Page 3
Full Report                                                 Oracle Trace V2.2-0

Event Name:    ERROR_DISPLAY

 Record Type       Item Name        Position
 ---------------   --------------   --------
 POINT EVENT       BIO                 x
                   DIO                 x
                   PAGEFAULTS          x
                   PAGEFAULT_IO        x
                   CPU                 x
                   CURRENT_PRIO        x
                   VIRTUAL_SIZE        x
                   WS_SIZE             x
                   WS_PRIVATE          x
                   WS_GLOBAL           x

Event Name:    WITHDRAW_EVENT

 Record Type       Item Name        Position
 ---------------   --------------   --------
 START EVENT       BIO                 x
                   DIO                 x
                   PAGEFAULTS          x
                   PAGEFAULT_IO        x
                   CPU                 x
                   CURRENT_PRIO        x
                   VIRTUAL_SIZE        x
                   WS_SIZE             x
                   WS_PRIVATE          x
                   WS_GLOBAL           x
```

(continued on next page)

Example 5–3 (Cont.) Display for SHOW DEFINITION Using the Full Format

```
END EVENT         BIO                x
                  DIO                x
                  PAGEFAULTS         x
                  PAGEFAULT_IO       x
                  CPU                x
                  CURRENT_PRIO       x
                  VIRTUAL_SIZE       x
                  WS_SIZE            x
                  WS_PRIVATE         x
                  WS_GLOBAL          x
```

5.5.2 NAMES_ONLY Format

If you specify /FORMAT=NAMES_ONLY with the SHOW DEFINITION command, Oracle Trace lists the names of the facilities defined on your system. This format is useful to determine the latest version of a facility installed on your system. The report also shows the collection classes that are available for each facility. This is useful to check before creating a facility selection (see Section 2.3). If more than one version of a facility definition exists, the facilities are ordered by creation date, with the newest (the default definition) at the top of the list.

Example 5–4 shows a sample of the display produced with the SHOW DEFINITION /FORMAT=NAMES_ONLY command.

Example 5–4 Display for SHOW DEFINITION Using the Names Only Format

```
25-AUG-1995 14:28       Facility Definition Information              Page 1
Names Only Report                                          Oracle Trace V2.2-0

 Facility:            Version:    Creation Date:      Class:
 -------------------  ----------  ------------------  -----------------
 ATM_SAMPLE           V1.0        25-AUG-89 14:17     ALL               (D)
 RDBVMS               V4.0        20-AUG-90 15:21     ALL
                                                      PERFORMANCE       (D)
                                                      RDBEXPERT
```

5.6 Facility Definition Options

The /OPTIONS qualifier to the CREATE DEFINITION command provides capabilities beyond those provided by the /EVENTS qualifier, particularly the ability to collect facility-specific items. Table 5–4 summarizes the **facility definition options**.

Table 5–4 Summary of Facility Definition Options

Option	Description
ITEM	Binds the name of a facility-specific item to a unique numeric identifier and specifies the characteristics of the data associated with that item.
GROUP	Conveniently allows you to refer to a set of items by a single name. You can use the group name in an EVENT or CLASS option to refer to all of the items within that group.
EVENT	Binds the name of an event to a unique numeric identifier and specifies the items associated with that event.
RELATE	Specifies a relationship between a pair of events and items between separate facilities.
CLASS	Binds a name to a set of events and a set of items to each event. A facility selection can specify a class name to limit data collection. Oracle Corporation recommends that application product developers define one or more of the following standard classes: • CAPACITY_PLANNING—Includes those events and items that are useful for capacity planning purposes. • DEBUGGING—Includes those events and items that are useful for tracing the execution of your application. • ERROR_LOGGING—Includes those events and items that are associated with error or exception handling routines in your application. • PERFORMANCE—Includes those events and items that are useful for application and/or database tuning. • WORKLOAD—Includes those events and items that are useful for gathering information useful for tracing the actual workload of the application and/or database management system.
DEFAULT_CLASS	Indicates which class to collect from if none is specified by the facility selection.

Note

Facility options are order-dependent. Each option must be defined before it is referenced. The order dependencies are: ITEM, GROUP, EVENT, RELATE, CLASS, and DEFAULT_CLASS.

6
Oracle Trace Commands

Oracle Trace provides a command-line interface. To use the Oracle Trace commands, preface them with the keyword COLLECT. For example:

```
$ COLLECT SHOW VERSION
Oracle Trace Version V2.2-0
$
```

For better user interface performance, you can enter the Oracle Trace command environment by entering the COLLECT command with no arguments. This eliminates binding to the history and administration databases for each command. Oracle Trace prompts you for commands until you return to DCL command level with the EXIT command. For example:

```
$ COLLECT
Trace> SHOW VERSION
Oracle Trace Version V2.2-0
Trace> EXIT
$
```

This chapter describes the format and usage of each Oracle Trace command. See *Oracle Trace Getting Started* for a quick reference guide to the syntax for each command. Table 6–1 summarizes the available Oracle Trace commands.

Table 6–1 Oracle Trace Commands

Command	Description
@ (Execute Procedure)	Executes the commands in a command file as if you had typed them at the Trace> prompt.
CANCEL COLLECTION	Stops data collection for an active collection. If a collection is pending, it removes the collection from the schedule.

(continued on next page)

Table 6–1 (Cont.) Oracle Trace Commands

Command	Description
CREATE DEFINITION [1]	Creates a facility definition in the Oracle Trace administration database.
CREATE SELECTION	Creates a facility selection in the Oracle Trace administration database. Selection names must be unique in the Oracle Trace administration database. If you use the /REPLACE qualifier, your new facility selection replaces the old selection of the same name.
DELETE DEFINITION	Deletes a facility definition from the Oracle Trace administration database.
DELETE SELECTION	Deletes a facility selection from the Oracle Trace administration database.
EXIT	Exits from Oracle Trace and returns to the DCL command level.
EXTRACT DEFINITION	Extracts a facility definition from the Oracle Trace administration database and stores it in a binary file.
HELP	Displays requested information about the Oracle Trace commands.
INSERT DEFINITION	Inserts a facility definition in a binary format into the Oracle Trace administration database.
SCHEDULE COLLECTION	Schedules data collection based on the specified qualifiers.
SET ACCESS	Grants or denies access to an Oracle Trace database.
SET HISTORY	Changes the history database that Oracle Trace uses for the SHOW HISTORY command or creates a new history database.
SHOW ACCESS	Displays a report of users who have access to an Oracle Trace database.
SHOW COLLECTION	Shows data collection information in one of two formats: BRIEF or FULL.
SHOW DEFINITION	Shows information about one or more facility definitions registered in the Oracle Trace administration database in one of two formats: FULL or NAMES_ONLY.

[1]Separate reference sections on the facility definition options, CLASS, DEFAULT_CLASS, EVENT, GROUP, ITEM, and RELATE, follow the CREATE DEFINITION command section.

(continued on next page)

Table 6–1 (Cont.) Oracle Trace Commands

Command	Description
SHOW HISTORY	Shows all error and/or informational messages that have occurred during one or all data collections active on your system.
SHOW REGISTER	Shows the individual processes for which data can be collected.
SHOW SELECTION	Displays a facility selection in one of three formats: BRIEF, FULL, or NAMES_ONLY.
SHOW VERSION	Shows the version number of Oracle Trace installed on your system.
SPAWN	Creates a subprocess of the current process.
STOP SYSTEM	Stops Oracle Trace and interrupts any active data collection.

@ (Execute Procedure)

The at sign (@) means execute, just as in DCL. When you type @ and the name of an indirect command file, Oracle Trace executes the statements in that file as if you had typed them one at a time at the Trace> prompt. The command file must be an OpenVMS text file that contains Oracle Trace commands.

Format

@ file-spec

Parameter

file-spec
The name of the indirect command file. You can specify a full OpenVMS file specification, a file name, or a logical name. If you specify a file name, Oracle Trace looks in your current default OpenVMS directory for a file by that name. The file must contain valid Oracle Trace commands. The default file type is COM.

Description

This command is useful because it allows you to prepare a sequence of commands that you use often and that must be repeated identically each time you issue them. For example, if you generate weekly or monthly reports, you will want the same parameters defined each time.

Example

```
Trace> @DISK$USER1:[SMITH.COMS]SCHEDULE_A_COLLECTION.COM
```

Executes the commands in the file SCHEDULE_A_COLLECTION.COM, where the command file contains the following lines:

```
CREATE SELECTION VAX_INFO /OPTIONS
  FACILITY RDBVMS
  FACILITY ACMS/CLASS=ALL
  EXIT
SCHEDULE COLLECTION MY_TEST MY_DATA.DAT -
  /SELECTION=VAX_INFO -
  /BEGIN=11:00 /END=12:00
EXIT
```

CANCEL COLLECTION

Stops one or more active or pending collections.

Format

CANCEL COLLECTION [collection-name]

Command Qualifiers	Defaults
/[NO]CONFIRM	/CONFIRM
/SELECTION=selection-name	/SELECTION=*

Parameter

collection-name
The name of the data collection that you want to cancel. The collection name is a unique text string that identifies one currently scheduled collection. You can use an asterisk (*) as a wildcard to cancel all collections.

The collection name is optional; however, you must specify either the collection name or a facility selection name (with the /SELECTION qualifier). If you specify both a collection name and a selection name, the facility selection must have been specified in the original SCHEDULE COLLECTION command or Oracle Trace issues an error message and does not cancel any data collection.

Qualifiers

/CONFIRM (default)
/NOCONFIRM
Specifies whether Oracle Trace prompts you to confirm the cancellation of each collection (similar to the OpenVMS DCL command DELETE/CONFIRM).

If you use the /CONFIRM qualifier, Oracle Trace identifies each collection that matches the specified collection name and asks if you want to cancel it. If you use /NOCONFIRM, Oracle Trace cancels the specified collection(s) without prompting for confirmation.

/SELECTION=selection-name
Specifies a facility selection that has one or more collections active or pending on the current system. If you specify an asterisk (*) as a wildcard character in place of the selection name, Oracle Trace cancels all data collection.

The /SELECTION qualifier is optional; however, you must specify either a facility selection name or a collection name. If not, Oracle Trace issues an error message indicating that the command is ambiguous.

If you specify both a facility selection name and a collection name, the selection must have been specified in the original SCHEDULE COLLECTION command or Oracle Trace issues an error message and does not cancel any data collection.

Description

In a cluster, the CANCEL COLLECTION command cancels data collection either locally or cluster-wide, depending on how the collection was originally scheduled. If you scheduled data collection with the /NOCLUSTER qualifier, CANCEL COLLECTION cancels the collection locally. If you scheduled data collection with the /CLUSTER qualifier, CANCEL COLLECTION cancels the collection on all nodes in the cluster.

If both a collection name and a facility selection name are supplied, they must be the same as those given in the original SCHEDULE COLLECTION command.

To cancel data collection, you must be the originator of the collection, or have OpenVMS BYPASS or SYSPRV privilege.

Examples

1.
```
$ COLLECT CANCEL COLLECTION MONDAYS_TEST /CONFIRM
Cancel MONDAYS_TEST [N]: YES
%EPC-S-SCHED_CANCEL, Data collection MONDAYS_TEST is cancelled
```

   Cancels the data collection named MONDAYS_TEST.

2.
```
$ COLLECT CANCEL COLLECTION /SELECTION=ACMS_DATA /NOCONFIRM
%EPC-S-SCHED_CANCEL, Data collection MY_TEST is cancelled
%EPC-S-SCHED_CANCEL, Data collection TUESDAYS_TEST is cancelled
%EPC-S-SCHED_CANCEL, Data collection WEDNESDAYS_TEST is cancelled
```

   Cancels all data collection associated with the facility selection ACMS_ DATA. In this example, three collections are cancelled.

3.
```
$ COLLECT CANCEL COLLECTION MY_TEST /SELECTION=JOES_DATA /NOCONFIRM
%EPC-E-SCHED_XFAILED, Cancel collection operation failed
%EPC-E-SCHED_NTFST, No collections found matching collection MY_TEST
 in selection JOES_DATA
```

   Attempts to cancel the data collection named MY_TEST. However, MY_ TEST was not scheduled with the facility selection JOES_DATA, so Oracle Trace issues an error message and does not cancel the collection.

CREATE DEFINITION

Creates a facility definition and defines its events.

Format

CREATE DEFINITION facility-name facility-id

Command Qualifiers	Defaults
/EVENTS=(event-name[, . . .])	See text
/[NO]OPTIONS[=file-spec]	/NOOPTIONS
/[NO]REPLACE	/NOREPLACE
/VERSION="version-code"	See text

Parameters

facility-name
The name of the facility that you want to define. The maximum length of the facility name is 27 characters.

facility-id
A unique numeric identifier for the facility. Facility IDs in the range 1 to 2047 are reserved to Oracle Corporation. You can define facilities with IDs in the range 2048 to 4095. Note that the facility ID must match the ID specified in the EPC$INIT service routine call (in the program source code).

All versions of a given facility use the same facility ID.

Qualifiers

/EVENTS=(event-name[, . . .])
Specifies the events that the facility includes. The maximum length of the event name is 15 characters. If you use the /EVENTS qualifier to define events, you can specify only duration events that collect the resource utilization items; you cannot specify additional items.

The event name must be a valid OpenVMS name (A–Z, 0–9, dollar signs ($), and underscores (_)). The name must begin with a letter and cannot end with a dollar sign or underscore.

The /EVENTS and /OPTIONS qualifiers are mutually exclusive, but you must have one or the other.

/OPTIONS[=file_spec]
/NOOPTIONS (default)
Allows you to specify events and items in the facility definition either in an options file or interactively. When you specify events and items interactively, the Option> prompt displays and you can enter your options.

Regardless of the approach you choose, facility options are order-dependent and you must define each option must before you reference it.

The /EVENTS and /OPTIONS qualifiers are mutually exclusive, but you must have one or the other. Use the /OPTIONS qualifier to define either point or duration events that collect items in place of or in addition to the resource utilization items.

Following the CREATE DEFINITION command are individual descriptions of the facility definition options.

/REPLACE
/NOREPLACE (default)
Specifies that the current facility definition replaces any previously existing facility definition with the same facility name and version code. If a facility definition exists and you attempt to redefine it without using the /REPLACE qualifier, Oracle Trace issues an error message and does not store the new definition.

If no facility definition with the same name and version code exists, Oracle Trace ignores the /REPLACE qualifier.

/VERSION="version-code"
Specifies the version of the facility. The name of the version can be any printable string up to 10 characters. You must enclose the text string with quotation marks (" "). This is a required qualifier and must match the version specified in the EPC$INIT service routine call.

Description

The CREATE DEFINITION command creates a facility definition that is stored in the Oracle Trace administration database, which is available cluster-wide. The definition is retained until you explicitly delete it using the DELETE DEFINITION command. Oracle Trace facility definitions consist of:

- Class definitions (optional)
- Creation date
- Creator's user name
- Event definitions (optional)

- Event names
- Facility ID
- Facility name
- Item definitions (optional)
- Item group definitions (optional)
- Version code

Use the CREATE DEFINITION command to define the facility name, ID, version code, and event names. Oracle Trace automatically records your user name and the creation date as part of the facility definition.

You need OpenVMS SYSPRV or BYPASS privilege to create a registered facility definition.

Examples

1.
```
$ COLLECT CREATE DEFINITION DATA_ENTRY 2052 /EVENTS=(INIT,STOP) -
_$ /VERSION="V1.0"
```

   Creates the facility definition for V1.0 of the DATA_ENTRY facility and specifies that the default resource utilization items should be collected for the events INIT and STOP. The facility ID for DATA_ENTRY is 2052.

2.
```
$ COLLECT CREATE DEFINITION MY_APPLICATION 2050 -
_$ /VERSION="T1.0-3" /OPTIONS=MY_APP_OPTIONS.TXT
```

   Creates the facility definition for T1.0-3 of the MY_APPLICATION facility and specifies that the facility definition options listed in the file MY_APP_OPTIONS.TXT be used. The facility ID for MY_APPLICATION is 2050.

3.
```
$ COLLECT CREATE DEFINITION /OPTIONS NEW_TOOL 2049 /VERSION="T1.0-1"
Option> ITEM IMAGE_NAME TEXT/SIZE=256
Option> EVENT INVOCATION/ITEMS=IMAGE_NAME
Option> CTRL/Z
$
```

   Creates a facility definition for T1.0-1 of the NEW_TOOL facility. The facility has one event (INVOCATION) which has one item (IMAGE_NAME) associated with it. The facility ID for NEW_TOOL is 2049.

CREATE DEFINITION Options—ITEM

The ITEM option identifies and describes the characteristics of a data item that the facility can collect.

Format

ITEM item-name datatype

Command Qualifiers	Defaults
/CHARACTERISTICS=(characteristic[, . . .])	See text
/IDENTIFIER=item-id	Next unused item-id
/RADIX=base-system	/RADIX=DECIMAL
/REPORT_HEADER="text"	/REPORT_HEADER=item-name
/REPORT_WIDTH=number-of-spaces	See text
/SIZE=n-bytes	See text
/USAGE_TYPE=usage-type	None

Parameters

item-name

Specifies a text string that names the item. The string must be a valid OpenVMS name (A–Z, 0–9, dollar signs ($), and underscores (_)). The name must begin with a letter, cannot end with a dollar sign ($) or underscore (_), and cannot be an Oracle Rdb reserved word. The maximum length of the item name is 15 characters.

datatype

Specifies the data type of the item. Table 6–2 lists the valid data types.

Table 6–2 ITEM Data Types

Data Type	Description
ASCIC	Varying-length counted string of up to 255 bytes
ASCIW	Varying-length counted string of up to 16,383 bytes
BYTE	Signed byte
FIXED_ASCIC	Fixed-length counted string of up to 255 bytes

(continued on next page)

Table 6–2 (Cont.) ITEM Data Types

Data Type	Description
LONGWORD	Signed longword (4 bytes)
QUADWORD	Signed quadword (8 bytes)
WORD	Signed word (2 bytes)

Note that the ASCIC and FIXED_ASCIC types consist of a string up to 256 characters, where the first byte is the size and the remainder is the actual string. For the ASCIW type, the first 2 bytes are the length, followed by the string.

Table 6–3 lists the statistics in Summary reports and their format.

Table 6–3 Summary Report Statistics and Their Format

Statistic	Elapsed Time	Others
Minimum	ZZZZZ9.99	ZZZZZZZZ9
Maximum	ZZZZZ9.99	ZZZZZZZZ9
Mean	ZZZZZ9.99	ZZZZZ9.99
Std Dev[1]	ZZZZZ9.99	ZZZZZ9.99
95 Prct[2]	ZZZZZ9.99	ZZZZZ9.99
Total	ZZZZZZZZ9	ZZZZZZZZ9
Count[3]	ZZZZZZZZ9	

[1]Std Dev is Standard Deviation.

[2]95 Prct is 95th Percentile.

[3]Because the Count is the same for all report items, it is only displayed for the first report item.

Z—represents a numeric character 0 through 9 for which leading zeros are replaced with blanks. Negative values display a minus sign to the left of the most significant digit.
9—represents a numeric character 0 through 9 that is replaced by asterisks (***) in reports when the numeric value overflows the field.

The Statistics are followed by a blank line.

Qualifiers

/CHARACTERISTICS=(characteristic[, . . .])

Valid characteristics are [NON]PRINTABLE and [NO]CRLF_INTERPRET.

PRINTABLE or NONPRINTABLE specifies whether the contents of an item can be displayed.

CRLF_INTERPRET or NOCRLF_INTERPRET specify whether Oracle Trace should search the string for a carriage-return and line-feed combination for wrapping purposes. If a carriage-return and line-feed are found contiguously, the text of the item is wrapped and all wrap counters are reset. If a single carriage-return or line-feed is found, the text will wrap when displayed but the wrap counters will not be reset. This can cause the output to wrap in unexpected places.

If both NONPRINTABLE and CRLF_INTERPRET are defined, the item will not be displayed on the report.

The default characteristics are PRINTABLE and NOCRLF_INTERPRET.

/IDENTIFIER=item-id
An integer between 1 and 100 that specifies a unique identifier for the item. (Identifiers 101 to 128 are reserved for Oracle Trace defined items.) The default is the next unused ID between 1 and 100.

/RADIX=base-system
Either HEXADECIMAL or DECIMAL that specifies the base of the number system used for numerical items. The default radix is DECIMAL.

/REPORT_HEADER="text"
A 1- to 15-character text string that specifies a heading to display when creating reports using this item. The default is the name of the item.

/REPORT_WIDTH=number-of-spaces
An integer that specifies the width of the column to use when displaying item values in a report. The default width depends on the item data type, as listed in Table 6–4.

Table 6–4 ITEM Default Report Widths

Data Type	Width
ASCIC	80 columns
ASCIW	80
BYTE	4
FIXED_ASCIC	80
LONGWORD	11
QUADWORD	32
WORD	6

/SIZE=n-bytes
An integer that specifies the maximum size in bytes of the ASCIC, FIXED_ ASCIC, and ASCIW data types. The /SIZE qualifier is required for these data types and ignored for all others.

/USAGE_TYPE=usage-type
One of the keywords in Table 6–5 that specifies how the data is to be used.

Table 6–5 ITEM Usage Types

Usage Type	Description
COUNTER	Typically a running count or total; its value either always increases or always decreases for the duration of the event. In an Oracle Trace Summary Report, for items with usage type COUNTER, Oracle Trace displays the difference between the start and end values.
LEVEL	A meter or gauge that indicates the current value of some metric; its value may increase or decrease over the duration of the event.
PERCENT	A percentage; similar to LEVEL, except that it has upper and lower limits of 100 and 0, respectively.
PRIVATE	Facility-defined data that does not fall into one of the other usages. Oracle Trace does not attempt to display this item on Oracle Trace reports.
TEXT	Text characters.

Valid usage types depend on the data type of the item, as shown in Table 6–6.

Table 6–6 ITEM Usage Types by Data Type

Data Type	Valid Usage Types	Default
ASCIC	TEXT, PRIVATE	TEXT
ASCIW	TEXT, PRIVATE	TEXT
BYTE	COUNTER, LEVEL, PERCENT, PRIVATE	COUNTER
FIXED_ASCIC	TEXT, PRIVATE	TEXT
LONGWORD	COUNTER, LEVEL, PERCENT, PRIVATE	COUNTER
QUADWORD	COUNTER, LEVEL, PERCENT, PRIVATE	COUNTER
WORD	COUNTER, LEVEL, PERCENT, PRIVATE	COUNTER

CREATE DEFINITION Options—GROUP

The GROUP option allows you to refer to a set of items by a single name. You can use a group name in the /ITEMS qualifier of an EVENT or CLASS option to refer to all of the items within that group.

Format

GROUP group-name

Command Qualifier	Default
/ITEMS=(item-name[, . . .])	None

Parameter

group-name
Specifies the name of the group. The group name must be a valid OpenVMS name and cannot end with a dollar sign ($) or underscore (_). The maximum length of the group name is 15 characters.

Qualifier

/ITEMS=(item-name[, . . .])
Specifies the items contained in the group. This is a required qualifier.

CREATE DEFINITION Options—EVENT

The EVENT option identifies and describes the characteristics of an event including the set of items to collect and (optionally) the items to collect when the event is a point event, start event, or end event. By default, Oracle Trace places each event in the facility class named ALL.

Format

EVENT event-name

Command Qualifiers	Defaults
/END_EVENT=(item-name[, . . .])	See text
/FIRST_SEGMENT_SIZE=n-bytes	None
/IDENTIFIER=event-id	Next unused event ID
/[NO]ITEMS=(item-name[, . . .])	/NOITEMS
/POINT_EVENT=(item-name[, . . .])	None
/REPORT_HEADER="text"	/REPORT_HEADER=event-name
/SEGMENT_SIZE=n-bytes	None
/START_EVENT=(item-name[, . . .])	See text

Parameter

event-name
Specifies the name of the event. The event name must be a valid OpenVMS name (A–Z, 0–9, dollar signs ($), and underscores (_)). The name must begin with a letter and cannot end with a dollar sign ($) or underscore (_). The maximum length of the event name is 15 characters.

Qualifiers

/END_EVENT=(item-name[, . . .])
Specifies the items that Oracle Trace collects at the end of a duration event. If you omit the /END_EVENT qualifier, Oracle Trace uses the /ITEMS qualifier to define the items to collect at the end of a duration event.

The /START_EVENT and /END_EVENT pair of qualifiers are mutually exclusive to the /POINT_EVENT qualifier. An event cannot be both point and duration.

/FIRST_SEGMENT_SIZE=n-bytes
Specifies the size of the first string segment for the event relation in the formatted database. If you do not specify a size for the first segment, the Oracle Trace formatting component uses 64 bytes.

See the *Oracle Trace Reporter User's Guide*, OPTIMIZATION PARAMETERS under the FORMAT command, and the Oracle Rdb Database Format Appendix for more information about string segmentation.

/IDENTIFIER=event-id
Specifies an integer between 1 and 128 that uniquely identifies the event. The default is the next unused event identifier between 1 and 128.

/ITEMS=(item-name[, . . .])
/NOITEMS (default)
Defines the set of all items to collect for an event. The default is /NOITEMS. This is a required qualifier.

/POINT_EVENT=(item-name[, . . .])
Specifies the items that Oracle Trace collects for a point event. Specify only items that have been defined using the /ITEMS qualifier.

The /POINT_EVENT qualifier is mutually exclusive to the /START_EVENT and /END_EVENT pair of qualifiers. An event cannot be both point and duration.

/REPORT_HEADER="text"
A 1- to 15-character text string that specifies a heading to display when creating reports using this event. The default report heading is the name of the event.

/SEGMENT_SIZE=n-bytes
Specifies the size of the remaining string segments (the first having been specified with the /FIRST_SEGMENT_SIZE qualifier) for the event relation in the formatted database. If you do not specify a size for the segments, the Oracle Trace formatting component uses 128 bytes.

See the *Oracle Trace Reporter User's Guide*, OPTIMIZATION PARAMETERS under the FORMAT command, and the Oracle Rdb Database Format Appendix for more information about string segmentation.

/START_EVENT=(item-name[, . . .])
Specifies the items that Oracle Trace collects at the beginning of a duration event. If you omit the /START_EVENT qualifier, Oracle Trace uses the /ITEMS qualifier to define the items to collect at the beginning of a duration event.

The /START_EVENT and /END_EVENT pair of qualifiers are mutually exclusive to the /POINT_EVENT qualifier. An event cannot be both point and duration.

Description

In the /ITEMS, /START_EVENT, /END_EVENT, and /POINT_EVENT, qualifiers, items are collected in the order specified. You can specify:

- Any item or item group that has already been defined.
- A predefined resource utilization item.
- The predefined item group RESOURCE_ITEMS, which collects all resource utilization items.
- An asterisk (*) as a wildcard symbol, meaning all of the ITEM options in the facility definition as well as resource utilization items. (This is for the /ITEMS qualifier only.)

CREATE DEFINITION Options—CLASS

The CLASS option defines a new class if the specified class does not already exist and binds an event (with a subset of its associated items) to a class.

Format

CLASS class-name event-name

Command Qualifier	Default
/[NO]ITEMS=(item-name[, . . .])	/NOITEMS

Parameters

class-name
Specifies the name of the class. A class name must be a unique 1- to 32-character string consisting of alphanumeric characters, dollar signs ($), and underscores (_) . The string must be unique within the context of the facility definition. The class name ALL is reserved.

event-name
Specifies the name of an event to add to the class. To define a class that contains multiple events, specify a separate CLASS option for each event.

Qualifier

/ITEMS=(item-name[, . . .])
/NOITEMS (default)
Specifies a set of items or item groups to collect for an event when the selection specifies this class. You can supply an asterisk (*) as a wildcard symbol to collect all of the items specified in the EVENT /ITEMS qualifier. The default is /NOITEMS.

CREATE DEFINITION Options—DEFAULT_CLASS

The DEFAULT_CLASS option designates which class to collect when the facility selection does not specify a class. If you omit the DEFAULT_CLASS option, the ALL class is designated as the default class. By default, Oracle Trace places each event in the facility class named ALL.

Format

DEFAULT_CLASS class-name

Parameter

class-name
Specifies the name of the default class. It must be the class ALL or a previously defined class. Oracle Corporation recommends that the class most frequently referred to by facility selections be specified as the default class.

CREATE DEFINITION Options—RELATE

The RELATE option defines a new relationship between two events or two items. The two events or items may be in separate facilities. Oracle Trace stores relationship information in the facility definition and in a table in the formatted database

Format

RELATE relationship-name table-name source-facility-name source-event-name source-item-name target-facility-name target-event-name target-item-name

Command Qualifiers	Defaults
/[NO]COMMENT="string"	/NOCOMMENT
/HIERARCHY_LEVEL	/HIERARCHY_LEVEL=0
/[NO]S1="string"	/NOS1

Parameters

relationship-name
Specifies a descriptive name for the relationship between the related pair of events or items. For example, the following are relationship-names for use by Oracle Expert for Rdb:

RDBEXPERT_APPLICATION
RDBEXPERT_APPLICATION_NAME
RDBEXPERT_PROGRAM
RDBEXPERT_PROGRAM_NAME

The relationship name can be up to 255 characters in length and must be enclosed in quotation marks (" ").

table-name
Specifies the name of the table (relation) in the formatted database that contains the information (facilities, events, and items) for the relationship. The table can contain information about one or many relationships. Table 6–7 shows the layout of the table in the formatted database.

A tuple for each relationship is stored in the EPC$RELATIONSHIP_TABLE_ NAMES table. This serves as an index (directory) to all relationship tables that are stored within the formatted database. The following table describes the field in the relationship table.

Field Name	Data Type/Size	Description
TABLE_NAME	Varying String, 31 bytes	Name of the table where the relationship resides.

Field names in **Bold** signifies key fields.

The name of the table stored in the formatted database is explicitly indicated on the RELATE option of the CREATE DEFINITION command. Table 6–7 describes the fields in the relation table.

Table 6–7 Relationship Table

Field Name	Data Type/Size	Description
RELATIONSHIP_NAME	Varying String, 255 bytes	Name describing the relationship of the 2 related events.
CREATOR_FACILITY_ NAME	Varying String, 27 bytes	The facility name that created this relationship.
CREATOR_FACILITY_ VERSION	Varying String, 10 bytes	The version of the facility that created this relationship.
SOURCE_FACILITY_ NAME	Varying String, 27 bytes	The source facility name.
SOURCE_EVENT_NAME	Varying String, 15 bytes	Name describing the source event.
SOURCE_ITEM_NAME	Varying String, 15 bytes	Name describing the source item.
TARGET_FACILITY_ NAME	Varying String, 27 bytes	The target facility name.
TARGET_EVENT_NAME	Varying String, 15 bytes	Name describing the target event.

Bold signifies key fields.

(continued on next page)

Table 6–7 (Cont.) Relationship Table

Field Name	Data Type/Size	Description
TARGET_ITEM_NAME	Varying String, 15 bytes	Name describing the target item.
COMMENT	Varying String, 80 bytes	The comment describing the relationship.
HIERARCHY_LEVEL	Signed longword	Hierarchy level for the relationship. Value of zero indicates no hierarchy level specified.
S1	Varying String, 255 bytes	Character string value for the S1 parameter.

Bold signifies key fields.

A typical table name is comprised of a facility name and a descriptive name indicating the purpose of the relationships stored in the table. For example, RDBX$WORKLOAD indicates that all relationships stored in this table are used to describe relationships for producing an Oracle Expert for Rdb workload hierarchy.

The table name can be up to 31 characters in length and must:

- Begin with a letter
- Contain only uppercase or lowercase letters, digits, dollar signs ($), and underscores (_)
- End with a letter or digit

source-facility-name
Specifies the name of the source facility that contains the source-event-name of interest. The source-facility-name can be the name of the current facility or it can be the name of an external facility, but either the source-facility-name or target-facility-name must be the current facility.

source-event-name
Specifies the name of the source event within this facility to relate to another target event.

source-item-name
Specifies the name of the source item, used to join the source-event-name to the target-event-name.

target-facility-name
Specifies the name of the target facility that contains the target-event-name of interest. The target-facility-name can be the name of the current facility or it can be the name of an external facility, but either the source-facility-name or target-facility-name must be the current facility.

target-event-name
Specifies the name of the target event within the facility indicated by the target-facility-name parameter.

target-item-name
Specifies the name of the target item, used to join the target-event-name to the source-event-name.

Qualifiers

/COMMENT
/NOCOMMENT (default)
Allows you to include a comment string describing the relationship. The comment can be up to 80 characters and must be enclosed with quotation marks (" ").

/HIERARCHY_LEVEL
/HIERARCHY_LEVEL=0 (default)
Allows you to specify that multiple event relationships have an inherent hierarchy. Low values indicate that the relationship is relatively higher than other relationships in the hierarchy, whereas high values indicate that the relationship is lower in the hierarchy. A value of 1 indicates that this relationship is highest. A value of 0 indicates that no hierarchy level was specified.

/S1
/NOS1 (default)
Allows you to include a string value describing some additional characteristic about the relationship. The string can be up to 255 characters and must be enclosed with quotation marks (" "). For example, Oracle Expert for Rdb requires the value "FILESPEC" be indicated if the *target-item-name* is a file specification for the RDBEXPERT_APPLICATION_NAME or RDBEXPERT_PROGRAM_NAME relationship. If *target-item-name* is not a file specification then use the default /NOS1 qualifier.

Description

See Section 5.2 for a description of how to use the RELATE option.

CREATE SELECTION

Creates a facility selection, that is, a list of the facilities for which to collect data. You can optionally specify the class of data to collect for each facility.

Format

CREATE SELECTION selection-name

Command Qualifiers	Defaults
/[NO]COMMENT="comment-string"	/NOCOMMENT
/FACILITY=(facility-name[, . . .])	/FACILITY=*
/[NO]OPTIONS[=file-spec]	/NOOPTIONS
/[NO]REPLACE	/NOREPLACE

Parameter

selection-name
Specifies the name of the facility selection that you want to define. The selection name must be a unique 1- to 32-character string consisting of alphanumeric characters, dollar signs ($), and underscores (_). The selection name must be unique for the local system or for the entire cluster in a cluster environment.

Qualifiers

/COMMENT
/NOCOMMENT (default)
Allows you to include a comment string describing the selection. The comment can be up to 80 characters and must be enclosed with quotation marks (" ").

/FACILITY=(facility-name[, . . .])
Specifies one or more facilities for which to collect data. You can use an asterisk (*) as a wildcard to have Oracle Trace collect data for all available facilities.

Use the /FACILITY qualifier to define simple facility selections where you want to collect data from the default class for one or more facilities. If you want to specify a collection class for individual facilities, use the /OPTIONS qualifier. The /FACILITY and /OPTIONS qualifiers are mutually exclusive.

If you do not specify either the /FACILITY or the /OPTIONS qualifier, the facility selection describes the default class data for all facilities.

/OPTIONS[=file-spec]
Specifies a file that contains the details of the facility selection. The default file type for the options file is OPT.

If you enter an OpenVMS file specification with the /OPTIONS qualifier, Oracle Trace uses that file as a source for facility selection options. If you enter /OPTIONS without a file specification, Oracle Trace prompts you for the options interactively. Press CTRL/Z or type EXIT to exit from the Option> prompt.

The /FACILITY and /OPTIONS qualifiers are mutually exclusive. If you do not specify either the /FACILITY or the /OPTIONS qualifier, the facility selection describes the default class data for all facilities. See the Description section for information about facility selection options.

/REPLACE
/NOREPLACE (default)
Specifies that the current facility selection replaces any previously existing selection with the same name. If a facility selection exists and you attempt to redefine it without using the /REPLACE qualifier, Oracle Trace issues a warning message and does not store the new selection.

If no facility selection with the same name exists in the Oracle Trace administration database, the /REPLACE qualifier is ignored.

Description

Before you create a facility selection for a facility that has been instrumented with Oracle Trace calls, you must be sure that your administration database contains a facility definition for the facility. Use the SHOW DEFINITION command to verify that your administration database contains the appropriate facility definition. If it does not, ask your system manager to use the EPC$INSERT command procedure to enter the facility definition into the administration database.

Oracle Trace facility selections consist of:

- The name of the selection
- A list of facilities for which to collect data
- A collection class for each facility

CREATE SELECTION

You use the CREATE SELECTION command to choose a subset of the available facilities for which you want to collect data. You can define most general-purpose selections with the command without the /OPTIONS qualifier. For example, the following command defines the facility selection ACMS_DATA to collect data for the default class for DEC ACMS:

```
$ COLLECT CREATE SELECTION ACMS_DATA /FACILITY=ACMS -
_$/VERSION=V3.3-0
```

Oracle Corporation recommends that you always specify the version number of the facility for which you wish to collect data. If you are running multiple versions of Oracle Rdb on your system, or if you have upgraded to a newer version of Oracle Rdb, the administration database can have more than one facility definition for Oracle Rdb. You must ensure that the version number you specify in your facility selection corresponds to the version of Oracle Rdb that your process runs. You can query the facility to learn the version your process is running. For example, for the Oracle Rdb facility you could use the following command:

```
$ RMU/SHOW VERSION
Executing RMU for DEC Rdb V6.0-05
```

However, to define a more detailed facility selection, you should use an options file. The /OPTIONS qualifier allows you to specify a different collection class for each facility from which you want to collect data. The qualifier takes a file name as an argument. The options file lists each facility and the collection class you want to use. Each facility is described on a separate line in the file. If you specify /OPTIONS but do not include a file name, Oracle Trace prompts you for the options. The format of the facility description in the options file is:

FACILITY facility-name [/VERSION="version-code"] [/CLASS=class-name]

facility-name
The name of the facility for which to collect data.

version-code
A text string identifying the version of the facility. The string must be enclosed with quotation marks (" ").

class-name
The class of data that you want collected for the facility. Facilities registered with Oracle Trace can have one or more collection classes associated with them. These classes are subsets of the available events and items chosen for their significance to a specific function. For example, a facility might have specific collection classes defined for performance, workload, or capacity planning. You use the SHOW DEFINITION /FORMAT=NAMES_ONLY command to

determine what classes are available for a given facility. All facilities have the ALL class which contains all of the events and associated items for the facility. Furthermore, one of the classes is designated as the default class based on its predicted importance and frequency of use.

Note that you cannot specify more than one version or class for the same facility.

Examples

1. `$ COLLECT CREATE SELECTION ACMS_AND_RDB /FACILITY=(ACMS,RDBVMS)`

   Creates a facility selection that collects the default events for both the DEC ACMS and Oracle Rdb facilities. The selection name is ACMS_AND_RDB.

2.
```
$ COLLECT CREATE SELECTION VAX_INFO_DATA /OPTIONS
Option> FACILITY RDBVMS
Option> FACILITY ACMS/CLASS=ALL
Option> FACILITY DBMS/VERSION="V4.1"/CLASS=WORKLOAD
Option> CTRL/Z
```

   Creates a facility selection that describes the following data:

   - Default collection class data for the latest version of the Oracle Rdb facility
   - All data for the ACMS facility
   - Workload class data for Version 4.1 of the Oracle CODASYL DBMS facility

3. `$ COLLECT CREATE SELECTION TP_DATA /OPTIONS=TP_FACS.OPT`

   Creates a facility selection that collects data on the facilities listed in the options file TP_FACS.DAT.

DELETE DEFINITION

Deletes one or more facility definitions from the Oracle Trace administration database. Use this command when you install a new version of a product and want to remove the facility definition for the previous version. Deleting registered facility definitions (those in the range of 1 to 2047) is a management function that requires OpenVMS BYPASS or SYSPRV privilege to perform. For non-registered facilities, you must have created the definition, or have OpenVMS BYPASS or SYSPRV privilege.

Format

DELETE DEFINITION facility-name [, . . .]

Command Qualifiers	Defaults
/[NO]CONFIRM	/CONFIRM
/VERSION="version-code"	See text

Parameter

facility-name
Specifies the name of one or more facilities for which you want to delete the facility definition. You cannot use any wildcard characters.

Qualifiers

/CONFIRM (default)
/NOCONFIRM
Specifies whether Oracle Trace prompts you to confirm the deletion of each facility definition (similar to the OpenVMS DCL command DELETE /CONFIRM).

If you use the /CONFIRM qualifier, Oracle Trace identifies each facility definition and version code that matches the specified facility name and version code and asks if you want to delete it. If you use /NOCONFIRM, Oracle Trace deletes the specified facility definition from the Oracle Trace administration database without prompting for confirmation.

/VERSION="version-code"
Specifies the version of the facility for which you want to delete the facility definition. You must enclose the text string with quotation marks (" "). If there is more than one version of the facility on the system, you can use the /VERSION qualifier to specify which version you want to delete. However, you

can use an asterisk (*) as a wildcard character to delete the facility definitions for all versions of a given facility.

This is a required qualifier.

Description

Use the DELETE DEFINITION command to delete facility definitions from the Oracle Trace administration database. You do not delete the actual facility images from your system, but merely the Oracle Trace facility definition for that facility.

You cannot delete a facility definition if any active or pending collections are collecting data from that facility. Use the SHOW SELECTION /FORMAT=BRIEF command to determine if a facility has active or pending data collection associated with it. If it does, you must use the CANCEL COLLECTION command to stop data collection before you can delete the facility definition. Also, any facility selections that specify the facility definition explicitly must be deleted. If a facility selection does not specify the version of the facility and another version exists on the system, you can delete the facility definition without canceling scheduled data collection.

Examples

1.
```
$ COLLECT DELETE DEFINITION TRANS2 /NOCONFIRM /VERSION="V2.0"
%EPC-S-FACDEL_DELETED, Facility definition TRANS2 V2.0 was deleted
```

   Deletes the TRANS2 version V2.0 facility definition from the Oracle Trace administration database.

2.
```
$ COLLECT DELETE DEFINITION TRANS2 /VERSION="V2.0"
Delete TRANS2 V2.0 [N]: YES
%EPC-S-FACDEL_DELETED, Facility definition TRANS2 V2.0 was deleted
```

   Deletes the TRANS2 facility definition from the Oracle Trace administration database after prompting for confirmation.

3.
```
$ COLLECT DELETE DEFINITION  MY_APPL /NOCONFIRM /VERSION=*
%EPC-S-FACDEL_DELETED, Facility definition MY_APPL V4.2 was deleted
%EPC-S-FACDEL_DELETED, Facility definition MY_APPL V4.3 was deleted
%EPC-S-FACDEL_DELETED, Facility definition MY_APPL V4.4 was deleted
```

   Deletes the facility definitions for all versions of the facility MY_APPL by using the /VERSION qualifier.

DELETE SELECTION

Deletes one or more facility selections from the Oracle Trace administration database.

Format

DELETE SELECTION selection-name [, . . .]

Command Qualifier	Default
/[NO]CONFIRM	/CONFIRM

Parameter

selection-name
Specifies the name of one or more facility selections that you want to delete. You cannot use wildcard characters.

Qualifier

/CONFIRM (default)
/NOCONFIRM
Specifies whether Oracle Trace prompts you to confirm the deletion of each facility selection (similar to the OpenVMS DCL command DELETE /CONFIRM).

If you use the /CONFIRM qualifier, Oracle Trace identifies each selection name that matches the name you specified and asks if you want to delete it. If you use /NOCONFIRM, Oracle Trace deletes all matching facility selections without prompting for confirmation.

Description

You can delete a facility selection from the Oracle Trace administration database with the DELETE SELECTION command. You must be the creator of the selection or have OpenVMS BYPASS or SYSPRV privilege. You cannot delete a facility selection if any data collection, either active or pending, is using that facility selection.

To delete a facility selection that is being used in any current or scheduled data collections, you must first cancel the data collection. See Section 3.4 for information on how to cancel all data collection using a particular facility selection.

Examples

1.
```
$ COLLECT DELETE SELECTION ACMS_DATA /NOCONFIRM
%EPC-S-SELDEL_DELETED, Selection ACMS_DATA was deleted
```

   Deletes the ACMS_DATA facility selection from the Oracle Trace administration database.

2.
```
$ COLLECT DELETE SELECTION RDB_DATA
Delete RDB_DATA [N]: YES
%EPC-S-SELDEL_DELETED, Selection RDB_DATA was deleted
```

   Deletes the RDB_DATA facility selection from the Oracle Trace administration database after prompting you for confirmation.

EXIT

Exits Oracle Trace and returns to the DCL command level. Also used to exit from the Option> prompt and return to the Trace> prompt.

Format

EXIT

Description

Use the EXIT command to quit the Oracle Trace prompting mode after entering a series of commands. Alternately, you can press CTRL/Z at the Trace> prompt.

Examples

1.
```
Trace> EXIT
$
```

   Exits Oracle Trace and returns to DCL command level.

2.
```
Trace> CTRL/Z
$
```

   Exits Oracle Trace and returns to DCL command level.

3.
```
Trace> CREATE SELECTION INFO_DATA /OPTIONS
Option> FACILITY ACMS/CLASS=ALL
Option> FACILITY DBMS/VERSION="V4.1"/CLASS=WORKLOAD
Option> EXIT
Trace>
```

   Exits from the Oracle Trace options environment and returns to the Trace> prompt.

EXTRACT DEFINITION

Extracts a facility definition from the Oracle Trace administration database and stores it in a binary file. You can later use the INSERT DEFINITION command to store the extracted facility definition in another Oracle Trace administration database.

Format

EXTRACT DEFINITION facility-name file-spec

Command Qualifiers	Defaults
/DECTRACE_VERSION="version-code"	/DECTRACE_VERSION="V1.2"
/VERSION="version-code"	See text

Parameters

facility-name
Specifies the name of the facility definition that you want to extract into a binary file. You cannot use wildcard characters.

file-spec
Specifies the name of the file that Oracle Trace creates to store the binary form of the facility definition. The default device and directory are your OpenVMS default device and directory. The default file type is EPC$DEF.

Qualifiers

/DECTRACE_VERSION="version-code"
Specifies an older version of Oracle Trace you want the definition to be extracted for. Valid versions are "V1.0" for Versions 1.0-0 and 1.0A-0, "V1.1" for Versions 1.1-0 and 1.1A-0, and "V1.2" for all versions after 1.2-0. If you do not specify an Oracle Trace version, Oracle Trace extracts facility definition compatible with Version 1.2 and higher.

If you attempt to extract a definition which uses functionality not supported by the specified version, an error is displayed and the definition is not extracted.

- Any facility definition that uses the [NO]CRLF_INTERPRET item option cannot be extracted to a Version 1.0 format.
- Any facility definition that uses the RELATE option, or any cross-facility items, cannot be extracted to a Version 1.0 or Version 1.1 format.

/VERSION="version-code"
Specifies the version of the facility. The name of the version can be any printable text string up to 10 characters. You cannot use wildcard characters, and the string must be enclosed with quotation marks (" ").

This is a required qualifier.

Description

Use the EXTRACT DEFINITION command to copy facility definitions from one system to another. The binary file contains the entire facility definition, including the original creation date. This allows an exact replica of the facility definition to be moved to a remote site and included in the remote site's Oracle Trace administration database by using the INSERT DEFINITION command.

You need OpenVMS BYPASS, READALL, or SYSPRV privilege to extract either a registered facility definition or the definition for a facility created by another user.

Examples

1. ```
   $ COLLECT EXTRACT DEFINITION MY_FACILITY MY_FAC031 /VERSION="V3.1"
   %EPC-S-FACEXT, Facility definition(s) was successfully extracted
   ```

   Creates a file containing the facility definition for V3.1 of MY_FACILITY. Oracle Trace uses your default device and directory and the default file type.

2. ```
   $ COLLECT EXTRACT DEFINITION SAMPLE_PROG SAMPLE020 -
   _$ /VERSION="V2.0" /DECTRACE_VERSION="V1.0"
   %EPC-S-FACEXT, Facility definition(s) was successfully extracted
   ```

   Extracts the facility definition for SAMPLE_PROG T2.0 to a binary file. This file can be inserted into target systems running Oracle Trace Versions 1.0 or 1.0A.

HELP

Displays information about Oracle Trace and Oracle Trace commands.

Format

HELP [topic] . . .

Parameter

topic
Specifies the subject about which you want information from the HELP library. You can specify the names of one main topic and up to eight subtopics with the HELP command.

Example

```
Trace> HELP CREATE
  CREATE

    Creates a facility definition or a facility selection.

  Additional information available:

  DEFINITION     SELECTION

CREATE Subtopic?
```

Invokes Oracle Trace HELP to display information about the CREATE DEFINITION and CREATE SELECTION commands and prompts for further information.

INSERT DEFINITION

Inserts a facility definition in a binary format (previously created by the EXTRACT DEFINITION command) into the Oracle Trace administration database.

Format

INSERT DEFINITION file-spec

Command Qualifiers	Defaults
/[NO]LIBRARY	/NOLIBRARY
/[NO]REPLACE	/NOREPLACE

Parameter

file-spec
Specifies the name of the binary file that contains the facility definition. The default device and directory are your OpenVMS default device and directory. The default file type is EPC$DEF.

Qualifiers

/LIBRARY
/NOLIBRARY (default)
Inserts the facility definition into the Oracle Trace facility library (EPC$FACILITY.TLB) located in SYS$COMMON:[SYSLIB]. This text library is referenced during reinstallations of Oracle Trace. If you are going to remove Oracle Trace from your system but wish to retain all of your facility definitions, you need to extract the definitions from the Oracle Trace administration database (with the EXTRACT DEFINITION command) and re-insert them into the Oracle Trace facility library.

/REPLACE
/NOREPLACE (default)
Specifies that the facility definition replaces any previously existing facility definition with the same facility name and version code. If a facility definition exists and you attempt to insert a matching facility and version without using the /REPLACE qualifier, Oracle Trace issues a warning and does not store the new definition.

If no facility definition with the same name and version code exists, the /REPLACE qualifier is ignored.

Description

The EXTRACT DEFINITION and INSERT DEFINITION commands allow you to replicate facility definitions on multiple systems. This is useful when you want to test an application in several different environments.

You can insert an Oracle Trace Version 1.0 definition into a Version 1.1 administration database, but you cannot insert a Version 1.1 definition into a Version 1.0 database. See the EXTRACT DEFINITION command for more information.

You need OpenVMS SYSPRV or BYPASS privilege to replace either a registered facility definition or the definition for a facility created by another user.

Examples

1. `$ COLLECT INSERT DEFINITION MY_FAC031 /REPLACE`

   Inserts the facility definition in the binary file MY_FAC031.EPC$DEF into the Oracle Trace administration database.

2. `$ COLLECT INSERT DEFINITION NEW_TOOL /LIBRARY`

   Inserts the facility definition in the binary file NEW_TOOL.EPC$DEF into the Oracle Trace administration database and the Oracle Trace facility library.

SCHEDULE COLLECTION

Schedules data collection to occur on the cluster or local node.

Format

SCHEDULE COLLECTION collection-name data-collection-file[, . . .]

Command Qualifiers	Defaults
/BEGINNING=time	/BEGINNING=current-time
/[NO]CLUSTER	/CLUSTER
/COLLECTION_FILES=(param=value[, . . .])	See text
/DURATION=time	See text
/ENDING=time	See text
/[NO]FILELIST	/NOFILELIST
/FLUSH_INTERVAL	See text
/PROTECTION=(ownership:access[, . . .])	See text
/REGISTRATION_ID=(registration-id[, . . .])	/REGISTRATION_ID=*
/SELECTION=selection-name	See text

Parameters

collection-name

The name of the collection you are scheduling. The collection name must be a unique 1- to 32-character string consisting of alphanumeric characters, dollar signs ($), and underscores (_). The collection name must be unique for the local system or for the entire cluster in a cluster environment.

data-collection-file

The name of the file or files to create for storing the collected data.

The file specification can consist of a device, directory, file name, and file type. The device and directory default to your current OpenVMS default device and directory.

Oracle Trace creates the data collection files when the collection is scheduled, not when data collection actually begins. The default protection on the data collection file is your default protection. You can set the protection on the file using the /PROTECTION qualifier. For example, by removing world read and write access and specifying group read and write access, you can restrict data collection to only those processes that are in your UIC group.

You can specify more than one data file to spread the output over multiple disks and avoid I/O bottlenecks during collection. Note that a process only connects to a single data collection file during a collection and will not fail-over to a new file.

See the FORMAT command for instructions on combining multiple data files after a collection has ended.

Qualifiers

/BEGINNING=time
Specifies when data collection starts. You can specify the starting time using either an OpenVMS absolute or delta time format, including both date and time.

By default, data collection starts immediately. If you specify a relative date and time, the starting time is determined by the relative offset from the current time. For example, if you ask for "+1:" at 10:00, the test begins at 11:00.

The starting time for data collection must be in the future. If you specify a starting time that has already passed, Oracle Trace issues an error message and does not schedule data collection.

/CLUSTER (default)
/NOCLUSTER
Specifies whether data collection occurs only on the local node or on all nodes of the cluster.

By default, SCHEDULE COLLECTION schedules data collection cluster-wide (that is, data collection occurs on every node in the cluster). You use the /NOCLUSTER qualifier to schedule data collection on only the current node.

On a standalone system, the /CLUSTER qualifier is ignored.

/COLLECTION_FILES=(param=value[, . . .])
Specifies the characteristics of the data collection files. Table 6–8 shows the parameters and their defaults.

Table 6–8 Collection File Characteristics

Parameter	Default Action
BUFFER_SIZE=n blocks	64 blocks
INIT_ALLOCATION=n blocks	Size determined by Oracle Trace
MAX_ALLOCATION=n blocks	All available disk space
PROTECTION=(ownership:access[, . . .])	File protection for directory or process

Oracle Corporation recommends that you always use the PROTECTION parameter to specify a world read and write protection for the collection file. This ensures that the Oracle Trace collection process always has privileges to write to the collection file.

The BUFFER_SIZE parameter specifies the size in blocks of the buffer used by the Oracle Trace service routines to log data to the data collection files. The buffer size determines the number of events logged per buffer flush. For example, to perform a flush for every 10, events you must multiply the *max_record_size* by 10 and then perform the calculation for buffer size.

For best performance, the collection buffer specified should be large enough to handle the maximum record size for the facilities in the selection. Oracle Trace requires 4 bytes of overhead per page, and the maximum record size must also take into account the Oracle Trace resource utilization and cross-facility items and 25 bytes for Oracle Trace control information. The minimum buffer size is calculated as:

buffer_size = max_record_size in bytes / (512—4) + 1

Changing the buffer size allows you to tradeoff the memory a process uses against the physical I/O it incurs. A low setting will use less memory, but more physical I/O. A high setting will use more memory, but less physical I/O. By default, the buffer size is set to the maximum of 64 blocks, the optimal value for using less I/O.

Oracle Trace stores all of the data that it collects in private buffers. Under normal circumstances, Oracle Trace writes the contents of these buffers out to a data collection file when either of the following occurs:

- Whenever a buffer becomes full

- Whenever a flush interval occurs, if you specified a flush interval in the SCHEDULE COLLECTION command. The FLUSH_INTERVAL qualifier specifies the interval in seconds for Oracle Trace to write out all process buffers to the data collection files.

When a process that is collecting data terminates, Oracle Trace writes the data that has accumulated in its buffers to the data collection file when:

- The application performs a normal exit.
- The application terminates abnormally (for example, producing an Oracle Rdb bugcheck).
- The application is stopped using Ctrl/C or Ctrl/Y and STOP.
- The collection is cancelled (using the CANCEL COLLECTION command) or the end time of the collection is reached.
- The $FORCEX system service is used to delete the process.

Note that data in the collection buffer is lost if STOP/ID is used to stop the process. STOP/ID calls the $DELPRC service which does not call the EPC$SHR.EXE shareable image exit handler.

Note

Oracle Corporation recommends that to avoid losing collection data, you never stop collections using the STOP/ID command. Instead, use the other methods described in this section.

The INIT_ALLOCATION and MAX_ALLOCATION parameters specify the minimum and maximum size (in 512 byte blocks) of the data collection files. Once the maximum size is reached, the file is closed and any processes that had been recording data to that file stop collecting. When all of the data collection files are full and no more processes can collect data, the collection is cancelled and a message is written to the Oracle Trace history database. If no maximum is given, Oracle Trace attempts to write to the files until either the collection is cancelled or the device or user's quota is full. Note that a process only connects to a single data collection file during a collection and will not fail-over to a new file.

The PROTECTION parameter specifies the protection on the data collection files. A process must have read and write access to a file in order to record event data in the file. Valid ownership categories are (S)ystem, (O)wner, (G)roup, and (W)orld. A valid access code is any combination of the following: (R)ead, (W)rite, (E)xecute, and (D)elete.

If you do not specify a value for each ownership category, or if you omit the /PROTECTION qualifier, Oracle Trace applies the current default protection for each unspecified category. If the data collection file replaces a previous version, the protection scheme of the old file is used on the new one.

Note that if a FILELIST or multiple data collection files are given, all files have the same characteristics.

/DURATION=time
Specifies the length of time over which data collection occurs. You must specify the duration as an OpenVMS delta time.

The /DURATION and /ENDING qualifiers are mutually exclusive.

/ENDING=time
Specifies when data collection ends. You can specify the ending time using either a relative or absolute OpenVMS time format, including both date and time.

You must specify an ending time or use the /DURATION qualifier. If you do not include the /ENDING or /DURATION qualifiers, Oracle Trace issues an error message and does not schedule data collection.

If you specify a relative date and time, the ending time is determined by the relative offset from the current time, *not* the starting time.

If the ending time is earlier than the starting time, Oracle Trace issues an error message and does not schedule data collection.

/FILELIST
/NOFILELIST (default)
Specifies that the second parameter to the SCHEDULE COLLECTION command is a **file list**, which is a file containing a list of file specifications to use as output data files for data collection. One advantage of using a file list is that you can specify multiple data files without worrying about the DCL limitations on the length of a command line.

The /FILELIST qualifier is position-dependent; you must specify it on the second parameter to the command. For example:

```
$ COLLECT SCHEDULE COLLECTION MY_TEST DATA_FILE.TXT /FILELIST
```

Each file specification must be on a separate line within the file list. For example:

```
DISK$USER1:[SMITH.COLLECTOR]DATA1.DAT
DISK$USER2:[DATA]DATA2.DAT
DISK$USER3:[DATA]DATA3.DAT
```

The default file type for the file list is TXT.

Note that a process only connects to a single data collection file during a collection and will not fail-over to a new file.

/FLUSH_INTERVAL=n seconds
None
Specifies the interval in seconds for Oracle Trace to write out all process buffers to the data collection files.

/PROTECTION=(ownership:access[, . . .])
This qualifier has been superseded by the PROTECTION parameter to the /COLLECTION_FILES qualifier. It is maintained only for upwards compatibility. Note that if you specify the protection using both methods, the values must be identical or the command will fail.

/REGISTRATION_ID=(registration-id[, . . .])
Limits collections to processes with information corresponding to the information you specify with this qualifier. By default, Oracle Trace collects for all processes and all facilities, or if there is a facility selection, it collects only for the facilities specified in a facility selection.

You can indicate the processes for which you want Oracle Trace to collect data by providing any of the following information about that process in the /REGISTRATION_ID qualifier:

- EPID
- Full image name including the device, directory specification, and version number
- User name
- Process name
- Information associated with the process by the registration-id argument passed by the facility when it first registers with Oracle Trace. See Chapter 7 for more information.

You can specify this information in any order. However, when you use the /REGISTRATION_ID qualifier, Oracle Trace only collects for processes when there is a match between the information you provide in the /REGISTRATION_ID qualifier and the information the EPC$INIT service routine provides for those processes. Therefore, you must be sure that the information you provide in the /REGISTRATION_ID corresponds to the information associated with each process you want to collect data for. See Section 3.1 for more information about this qualifier.

SCHEDULE COLLECTION

/SELECTION=selection-name
Specifies the facility selection to use for the collection. The facility selection lists the facilities for which data will be collected and specifies a collection class for each.

This is a required qualifier.

Description

You can schedule data collection cluster-wide or on a subset of the nodes in a VMScluster. To schedule data collection on a subset of a cluster, you must log into each node on which you want to schedule the collection and start local data collection on each of those nodes.

Multiple collection can be active on a node at any time. You can schedule many different collections on your local system or cluster, and they can be either overlapping or concurrent.

The situations that result in an error are:

- If the ending time is earlier than the starting time
- If the starting date and time are in the past
- If you do not have write access to the Oracle Trace administration or history databases

See Section 3.2 for more information on scheduling data collection.

Examples

1.
```
$ COLLECT SCHEDULE COLLECTION MY_TEST MY_DATA.DAT -
_$ /SELECTION=ACMS_DATA -
_$ /BEGIN=11:00 /END=12:00 -
_$ /COLLECTION_FILES=(PROTECTION=(W:RW))-
_$ /NOCLUSTER
%EPC-S-SCHED, Data collection MY_TEST is scheduled
```

   Schedules the collection MY_TEST to run on the local node and to begin at 11:00 and end at 12:00 on the current day. Oracle Trace stores the collected data in the file MY_DATA.DAT in your default device and directory and uses your default protection scheme.

2.
```
$ COLLECT SCHEDULE COLLECTION WEDNESDAYS_TEST WEDNESDAY_DATA.DAT -
_$ /SELECTION=RDB_AND_ACMS -
_$ /BEGINNING="+1-" /DURATION="2:" -
_$ /COLLECTION_FILES=(INIT_ALLOCATION=5000,MAX_ALLOCATION=5000, -
_$ BUFFER_SIZE=32, PROTECTION=(W:RW)) /CLUSTER
%EPC-S-SCHED, Data collection WEDNESDAYS_TEST is scheduled
```

Schedules a cluster-wide collection to begin at the current time on the following day. The name of the collection is WEDNESDAYS_TEST. It collects the data described by the RDB_AND_ACMS facility selection. Data is collected for two hours and is stored in the file WEDNESDAY_ DATA.DAT. A large initial allocation is specified to avoid the overhead of extents and to ensure that adequate disk space is reserved for the collection. Note that the data collection files are created immediately when the collection is scheduled, not when the collection becomes active.

3.
```
$ COLLECT SCHEDULE COLLECTION ALL_DAY LOTSOFDATA.DAT -
_$ /SELECTION=JUST_RDB -
_$ /DURATION="8:" -
_$ /CLUSTER -
_$ /COLLECTION_FILES=(INIT_ALLOCATION=10000, MAX_ALLOCATION=50000, -
_$ PROTECTION=(W:RW))
_$ BUFFER_SIZE=64, PROTECTION=(G:RW)) -
_$ /REGISTRATION_ID=(DISK1:[PAYROLL.*]*_CHECKS.EXE;*)
```

Schedule data collection to begin immediately and run for eight hours. The collection runs on the entire cluster and collects Oracle Rdb event data from all check writing programs in the PAYROLL account: WEEKLY_CHECKS.EXE, MONTHLY_CHECKS.EXE, and ON_DEMAND_ CHECKS.EXE). A maximum file size is specified so that the disk is not filled by the collection.

SET ACCESS

Allows non-privileged users access to the Oracle Trace administration and history databases, or to a specified formatted database.

Format

SET ACCESS user-name Trace-database

Command Qualifier	Default
/TYPE=access-type	/TYPE=WRITE

Parameters

user-name
The name of the user for whom you want to grant or deny access to the database. You can use the wildcard PUBLIC to grant access to all users.

Trace-db
The full file specification of the Oracle Rdb root file of the Oracle Trace database that you wish to grant or deny access to. The directory specification defaults to the current directory and the file extension defaults to RDB. Note that you can use symbolic names.

Qualifier

/TYPE=access-type
Specifies the type of access to the database the user may have. Valid types are:

- READ
- WRITE (default)
- NONE

Users with write access may add or delete other users from the access list. Note that you cannot downgrade your own privileges from WRITE to READ or NONE. Also, note that you must have write access to a formatted database to be able to generate an Oracle Trace frequency report from it.

Description

With Oracle Rdb Version 4.0 and higher, only creators of a database or users with BYPASS or SYSPRV privileges may read or modify a database. The SET ACCESS command allows you to grant non-privileged users access to Oracle Trace databases.

SET ACCESS displays an error if:

- The access-type is illegal
- The person entering the command does not have the WRITE access to the Oracle Trace administration database
- The specified database does not exist
- The database is not an Oracle Trace Version 2.2 database

Note that world write access to the Oracle Trace administration and history databases can be set as part of the Oracle Trace installation procedure. Users must have write access to the administration database in order to schedule collections. Users must also have write access to the history database in order to log error messages during data collection.

Examples

1. `$ COLLECT SET ACCESS PUBLIC EPC$ADMIN_DB /TYPE=WRITE`

   Grants world write access to the Oracle Trace administration database.

2. `$ COLLECT SET ACCESS SMITH WORK1:[JONES]TODAY.RDB /TYPE=READ`

   Grants read access to user SMITH. This allows SMITH to generate Oracle Trace detail or summary reports based on the data in the TODAY.RDB formatted database in another user's (JONES) account.

SET HISTORY

Temporarily changes the history database that Oracle Trace uses for the SHOW HISTORY command, or creates a new history database.

Format

SET HISTORY [history-file]

Command Qualifier	Default
/[NO]NEW_FILE	/NONEW_FILE

Parameter

history-file
Specifies the history database to use when displaying history information with the SHOW HISTORY command. The SET HISTORY command remains in effect for the life of the user's process or until another SET HISTORY command is issued. The default file type is RDB.

If you do not specify the /NEW_FILE qualifier, then the history-file is a required parameter.

Qualifier

/NEW_FILE
/NONEW_FILE (default)
Specifies that a new history database will be created. The /NEW_FILE qualifier does not accept a file specification. Oracle Trace creates the new history file with the logical name EPC$HISTORY_DB and the next higher version number.

Use of the /NEW_FILE qualifier requires OpenVMS OPER privilege.

Description

The SET HISTORY command performs two separate functions:

- Setting an alternate history file for SHOW HISTORY commands
- Creating a new history file for recording Oracle Trace messages

If you specify the history-file parameter, Oracle Trace points all SHOW HISTORY commands to the specified file. This command is useful for examining the information in an old history file. Note that the scope of this command is limited to within the Trace> prompt context for this process. If you exit from the Oracle Trace command environment and then re-enter it, all subsequent SHOW HISTORY commands will reference the current, active history file.

If you specify the /NEW_FILE qualifier, Oracle Trace creates a new history file and writes any new messages to the new file. A slight lapse between the time you issue the SET HISTORY command and the time Oracle Trace starts writing to the new file should not exceed 30 seconds. In the meantime, history records continue to be written to the old file.

If you specify both a history file name and the /NEW_FILE qualifier, Oracle Trace issues an error message and does not perform either function.

Examples

1.
```
$ COLLECT
Trace> SET HISTORY OLD_HISTORY
Trace> SHOW HISTORY
```

   Temporarily changes the history file for SHOW HISTORY commands to the file OLD_HISTORY.RDB so that you can display information from an old history database.

2.
```
$ COLLECT SET HISTORY /NEW_FILE
```

   Creates a new history file with the same file name as referred to by the logical name EPC$HISTORY_DB and a version number one higher than the current file.

SHOW ACCESS

Displays a report of all users who have access to an Oracle Trace database. The database may be the Oracle Trace history or administration databases, or any formatted database.

Format

SHOW ACCESS db-name

Command Qualifier	Defaults
/OUTPUT=file-spec	/OUTPUT=SYS$OUTPUT

Parameter

database-name
Name of the Oracle Trace database to display the access list for. The directory specification defaults to the current directory and the file extension defaults to RDB. Note that you can use symbolic names.

Qualifier

/OUTPUT=file-spec
Specifies the destination for the display. By default, Oracle Trace displays the information on the current SYS$OUTPUT device (usually, your terminal). The /OUTPUT qualifier can redirect the output to a file or another device.

Description

See the description of the SET ACCESS command.

Example

```
$ COLLECT SHOW ACCESS EPC$ADMIN_DB

24-JUN-1992 15:41      Database Access Information                    Page 1
Database: EPC$DATABASE_DIR:EPC$ADMIN_DB.RDB             Oracle Trace V2.2-0
    Name                  Access  Creator/User
    --------------------  ------  ------------
    JONES                 READ    U
    PUBLIC                READ    U
    SMITH                 WRITE   U
    SYSTEM                WRITE   C
```

Displays the access to the Oracle Trace administration database.

SHOW COLLECTION

Displays information about active or pending data collection on your system.

Format

SHOW COLLECTION [collection-name]

Command Qualifiers	Defaults
/[NO]CLUSTER	/CLUSTER
/FORMAT=type	/FORMAT=BRIEF
/OUTPUT=file-spec	/OUTPUT=SYS$OUTPUT
/SELECTION=selection-name	/SELECTION=*

Parameter

collection-name
Specifies the name of the data collection for which to display information. You can omit the collection name or use an asterisk (*) as a wildcard character to display information about all collections.

If you specify both a facility selection (with the /SELECTION qualifier) and a collection, Oracle Trace uses the collection name first. If the data collection does not use the specified selection, Oracle Trace executes the command using the collection name but also issues a warning message.

Qualifiers

/CLUSTER (default)
/NOCLUSTER
Specifies whether to display information about data collection scheduled only on the local node, or about data collection scheduled on any node in the cluster.

By default, SHOW COLLECTION displays information about data collection scheduled on any node in the cluster.

On a standalone system, the /CLUSTER qualifier is ignored.

/FORMAT=type
Specifies the amount of information to display. Valid types are:

BRIEF
FULL

The default format is BRIEF, which displays the collection name, facility selection name, and starting and stopping times for data collection. If you specify /FORMAT=FULL, Oracle Trace displays a complete description of the data collection.

/OUTPUT=file-spec
Specifies the destination for the display. By default, Oracle Trace displays the information on the current SYS$OUTPUT device (usually, your terminal). The /OUTPUT qualifier can redirect the output to a file or another device.

/SELECTION=selection-name
Specifies the name of a facility selection for which you want to see schedule information. If you use an asterisk (*) as a wildcard character, or if you omit both the collection name and the /SELECTION qualifier, Oracle Trace displays information about all data collection currently running or scheduled for the system or VMScluster.

Description

See Section 3.5.1 for a full description of the information in the SHOW COLLECTION displays.

Examples

1. `$ COLLECT SHOW COLLECTION /SELECTION=ACMS_DATA /NOCLUSTER`

   Displays information about any data collection running or scheduled to run on the local node that uses the facility selection ACMS_DATA.

2. `$ COLLECT SHOW COLLECTION WEDNESDAYS_TEST`

   Displays information about the data collection WEDNESDAYS_TEST.

SHOW DEFINITION

Displays information about one or more facility definitions registered in the Oracle Trace administration database.

Format

SHOW DEFINITION [facility-name]

Command Qualifiers	Defaults
/FORMAT=type	/FORMAT=NAMES_ONLY
/OUTPUT=file-spec	/OUTPUT=SYS$OUTPUT
/VERSION="version-code"	/VERSION=*

Parameter

facility-name
Specifies the name of the facility definition you want information about. You can use an asterisk (*) as a wildcard character to display information about all of the facilities defined on the system.

If you do not specify a facility name, Oracle Trace displays information on all the currently defined facilities.

Qualifiers

/FORMAT=type
Specifies the amount of information to display. Valid types are:

FULL
NAMES_ONLY

The default format type is NAMES_ONLY.

/OUTPUT=file-spec
Specifies the destination for the display. By default, Oracle Trace displays the information on the current SYS$OUTPUT device (usually, your terminal). The /OUTPUT qualifier can redirect the output to a file or another device.

/VERSION="version-code"
Specifies the version of the facility that you want information about. You must enclose the text string with quotation marks (" ").

If you do not specify a version code or if you use an asterisk (*) as a wildcard character, Oracle Trace displays information about all versions of the facility that you name.

Description

See Section 5.5 for a description of each type of SHOW DEFINITION display.

Examples

1.
```
$ COLLECT SHOW DEFINITION /FORMAT=NAMES_ONLY

23-DEC-1995 11:42       Facility Definition Information       Page 1
Names Only Report                                  Oracle Trace V2.2-0

 Facility:          Version:    Creation Date:      Class:
 -----------------  ----------  ------------------  ------------
 ATM_SAMPLE         V1.0        25-AUG-1995 14:17   ALL         (D)
 MY_FACILITY        V4.2        23-OCT-1995 17:09   ALL         (D)
```

   Displays the names, versions, creation dates, and collection classes for all of the facility definitions on the system.

2.
```
$ COLLECT SHOW DEFINITION MY_FACILITY /VERSION="V4.2" /FORMAT=FULL

23-Dec-1995 11:43      Facility Definition Information        Page 1
Full Report                                        Oracle Trace V2.2-0

Facility:        MY_FACILITY
Number:          2048
Version:         V4.2
Creation date:   13-Oct-1995 17:09
Created by:      SMITH

Events:
   .
   .
   .
Items:
   .
   .
   .
```

   Displays complete information about the MY_FACILITY V4.2 facility definition.

SHOW HISTORY

Displays a record of the informational and error messages that are encountered during data collection.

Format

SHOW HISTORY [collection-name]

Command Qualifiers	Defaults
/BEFORE=time	All data in file
/[NO]CHRONOLOGICAL	/CHRONOLOGICAL
/[NO]CLUSTER	/CLUSTER
/FORMAT=type	/FORMAT=ERROR
/NODE="node-name"	/NODE=*
/OUTPUT=file-spec	/OUTPUT=SYS$OUTPUT
/SINCE=time	All data in file

Parameter

collection-name
The name of the collection for which to display information. By default, Oracle Trace displays the history for all collections. You can display information for a single collection by specifying a collection name on the command line. You cannot use wildcard characters.

Qualifiers

/BEFORE=time
Specifies that only information before a specific date and time be displayed. By default, Oracle Trace displays all information in the history file.

/CHRONOLOGICAL (default)
/NOCHRONOLOGICAL
Specifies for history messages to be displayed in chronological order with headers displayed for each message. This qualifier is useful for debugging purposes in a cluster environment.

/CLUSTER (default)
/NOCLUSTER
Specifies whether Oracle Trace displays information on the current node only or on all nodes in the VMScluster.

By default, SHOW HISTORY displays cluster-wide information.

On a standalone system, the /CLUSTER qualifier is ignored.

/FORMAT=type
Specifies the type of messages that should be displayed. By default, SHOW HISTORY displays only the error messages. The valid types are:

ALL
INFORMATIONAL
ERROR (default)

/NODE="node-name"
Selects only those messages from the specified node. By default, Oracle Trace displays history messages from all nodes in the VMScluster.

/OUTPUT=file-spec
Specifies the destination for the display. By default, Oracle Trace displays the information on the current SYS$OUTPUT device (usually, your terminal). The /OUTPUT qualifier can redirect the output to a file or another device.

/SINCE=time
Specifies that only information after a specific date and time be displayed. By default, Oracle Trace displays all of the information in the history file.

Description

Displays information about the Oracle Trace system, including:

- When the system started and stopped
- When collections started, stopped, or were cancelled
- When processes registered and unregistered
- Any errors that occurred in the Oracle Trace registration and collecting processes

The SHOW HISTORY command can be used in conjunction with the SET HISTORY command to display information from older history databases.

See Section 3.5.2 for a description of the information in the SHOW HISTORY displays.

Examples

1. `$ COLLECT SHOW HISTORY`

   Displays the errors which have occurred for the Oracle Trace system on the local VMScluster (in a cluster environment).

2. `$ COLLECT SHOW HISTORY /FORMAT=INFORMATIONAL /OUTPUT=HISTORY.LOG`

   Writes a description of all informational messages and events to the file HISTORY.LOG.

3. ```
   $ COLLECT
   Trace> SET HISTORY OLD_HISTORY
   Trace> SHOW HISTORY
   ```

   Temporarily changes the history file for SHOW HISTORY commands to the file OLD_HISTORY.RDB so that you can display information from an old history database.

4. `$ COLLECT SHOW HISTORY /NODE=MYVAX2 /FORMAT=ALL`

   Displays all of the errors and messages for node MYVAX2.
   ```

SHOW REGISTER

Displays information about processes currently registered with the Oracle Trace administration database, the facilities and registration IDs for the processes, and whether the processes are currently collecting data.

Format

SHOW REGISTER

Command Qualifiers	Defaults
/[NO]CLUSTER	/CLUSTER
/OUTPUT=file-spec	/OUTPUT=SYS$OUTPUT

Qualifiers

/CLUSTER
/NOCLUSTER (default)
Specifies whether Oracle Trace displays information on the processes on the local node only or on all nodes in the VMScluster.

On a standalone system, the /CLUSTER qualifier is ignored.

/OUTPUT=file-spec
Specifies the destination for the display. By default, Oracle Trace displays the information on the current SYS$OUTPUT device (usually, your terminal). The /OUTPUT qualifier can redirect the output to a file or another device.

Description

This command is useful for determining valid registration IDs that can be used to schedule per-user or per-image collections. The register display is also useful for solving Oracle Trace collection problems.

See Section 3.1 for a description of the information in the SHOW REGISTER display.

Example

```
$ COLLECT SHOW REGISTER
```

Displays the names and process IDs of processes registered with the Oracle Trace administration database.

SHOW SELECTION

Displays information about one or more facility selections stored in the Oracle Trace administration database on the current system or VMScluster.

Format

SHOW SELECTION [selection-name]

Command Qualifiers	Defaults
/FORMAT=type	/FORMAT=BRIEF
/OUTPUT=file-spec	/OUTPUT=SYS$OUTPUT

Parameter

selection-name
The name of the facility selection for which you want information. You can use an asterisk (*) as a wildcard character to display information about all facility selections defined on the system.

If you do not specify a selection name, Oracle Trace displays information on all of the facility selections defined on the system.

Qualifiers

/FORMAT=type
Specifies the amount of information to display. Valid types are:

BRIEF
FULL
NAMES_ONLY

The default format is BRIEF.

/OUTPUT=file-spec
Specifies the destination for the display. By default, Oracle Trace displays the information on the current SYS$OUTPUT device (usually, your terminal). The /OUTPUT qualifier can redirect the output to a file or another device.

Description

See Section 2.5 for a full description of the information in the SHOW SELECTION displays.

Examples

1. `$ COLLECT SHOW SELECTION /FORMAT=NAMES_ONLY`

   Displays the names of all facility selections on the system.

2. `$ COLLECT SHOW SELECTION ACMS_DATA /FORMAT=FULL -`
   `_$ /OUTPUT=ACMS_DATA_FULL.LIST`

   Writes a complete description of the ACMS_DATA facility selection to the file ACMS_DATA_FULL.LIST.

SHOW VERSION

Displays the version number of Oracle Trace running on your system.

Format

SHOW VERSION

Example

```
Trace> SHOW VERSION
Oracle Trace Version V2.2
Trace>
```

Displays the version number of Oracle Trace installed on the system—in this case, Version 2.2.

SPAWN

Creates a subprocess of the current process. Allows you to temporarily exit from the Oracle Trace command environment without detaching from the Oracle Trace administration database.

Format

SPAWN [command]

Command Qualifiers	Defaults
/[NO]CARRIAGE_CONTROL	See text
/[NO]CLI	See text
/INPUT=file-spec	See text
/[NO]KEYPAD	/KEYPAD
/[NO]LOGICAL_NAMES	/LOGICAL_NAMES
/[NO]NOTIFY	/NONOTIFY
/OUTPUT=file-spec	/OUTPUT=SYS$OUTPUT
/PROCESS=subprocess-name	See text
/[NO]PROMPT[=string]	See text
/[NO]SYMBOLS	/SYMBOLS
/[NO]WAIT	/WAIT

Parameter

command
Specifies an optional command to be executed by the subprocess you are creating. If you specify the command parameter, you create a subprocess which executes the command and returns control to the Oracle Trace session when the command terminates. If you include the /INPUT qualifier with the command parameter, the subprocess reads the commands from the specified input file after the command executes. The command string cannot exceed 132 characters.

If you omit the command parameter, the SPAWN command creates a subprocess and attaches your terminal to it. You can return to your Oracle Trace session by logging out of the subprocess or by issuing the ATTACH /PARENT command. If you have created several subprocesses, you can switch between them using the DCL command ATTACH/IDENTIFICATION.

Qualifiers

/CARRIAGE_CONTROL
/NOCARRIAGE_CONTROL
Determines whether carriage control or line feed characters (or both) are prefixed to the DCL-prompt string of the subprocess. The default is the current setting of the parent process.

/CLI[=cli]
/NOCLI
Specifies an alternate command language interpreter (CLI) for the subprocess to use. The default is the CLI the parent process uses.

The CLI you specify must be located in SYS$SYSTEM and have the file type EXE.

/INPUT=file-spec
Specifies an input file containing one or more commands for the spawned subprocess to execute. If you specify a command with an input file, the command is processed before the commands in the input file. The subprocess terminates when processing is complete. You cannot use wildcards in the file specification.

/KEYPAD (default)
/NOKEYPAD
Determines whether DCL keypad symbols and the current DCL keypad state are copied from the parent process to the subprocess. The default is to copy any key definitions or states (or both) you have established with the DEFINE /KEY command. Use the /NOKEYPAD qualifier if you do not want the key settings to be copied.

/LOGICAL_NAMES (default)
/NOLOGICAL_NAMES
Determines whether the system passes process logical names and logical name tables to the subprocess. The default is to copy all process logical names and logical name tables except those marked CONFINE or those created in executive or kernel mode.

/NOTIFY
/NONOTIFY (default)
Determines whether a message is sent to your terminal to notify you that your subprocess has been completed or aborted. Do not specify /NOTIFY unless you also specify the /NOWAIT qualifier.

/OUTPUT=file-spec
Specifies the output file to which the output of the SPAWN operation is to be written. When you specify /NOWAIT, you should use /OUTPUT to specify an output other than SYS$OUTPUT to prevent your terminal from being used by both processes simultaneously. By default, Oracle Trace directs the output to the current SYS$OUTPUT device (usually, your terminal).

/PROCESS=subprocess-name
Specifies the name of the subprocess to be created. The default name for the subprocess is USERNAME_#. The pound sign (#) denotes a unique number.

/PROMPT[=string]
Specifies the prompt string for the subprocess. If you specify /PROMPT but do not specify a string, the default prompt is displayed. The default is to copy the current prompt string from the parent process.

/SYMBOLS (default)
/NOSYMBOLS
Determines whether the system passes DCL global and local symbols to the subprocess.

/WAIT (default)
/NOWAIT
Controls whether the system waits until the subprocess is completed before allowing more commands to be issued by the parent process. The /NOWAIT qualifier allows you to enter more commands while the specified subprocess is running. When you specify /NOWAIT, you should also specify /OUTPUT to direct output to a file (rather than to your screen). This prevents your terminal from being used by both processes simultaneously.

Description

The SPAWN command creates a subprocess of your current Oracle Trace session—your parent process. The context of your Oracle Trace process is copied to the subprocess.

You can use the SPAWN command to leave the Oracle Trace command environment temporarily, to perform some non-Oracle Trace activities, and then return to your original Oracle Trace session. The advantage to using the SPAWN command is that you do not have to detach and reattach to the Oracle Trace history or administration databases each time you need to leave the Oracle Trace command environment. Spawning also allows you to maintain your command recall buffer within the Oracle Trace environment.

Any qualifiers to the SPAWN command must be specified *before* the command parameter. If you want to use qualifiers on the command parameter, enclose the entire command in quotation marks (""). For example:

```
Trace> SPAWN/OUTPUT=MY_DIR.TXT "DIR/SIZE/DATE"
%EPC-S-SPAWN, Spawn successfully completed
```

Example

```
$ COLLECT
Trace> SPAWN
$ SHOW DEVICE DUA2

Device                 Device          Error    Volume        Free  Trans Mnt
 Name                  Status          Count     Label       Blocks Count Cnt
MYVAX$DUA2:            Mounted             0     USER2       242686     2  25

$ LOGOUT
  Process SMITH_1 logged out at 10-FEB-1995 16:27:58.50
%EPC-S-SPAWN, Spawn successfully completed
Trace>
```

Creates a subprocess of the current process so that you can execute some non-Oracle Trace activities. In this case, the user checks the free space on a disk, then returns to the parent process.

STOP SYSTEM

Stops the Oracle Trace Registrar process and interrupts any active data collection. This command is usually performed as part of your system shutdown procedure. Stopping Oracle Trace is a management function that requires OpenVMS OPER privilege to perform.

Format

STOP SYSTEM

Command Qualifier	Default
/[NO]ABORT	/NOABORT

Qualifier

/ABORT
/NOABORT (default)
Specifies whether Oracle Trace can abort active data collection on the system. If data collection is occurring on the system and you use the STOP SYSTEM command without the /ABORT qualifier, Oracle Trace issues an error message indicating that data collection must first be interrupted, and it does not stop either the Registrar process or the active collection.

To stop the system while data collection is occurring, you must use the /ABORT qualifier. In this situation, Oracle Trace issues a warning message that data collection will be interrupted and stops the system.

If no collections are currently active, Oracle Trace ignores the /[NO]ABORT qualifier.

Description

This command is *not* intended for normal interactive use. However, it is supplied for unusual cases where you want to stop the Oracle Trace system interactively. It is recommended that you stop the Oracle Trace system as part of your normal system shutdown procedure. You should add the following lines to your file SYS$MANAGER:SHUTDWN.COM:

```
$!Shutdown Oracle Trace and abort all active data collection
$ COLLECT STOP SYSTEM /ABORT
```

STOP SYSTEM

The STOP SYSTEM /ABORT command operates asynchronously. If you also stop the Oracle Rdb monitor, make sure to include a sufficient delay to allow the Register to detach from the Oracle Rdb databases or else, Oracle Rdb and Oracle Trace will hang waiting for each other. If this happens, use a command with this format to stop Oracle Trace.

```
STOP/ID=identifier of EPC$REGISTER
```

Note

You must stop the Oracle Trace **before** attempting to stop the Oracle Rdb monitor process.

To restart Oracle Trace, issue the command:

```
$ @SYS$STARTUP:EPC$STARTUP.COM
```

Examples

1.
```
$ COLLECT STOP SYSTEM
%EPC-I-STOPPED, System stopped at user's request.
$
```

   Stops Oracle Trace if data collection is not currently active on the system.

2.
```
$ COLLECT
Trace> STOP SYSTEM
%EPC-E-NOTSTOPPED, System was not stopped because data collection
is active.
%EPC-I-USEABORT, You must use the /ABORT qualifier to interrupt active
data collection.
```

   Attempts to stop Oracle Trace while data collection is occurring. Oracle Trace issues an error message indicating that data collection is active on the system and that the system cannot be stopped without the /ABORT qualifier.

3.
```
Trace> STOP SYSTEM /ABORT
%EPC-I-STOPPED, System stopped at user's request.
Trace> CTRL/Z
$
```

   Interrupts all active data collection and shuts down the Oracle Trace registrar process.

7

Oracle Trace Service Routines

Oracle Trace provides a set of service routines to use when instrumenting an application program so that event data can be collected from it. These routines are to be embedded in the source code of the program. Chapter 4 describes how to correctly instrument your code with Oracle Trace service routine calls.

Table 7–1 lists and briefly describes the Oracle Trace service routine calls.

Table 7–1 Oracle Trace Service Routines

Routine Name	Description
For Instrumenting All Applications	
EPC$INIT	Registers a facility with Oracle Trace to enable data collection on this facility.
EPC$EVENT	Records the occurrence of a point event to the data collection file.
EPC$EVENTW	Records the occurrence of a point event to the data collection file and waits for processing to complete before returning.
EPC$START_EVENT	Records the start of a duration event to the data collection file.
EPC$START_EVENTW	Records the start of a duration event to the data collection file and waits for processing to complete before returning.
EPC$END_EVENT	Records the end of a duration event to the data collection file.
EPC$END_EVENTW	Records the end of a duration event to the data collection file and waits for processing to complete before returning.

(continued on next page)

Table 7–1 (Cont.) Oracle Trace Service Routines

Routine Name	Description
For Instrumenting Cross-Facility Items	
EPC$SET_CF_ITEMS	Assigns values to all of the cross-facility items.
EPC$SET_CF_VALUE	Assigns a values to a cross-facility item.
For Instrumenting Multithreaded Applications	
EPC$SET_CONTEXT	Sets the context for a new or existing thread so resource utilization and cross-facility items can be collected.
EPC$DELETE_CONTEXT	Deletes the context associated with a particular thread.
EPC$GET_CF_ITEMS	Retrieves all cross-facility items into a buffer. This routine pertains to client/server and multithreaded applications and depends on the EPC$SET_ CONTEXT routine to change contexts for cross-facility items.

The Oracle Trace service routines are similar to the OpenVMS system services. Any routines that perform I/O have two types: synchronous (W) and asynchronous. A synchronous routine ends in W (for example, EPC$START_ EVENTW) whereas the asynchronous version has no W (for example, EPC$START_EVENT). The routines that record data collection records in the data collection file are EPC$EVENT(W), EPC$START_EVENT(W), and EPC$END_EVENT(W). All Oracle Trace routines operate in executive mode and are AST reentrant.

7.1 Error Handling

Error reporting is taken care of as follows. Each routine returns a longword condition value informing the caller of the completion status of the call. An optional quadword argument containing the completion status for the asynchronous part of the calls can be returned to the caller. In addition, all errors encountered during data collection are logged to the system-wide or cluster-wide history database.

Two types of errors are returned: Oracle Trace errors and system service errors. Oracle Trace errors are prefaced with EPC$. If an unexpected error occurs during a system routine, Oracle Trace returns the actual system code.

These system errors usually result from low process quotas or system parameters. EPC$INIT calls the following system routines:

CLREF	ENQ	GETSYI
CREMBX	EXPREG	GETTIM
DCLAST	GETDVI	QIO
DEQ	GETJPI	SETEF

The Oracle Trace start, end, and point event routines call the following system routines:

CLREF	ENQ	QIO
DCLAST	EXPREG	SETEF
DEQ	GETTIM	

Each Oracle Trace service routine has a set of return values associated with it. See the description of each service routine for a complete list of return values. Most of these values are self explanatory, but the following values are not as obvious as the rest:

- The EPC$_DISABLED return value has several different meanings depending on the context it is returned in. First, this value is returned if the facility and version do not match any active collection. Second, if this facility had been collecting data within the context of the current image and an error occurred on an earlier Oracle Trace call, EPC$_DISABLED will be returned on subsequent calls to Oracle Trace. Finally, this value will be returned if data collection has been disabled by defining the EPC$DISABLED logical. See Section 8.5 for more information on disabling data collection.
- The EPC$_EVNTNOTCOL return value indicates that the facility and version match the criteria of an active collection, but the current event was not specified in the selection criteria.
- The EPC$_FACNOTCOL return value indicates that the process has registered another facility which is actively collecting data, but the current facility does not match the selection criteria.
- The EPC$_INIT2 return value indicates that the current facility and version have already registered once within this image activation. This represents an instrumentation error in the application. EPC$INIT should only be called once by each facility within an image activation.

- The EPC$_NOREGEXISTS return value indicates that the Oracle Trace Registrar process (EPC$REGISTRAR) is not running on the local node. Use the SYS$STARTUP:EPC$STARTUP.COM procedure to restart the Registrar process. See Section 8.2 for more information on starting and stopping the Registrar process.
- The EPC$_NOTINSTALLED return value indicates that the Oracle Trace software is not installed on the system.

7.2 Program Execution Modes

Generally, Oracle Trace executes in either executive mode or executive AST mode. When an Oracle Trace operation is initiated, processing begins in executive mode. If device I/O is necessary to process the request, either the $QIO or the $QIOW system service is called. If the Oracle Trace call is synchronous, the $QIOW system service is called. If the call is asynchronous, the $QIO system service is called and an executive mode AST is specified to signal completion. At this point, Oracle Trace exits from executive mode. User processing is interrupted by Oracle Trace processing in executive AST mode when the I/O completes.

Oracle Trace should not be called from kernel mode, from executive AST mode, or from executive mode when executive-mode ASTs are disabled.

EPC$DELETE_CONTEXT

EPC$DELETE_CONTEXT deletes all associated context for the given context variable.

Format

EPC$DELETE_CONTEXT [efn] ,context-variable [,status] [,astadr] [,astparam] [,flags]

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

efn

VMS Usage: ef_number
type: longword (unsigned)
access: read only
mechanism: by value

The OpenVMS event flag to be set when EPC$DELETE_CONTEXT completes. The **efn** argument is a longword containing the address of this flag. Note that event flag 47 is reserved for Oracle Corporation.

Upon initiation, EPC$DELETE_CONTEXT clears the specified event flag (or event flag 0 if **efn** was not specified). When EPC$DELETE_CONTEXT returns, it sets the specified event flag (or event flag 0).

Overhead is significantly reduced if you do not use the **efn** argument. Use the EPC$M_NOEF flag to specify that the event flag should not be cleared or set. See the **flags** argument for information.

context-variable

VMS Usage: context
type: longword (unsigned)
access: modify
mechanism: by reference

The address of the longword that contains the context variable to delete. EPC$DELETE_CONTEXT sets the longword value to zero.

EPC$DELETE_CONTEXT

status

VMS Usage: io_status_block
type: quadword (unsigned)
access: write only
mechanism: by reference

When you specify the **status** argument, EPC$DELETE_CONTEXT sets the first longword to zero at initiation. The second longword is always zero. Upon completion, a condition value returns in the first longword.

astadr

VMS Usage: ast_procedure
type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of the entry mask of the AST service routine to be executed when EPC$DELETE_CONTEXT completes.

If you specify the **astadr** argument, the AST routine executes in the same access mode as the caller of EPC$DELETE_CONTEXT.

astparam

VMS Usage: user_arg
type: longword (unsigned)
access: read only
mechanism: by value

The AST parameter to be passed to the AST service routine specified by the **astadr** argument. The **astparam** argument is a longword parameter.

Include this argument only when you specify the **astadr** argument.

flags

VMS Usage: vector_longword_unsigned
type: longword (unsigned)
access: read only
mechanism: by value

The **flags** argument is used to specify whether the event flag should be cleared upon service initiation and set upon completion. If not, the overhead of clearing and setting the event flag is eliminated. This bit is defined as the constant: EPC$M_NOEF.

The **flags** argument is also used to determine if an AST routine should be called upon service completion. If there is no need to set up an AST, the overhead is eliminated. Oracle Trace only calls the AST routine if the Oracle Trace command does not complete synchronously (for example, if it had to perform some I/O). This bit is defined as the constant: EPC$M_SYNCSTS.

If the EPC$M_SYNCSTS flag is set and the routine completes asynchronously, then EPC$SUCCESS is returned. If the flag is set and the routine completes synchronously (no I/O occurred), then EPC$_SYNC is returned.

Note

The following values must be set for the EPC$M_xxx bits in the **flags** argument. Oracle Trace provides include files for many languages containing the values of these bits (see Section 4.4.3). If no EPC$DEFINITIONS file exists for your language, you need to define the following constants in your instrumented code:

- EPC$M_SYNCSTS = 1
- EPC$M_NOEF = 2
- EPC$M_BITMAP = 4
- EPC$M_LONGREC = 8

Description

The EPC$DELETE_CONTEXT routine deletes the context variable. All associated context for Oracle Trace to collect resource utilization or cross-facility items for a particular context variable is deleted.

The value of the **context-variable** argument is set to zero upon completion.

A facility calls the EPC$DELETE_CONTEXT service only if an associated EPC$SET_CONTEXT call was previously issued and when execution of a thread completes to its entirety.

Return Values

EPC$_BADASTADR	Bad AST address specified.
EPC$_BADASTPRM	Bad AST parameter specified.
EPC$_BADSTATUS	Bad status argument passed.
EPC$_BADTHREAD	Bad context-variable argument passed.
EPC$_DISABLED	Collection has been disabled.
EPC$_INSARGS	Insufficient arguments specified.
EPC$_NOTHREAD	No context-variable argument was passed.
EPC$_NOTINSTALL	Oracle Trace software is not installed.
EPC$_SUCCESS	This call was successful.
EPC$_SYNC	The EPC$M_SYNC flag was set, and the call did not require an I/O.

EPC$END_EVENT

EPC$END_EVENT records the end of an event to the data collection file.

The EPC$END_EVENT routine completes asynchronously. It returns control to the caller after initiating the $END_EVENT processing, without waiting for I/O processing to complete.

Format

EPC$END_EVENT [efn] ,facility ,event-id ,handle [,context-variable] [,user-defined-buffer] [,status] [,astadr] [,astparam] [,flags]

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

efn

VMS Usage: ef_number
type: longword (unsigned)
access: read only
mechanism: by value

The OpenVMS event flag to be set when EPC$END_EVENT completes. The **efn** argument is a longword containing the address of this flag. Note that event flag 47 is reserved for Oracle Corporation.

Upon initiation, EPC$END_EVENT clears the specified event flag (or event flag 0 if **efn** was not specified). When EPC$END_EVENT returns, it sets the specified event flag (or event flag 0).

Overhead is significantly reduced if you do not use the **efn** argument. Use the EPC$M_NOEF flag to specify that the event flag should not be cleared or set. See the **flags** argument for information.

facility

VMS Usage: longword_unsigned
type: longword (unsigned)
access: read only
mechanism: by value

EPC$END_EVENT

The facility number for the calling facility. Facilities 1 through 2047 are registered with Oracle Corporation and are guaranteed to be unique across all systems. Facilities 2048 through 4097 are user-defined.

event-id

VMS Usage: longword_unsigned
type: longword (unsigned)
access: read only
mechanism: by value

The event identifier for the event. Event IDs range from 1 to 128.

handle

VMS Usage: context
type: longword (unsigned)
access: write only
mechanism: by reference

The address of the longword to retrieve a process unique event handle for this particular event.

The supplied longword **handle** is the same **handle** that was returned from the corresponding EPC$START_EVENT call for an event instance.

context-variable

VMS Usage: context
type: longword (unsigned)
access: read only
mechanism: by reference

The address of the longword used to identify the context in which this call appears. When specified, the value of the longword corresponds to the **context-variable** argument returned from the call to EPC$SET_CONTEXT.

user-defined-buffer

VMS Usage: user-defined-buffer
type: any binary or ASCII data
access: read only
mechanism: by descriptor—fixed length descriptor or buffer address

If the record buffer size value fits in a word (less than 65535 bytes), then the **user-defined-buffer** can contain the address of a descriptor pointing to a record buffer containing the item values for an event. If the record buffer is too large to fit in a descriptor, then **user-defined-buffer** can contain the address of a descriptor defined as a longword containing the size, followed by a pointer to the actual string, providing the EPC$M_LONGREC flag is set.

You must define facility-specific items using ASCIC data types, as described in Section 4.7.3.

Note that the item values are facility-specific; they do not include any resource utilization items. Oracle Trace handles the resource item values transparently. As a result, you do not need to specify the **user-defined-buffer** on the EPC$END_EVENT call if you want only the set of standard resource utilization items.

status

VMS Usage: io_status_block
type: quadword (unsigned)
access: write only
mechanism: by reference

When you specify the status argument, EPC$END_EVENT sets the first longword to zero at initiation. The second longword is always zero. Upon completion, a condition value returns in the first longword.

astadr

VMS Usage: ast_procedure
type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of the entry mask of the AST service routine to be executed when EPC$END_EVENT completes.

If **astadr** is specified, the AST routine executes in the same access mode as the caller of EPC$END_EVENT.

astparam

VMS Usage: user_arg
type: longword (unsigned)
access: read only
mechanism: by value

AST parameter to be passed to the AST service routine specified by the **astadr** argument. The **astparam** argument is a longword parameter.

Include this argument only when you specify the **astadr** argument.

flags

VMS Usage: vector_longword_unsigned
type: longword (unsigned)
access: read only

mechanism: by value

The **flags** argument is used to specify whether the event flag should be cleared upon service initiation and set upon completion. If not, the overhead of clearing and setting the event flag is eliminated. This bit is defined as the constant: EPC$M_NOEF.

The **flags** argument is also used to determine if an AST routine should be called upon service completion. If there is no need to set up an AST, the overhead is eliminated. Oracle Trace will only call the AST routine if the Oracle Trace command could not complete synchronously (for example, if it had to perform some I/O). This bit is defined as the constant: EPC$M_SYNCSTS.

If the EPC$M_SYNCSTS flag is set and the routine completes asynchronously, then EPC$SUCCESS is returned. If the flag is set and the routine was able to complete synchronously (no I/O occurred), then EPC$_SYNC is returned.

The bit EPC$M_LONGREC indicates that the **user-defined-buffer** is the address of a longword buffer, not a descriptor.

Note

The following values must be set for the EPC$M_xxx bits in the FLAGS argument. Oracle Trace provides include files for many languages containing the values of these bits (see Section 4.4.3). If no EPC$DEFINITIONS file exists for your language, you need to define the following constants in your instrumented code:

- EPC$M_SYNCSTS = 1
- EPC$M_NOEF = 2
- EPC$M_BITMAP = 4
- EPC$M_LONGREC = 8

Description

The EPC$END_EVENT routine enables a given facility to log an end event record to the data collection file. This service is called at the end of a duration event with various items determined by the item flags list for a given facility.

For performance reasons, Oracle Trace does not write every record directly to the data collection file. Instead, records are buffered and flushed to disk when the buffer is full. The EPC$END_EVENT routine completes asynchronously. It returns to the caller after copying the end event record to the buffer without waiting for any buffer flushes to complete.

Return Values

EPC$_BADASTADR	Bad AST address specified.
EPC$_BADASTPRM	Bad AST parameter specified.
EPC$_BADEVNT	Bad event ID argument passed.
EPC$_BADEVNTREC	Bad event record argument passed.
EPC$_BADFAC	Bad facility code passed.
EPC$_BADHANDL	Bad handle argument passed.
EPC$_BADSTATUS	Bad status argument passed.
EPC$_BADTHREAD	Bad context-variable argument passed.
EPC$_DCFCORRUPT	Data capture file is corrupt.
EPC$_DISABLED	Collection has been disabled.
EPC$_EVTNOTCOL	Event is not being collected.
EPC$_FACNOTCOL	Facility is not being collected.
EPC$_ILLBUFLEN	The record buffer was larger than the maximum.
EPC$_INSARGS	Insufficient arguments specified.
EPC$_NODCFEXISTS	The specified data capture file does not exist.
EPC$_NOEVNT	No event ID argument specified.
EPC$_NOFAC	No facility argument specified.
EPC$_NOHANDL	No handle argument passed.
EPC$_NOTINSTALL	Oracle Trace software is not installed.
EPC$_SUCCESS	This call was successful.
EPC$_SYNC	The EPC$M_SYNC flag was set, and the call did not require an I/O.

EPC$END_EVENTW

EPC$END_EVENTW records the end of an event to the data collection file. This service is called at the end of a duration event with various items determined by the item flags list for a given facility.

The EPC$END_EVENTW routine completes synchronously. It returns control to the caller when the $END_EVENTW processing completes.

Format

EPC$END_EVENTW [efn] ,facility ,event-id ,handle [,context-variable] [,user-defined-buffer] [,status] [,astadr] [,astparam] [,flags]

Description

For asynchronous completion, use the EPC$END_EVENT routine. EPC$END_EVENT returns to the caller after initiating the processing without waiting for I/O to complete.

For performance reasons, Oracle Trace does not write every record directly to the data collection file. Instead, records are buffered and flushed to disk when the buffer is full. The EPC$END_EVENTW routine completes synchronously. That is, if a buffer flush needs to be performed, it waits for I/O to complete before returning to the caller.

In all other respects, EPC$END_EVENTW is identical to EPC$END_EVENT. Refer to EPC$END_EVENT for all other information about the EPC$END_EVENTW service routine.

EPC$EVENT

EPC$EVENT records the occurrence of a point event to the data collection file.

The EPC$EVENT routine completes asynchronously. It returns control to the caller after initiating the $EVENT processing without waiting for I/O processing to complete.

Format

EPC$EVENT [efn] ,facility ,event-id [,context-variable] [,user-defined-buffer] [,status] [,astadr] [,astparam] [,flags]

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

efn

VMS Usage: ef_number
type: longword (unsigned)
access: read only
mechanism: by value

The OpenVMS event flag to be set when EPC$EVENT completes. The **efn** argument is a longword containing the address of this flag. Note that event flag 47 is reserved for Oracle Corporation.

Upon initiation, EPC$EVENT clears the specified event flag (or event flag 0 if **efn** was not specified). When EPC$EVENT returns, it sets the specified event flag (or event flag 0).

Overhead is significantly reduced if you do not use the **efn** argument. Use the EPC$M_NOEF flag to specify that the event flag should not be cleared or set. See the **flags** argument for information.

facility

VMS Usage: longword_unsigned
type: longword (unsigned)
access: read only
mechanism: by value

The facility number for the calling facility. Facilities 1 through 2047 are registered with Oracle Corporation and are guaranteed to be unique across all systems. Facilities 2048 through 4097 are user-defined.

event-id

VMS Usage: longword_unsigned
type: longword (unsigned)
access: read only
mechanism: by value

The event identifier for an event. Event IDs range from 1 to 128.

context-variable

VMS Usage: context
type: longword (unsigned)
access: read only
mechanism: by reference

The address of the longword used to identify the context variable. When specified, the value of the longword corresponds to the **context-variable** argument returned from the call to EPC$SET_CONTEXT.

user-defined-buffer

VMS Usage: user-defined-buffer
type: any binary or ASCII data
access: read only
mechanism: by descriptor—fixed length descriptor or buffer address

If the record buffer size value fits in a word (less than 65535 bytes), then the **user-defined-buffer** can contain the address of a descriptor pointing to a record buffer containing the item values for an event. If the record buffer is too large to fit in a descriptor, then **user-defined-buffer** can contain the address of a descriptor defined as a longword containing the size, followed by a pointer to the actual string, providing the EPC$M_LONGREC flag is set.

You must define facility-specific items using ASCIC data types, as described in Section 4.7.3.

Note that the item values are facility-specific; they do not include any resource utilization items. Oracle Trace handles the resource item values transparently. As a result, you do not need to specify the **user-defined-buffer** on the EPC$EVENT call if you only want the set of standard resource utilization items.

status

VMS Usage: io_status_block
type: quadword (unsigned)
access: write only
mechanism: by reference

When you specify the **status** argument, EPC$EVENT sets the first longword to zero at initiation. The second longword is always zero. Upon completion, a condition value returns in the first longword.

astadr

VMS Usage: ast_procedure
type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of the entry mask of the AST service routine to be executed when EPC$EVENT completes.

If the **astadr** is specified, the AST routine executes in the same access mode as the caller of EPC$EVENT.

astparam

VMS Usage: user_arg
type: longword (unsigned)
access: read only
mechanism: by value

The AST parameter to be passed to the AST service routine specified by the **astadr** argument. The **astparam** argument is a longword parameter.

Include this argument only when you specify the **astadr** argument.

flags

VMS Usage: vector_longword_unsigned
type: longword (unsigned)
access: read only
mechanism: by value

The **flags** argument is used to specify whether the event flag should be cleared upon service initiation and set upon completion. If not, the overhead of clearing and setting the event flag is eliminated. This bit is defined as the constant: EPC$M_NOEF.

The **flags** argument is also used to determine if an AST routine should be called upon service completion. If there is no need to set up an AST, the overhead is eliminated. Oracle Trace only calls the AST routine if the Oracle Trace command does not complete synchronously (for example, if it had to perform some I/O). This bit is defined as the constant: EPC$M_SYNCSTS.

If the EPC$M_SYNCSTS flag is set and the routine completes asynchronously, then EPC$SUCCESS is returned. If the flag is set and the routine completes synchronously (no I/O occurred), then EPC$_SYNC is returned.

The bit EPC$M_LONGREC indicates that the **user-defined-buffer** is the address of a longword buffer, not a descriptor.

Note

The following values must be set for the EPC$M_xxx bits in the **flags** argument. Oracle Trace provides include files for many languages containing the values of these bits (see Section 4.4.3). If no EPC$DEFINITIONS file exists for your language, you need to define the following constants in your instrumented code:

- EPC$M_SYNCSTS = 1
- EPC$M_NOEF = 2
- EPC$M_BITMAP = 4
- EPC$M_LONGREC = 8

Description

EPC$EVENT routine enables a given facility to record the occurrence of a point event to the data collection file.

For performance reasons, Oracle Trace does not write every record directly to the data collection file. Instead, records are buffered and flushed to disk when the buffer is full. The EPC$EVENT routine completes asynchronously. That is, it returns to the caller after copying the point event record to the buffer without waiting for any buffer flushes to complete.

Return Values

EPC$_BADASTADR	Bad AST address specified.
EPC$_BADASTPRM	Bad AST parameter specified.
EPC$_BADEVNT	Bad event ID argument passed.
EPC$_BADEVNTREC	Bad event record argument passed.
EPC$_BADFAC	Bad facility code passed.
EPC$_BADSTATUS	Bad status argument passed.
EPC$_BADTHREAD	Bad context-variable argument passed.
EPC$_DCFCORRUPT	Data capture file is corrupt.
EPC$_DISABLED	Collection has been disabled.
EPC$_EVTNOTCOL	Event is not being collected.
EPC$_FACNOTCOL	Facility is not being collected.
EPC$_ILLBUFLEN	The record buffer was larger than the maximum.
EPC$_INSARGS	Insufficient arguments specified.
EPC$_NODCFEXISTS	The specified data capture file does not exist.
EPC$_NOEVNT	No event ID argument specified.
EPC$_NOFAC	No facility argument specified.
EPC$_NOTINSTALL	Oracle Trace software is not installed.
EPC$_SUCCESS	This call was successful.
EPC$_SYNC	The EPC$M_SYNC flag was set, and the call did not require an I/O.

EPC$EVENTW

EPC$EVENTW records the occurrence of a point event to the data collection file and waits for I/O processing to complete before returning control.

The EPC$EVENTW routine completes synchronously. It returns to the caller when the $EVENTW processing completes.

Format

EPC$EVENTW [efn] ,facility ,event-id [,context-variable] [,user-defined-buffer] [,status] [,astadr] [,astparam] [,flags]

Description

For asynchronous completion, use the EPC$EVENT service; EPC$EVENT returns to the caller after initiating the processing without waiting for I/O to complete.

For performance reasons, Oracle Trace does not write every record directly to the data collection file. Instead, records are buffered and flushed to disk when the buffer is full. The EPC$EVENTW routine completes synchronously. That is, if a buffer flush needs to be performed, it will wait for the I/O to complete before returning to the caller.

In all other respects, EPC$EVENTW is identical to EPC$EVENT. Refer to EPC$EVENT for all other information about the EPC$EVENTW service routine.

EPC$GET_CF_ITEMS

The GET_CF_ITEMS service routine returns all values for the cross-facility items for this process or thread of execution (if the **context-variable** argument is specified).

Format

EPC$GET_CF_ITEMS [efn] ,cf-items [,context-variable] [,flags]

Returns

VMS Usage:	condition_value
type:	longword (unsigned)
access:	write only
mechanism:	by value

Arguments

efn

VMS Usage:	ef_number
type:	longword (unsigned)
access:	read only
mechanism:	by value

Number of the OpenVMS event flag to be set when EPC$GET_CF_ITEMS completes. The **efn** argument is a longword containing this number. If the EPC$M_NOEF flag is set in the **flags** longword, then this flag is not set.

cf-items

VMS Usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	write only
mechanism:	by reference

The address of a list of contiguous longwords to return the cross-facility item values. The number of elements in this list is indicated by the EPC$K_NUM_CROSS_FAC_ITEMS, currently 14. The list of cross-facility item values is returned from the call to EPC$GET_CF_ITEMS by the client portion of the facility.

context-variable
VMS Usage: context
type: longword (unsigned)
access: read only
mechanism: by reference

The address of the longword used to identify the context variable. When specified, the value of the longword corresponds to the **context-variable** argument returned from the call to EPC$SET_CONTEXT.

flags
VMS Usage: flags
type: longword (unsigned)
access: read only
mechanism: by value

If the EPC$M_NOEF flag bit is set, then the **efn** is not cleared upon calling or upon completion of the Oracle Trace service routine.

Description

The EPC$GET_CF_ITEMS service routine returns all values for the cross-facility items for this process or thread of execution (if the **context-variable** argument is specified). The EPC$GET_CF_ITEMS routine is used by client/server facilities that need to pass the values of the cross-facility items from the client process of the facility to the server process of the facility. The EPC$GET_CF_ITEMS routine must be called by the client process of the facility to retrieve a list of cross-facility item values of the client process. It is the responsibility of the facility to transport the cross-facility item values to the server process. The associated EPC$SET_CF_ITEMS routine must be called by the server process of the facility to set the values of the cross-facility item values. If the client process of the facility is multithreaded, it must also call the EPC$SET_CONTEXT routine prior to the call to the EPC$GET_CF_ ITEMS routine in order to create context for the thread running in the client process.

The EPC$GET_CF_ITEMS routine completes in a synchronous nature.

Return Values

EPC$_BADTHREAD	Bad context-variable argument passed.
EPC$_BADCF-ITEMS	Bad list of cross-facility items argument passed.
EPC$_DISABLED	Collection has been disabled.
EPC$_INSARGS	Insufficient arguments specified.
EPC$_NOTINSTALL	Oracle Trace software is not installed.
EPC$_SUCCESS	This call was successful.
EPC$_SYNC	The EPC$M_SYNC flag was set, and the call did not require an I/O.

EPC$INIT

EPC$INIT registers a facility with Oracle Trace.

Format

EPC$INIT [efn] ,facility ,version [,registration-id] [,eventflgs] [,itemflgs] [,status] [,astadr] [,astparam] [,flags] [,min-event] [,max-event] [,min-item] [,max-item]

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

efn

VMS Usage: ef_number
type: longword (unsigned)
access: read only
mechanism: by value

The OpenVMS event flag to be set when EPC$INIT completes. The **efn** argument is a longword containing the address of this flag.

Upon initiation, EPC$INIT clears the specified event flag (or event flag 0 if **efn** was not specified). When EPC$INIT returns, it sets the specified event flag (or event flag 0).

Overhead is significantly reduced if you do not use the **efn** argument. Use the EPC$M_NOEF flag to specify that the event flag should not be cleared or set. See the **flags** argument for information.

facility

VMS Usage: longword_unsigned
type: longword (unsigned)
access: read only
mechanism: by value

The facility number for the calling facility. Facilities 1 through 2047 are registered with Oracle Corporation and are guaranteed to be unique across all systems. Facilities 2048 through 4097 are user-defined.

version

VMS Usage: char_string
type: character-coded text string
access: read only
mechanism: by descriptor

The string, up to 10 characters, identifying the current version of the facility. Note that the string must match identically with that specified in the facility definition.

registration-id

VMS Usage: char_string
type: character-coded text string
access: read only
mechanism: by descriptor

A string of up to 255 characters containing the identifier this facility wants to associate with this OpenVMS process. This identifier is "registered" in the Oracle Trace administration database along with other information for this process. The registration identifier gives the user the ability to schedule data collection for a group of processes. For example, if ACMS uses this identifier to describe a particular application, the user can then choose to schedule data collection for all processes that belong to that particular application, rather than collect data on all ACMS applications.

The **registration-id** can contain the following characters:

- A through Z
- a through z
- 0 through 9
- Underscore (_)
- Hyphen (-)
- Dollar sign ($)

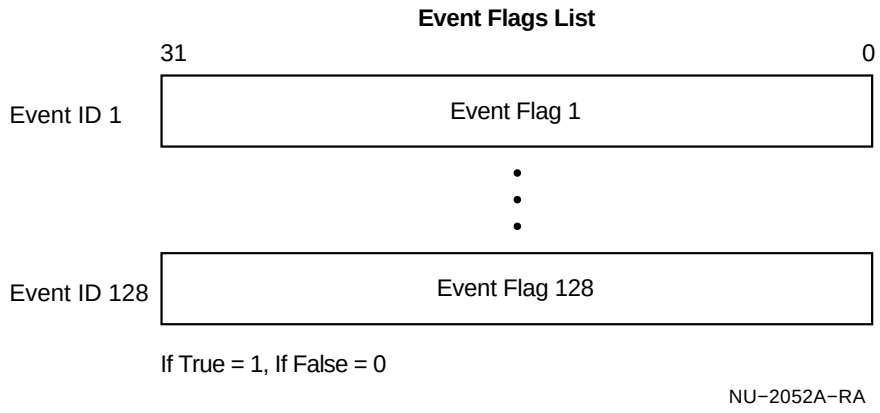
eventflgs

VMS Usage: vector_longword_unsigned
type: longword (unsigned)
access: write only
mechanism: by reference

A list of 128 longword Boolean values. Each list element corresponds to an event identifier and designates whether data collection is enabled for a particular event.

A facility needs only one copy of the event flags. The event flags should be declared as non-volatile structures. The memory allocated should exist for the lifetime of the image activation.

The following figure shows the physical structure of the event flags:



The EPC$M_BITMAP constant allows you to use a bitmap array instead of an array of longwords. Applications using a language that supports bit testing can take advantage of this feature to use less memory for the event flags list. See the **flags** argument for information.

itemflgs

VMS Usage: item_flags_list
type: longword (unsigned)
access: write only
mechanism: by reference

A list of 128 item flags. Each set of item flags, referred to as a list element, is 128 bits and corresponds to a particular event. The flags designate which items are to be collected for a particular event. If bit 0 is set for item flags 1, then item 1 is to be collected for event 1.

A facility needs only one copy of the item flags. The item flags should be declared as non-volatile structures. The memory allocated should exist for the lifetime of the image activation.

The following figure shows the physical structure of the item flags list:

status

VMS Usage: io_status_block
type: quadword (unsigned)
access: write only
mechanism: by reference

When you specify the **status** argument, EPC$INIT sets the first longword to zero at initiation. The second longword is always zero. Upon completion, a condition value returns in the first longword.

astadr

VMS Usage: ast_procedure
type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of the entry mask of the AST service routine to be executed when EPC$INIT completes.

If an **astadr** is specified and the EPC$M_CALLBACK flag is set, then astadr is called with the optional astparam whenever a collection is activated for the facility. This is useful if you use global flags or have something in your instrumentation that needs to be reset each time a new collection is activated.

astparam

VMS Usage: user_arg
type: longword (unsigned)
access: read only
mechanism: by value

The AST parameter to be passed to the AST service routine specified by the **astadr** argument. The **astparam** argument is a longword parameter.

Include this argument only when you specify the **astadr** argument.

flags

VMS Usage: vector_longword_unsigned
type: longword (unsigned)
access: read only
mechanism: by value

The **flags** argument specifies whether to use an array of longwords or a bitmap for each event in order to determine if data collection is active. This allows applications using a language that supports bit testing to use less memory, while not adversely affecting users of other languages. This bit is defined as the constant: EPC$M_BITMAP.

The **flags** longword is also used to specify whether the event flag should be cleared upon service initiation and set upon completion. If not, the overhead of clearing and setting the event flag is eliminated. This bit is defined as the constant: EPC$M_NOEF.

Note

The following values must be set for the EPC$M_xxx bits in the **flags** argument. Oracle Trace provides include files for many languages containing the values of these bits. If no EPC$DEFINITIONS file exists for your language, you need to define the following constants in your instrumented code:

- EPC$M_SYNCSTS = 1
- EPC$M_NOEF = 2
- EPC$M_BITMAP = 4
- EPC$M_LONGREC = 8
- EPC$M_CALLBACK = 64

min-event

VMS Usage: minimum event number
type: longword (unsigned)
access: read only
mechanism: by value

The minimum and maximum event numbers describe how large the longword array or event bitmap is. If this is left blank, the default value is 1. If used, this argument allows a savings in the amount of virtual memory a process must allocate for the Oracle Trace event collection flags list.

If the **min-event** argument is used, Oracle Trace negatively offsets the **min-event** into the first event flag. To check if a particular event flag is set, use a negative offset into the event collection flags list. For example, if **min-event** = 50, then the first event flag in the event flags structure corresponds to event ID 50.

max-event

VMS Usage: maximum event number
type: longword (unsigned)
access: read only
mechanism: by value

The maximum and minimum event numbers describe how large the longword array or event bitmap is. If this is left blank, the default value is 128 (the largest number of events possible for a facility). If used, this argument allows a savings in the amount of virtual memory a process must allocate for the Oracle Trace event flags. The **max-event** value must be greater than or equal to the **min-event**.

The **max-event** value may be greater than the actual number of events in the facility definition. This allows you to add events to the facility definition without requiring a new image creation. Note that the buffers used for the event collection flags list must be large enough to support the maximum number of events, regardless of whether all the event flags are ever used.

min-item

VMS Usage: minimum item number
type: longword (unsigned)
access: read only
mechanism: by value

The minimum and maximum item numbers describe how large the item bitmap is. If this is left blank, the default value is 1. If used, this allows a savings in the amount of virtual memory a process must allocate for the Oracle Trace item flags. If the **min-item** argument is used it means that Oracle Trace negatively offsets the **min-item** into the first item flags list. To check if a particular item flag is set, use a negative offset into the item flags list. For example, if **min-item** = 50, then the first item flag in the item flags list structure corresponds to item ID 50.

max-item

VMS Usage: maximum item number
type: longword (unsigned)
access: read only
mechanism: by value

The maximum and minimum item numbers describe how large the item bitmap is. If this is left blank, the default value is 128 (the largest number of items possible for a facility). If used, this argument allows a savings in the amount of virtual memory a process must allocate for the Oracle Trace item flags. The **max-item** value must be greater than or equal to the **min-item** value.

The **max-item** value may be greater than the actual number of items in the facility definition. This allows you to add items to the facility definition without requiring a new image creation. Note that the buffers used for the item flags list must be large enough to support the maximum number of items, regardless of whether all the item flags are ever used.

Note that item IDs 100 through 128 are reserved for Oracle Corporation. If you want to collect the Oracle Trace resource utilization items, with your events, **max-item** cannot be set below 128.

Description

The EPC$INIT routine registers a facility with Oracle Trace. By registering, active and pending collections are enabled for this facility (provided that data collection is scheduled using the SCHEDULE command).

The EPC$INIT routine completes in both a synchronous and asynchronous nature. The initial values of the **eventflgs** and **itemflgs** arguments, if specified, are returned synchronously, which allows the facility to determine which events and associated items are to be collected immediately following the call to EPC$INIT. However, the values of the **eventflgs** and **itemflgs** can change as collections become active. The values of the **eventflgs** and **itemflgs** change to reflect the values in the corresponding facility selection for the data collection.

When a collection is activated, the Oracle Trace Registrar notifies the facility by setting the appropriate event and item flags for that collection. If the facility had specified an AST routine and passed a flag argument with the EPC$M_CALLBACK flag set, then the AST routine is automatically called with the optional ASTPARAM when the collection is activated.

The **status** argument is returned when EPC$INIT completes.

Return Values

EPC$_BADEVNTFLGS	Bad facility event flags passed.
EPC$_BADFAC	Bad facility code passed.
EPC$_BADFACVER	Bad facility version passed.
EPC$_BADITMFLGS	Bad facility item flags passed.
EPC$_BADREGID	Bad facility registration group ID passed.
EPC$_DISABLED	Data collection disabled.
EPC$_FACVERREQ	Facility version required.
EPC$_INIT2	EPC$INIT called twice.
EPC$_NODCFEXISTS	No data capture file exists.
EPC$_NOREGEXISTS	No registrar process exists on this node.
EPC$_NOTINSTALL	Oracle Trace software is not installed.
EPC$_REGTIMEOUT	Registrar did not respond within the time set by the logical EPC$BE_REGTIMEOUT (default is 90 seconds).
EPC$_SUCCESS	Routine completed successfully.
EPC$_SYNC	The EPC$M_SYNC flag was set, and the call did not require an I/O.

EPC$SET_CF_ITEMS

The SET_CF_ITEMS service routine sets all values for the cross-facility items for this process or thread of execution (if context-variable argument is specified).

Format

EPC$SET_CF_ITEMS [efn] ,cf-items [,context-variable] [,flags]

Returns

VMS Usage:	condition_value
type:	longword (unsigned)
access:	write only
mechanism:	by value

Arguments

efn

VMS Usage:	ef_number
type:	longword (unsigned)
access:	read only
mechanism:	by value

The number of the OpenVMS event flag to be set when EPC$SET_CF_ITEMS completes. The **efn** argument is a longword containing this number. If the EPC$M_NOEF flag is set in the **flags** longword, then this flag is not set.

cf-items

VMS Usage:	vector_longword_unsigned
type:	longword (unsigned)
access:	write only
mechanism:	by reference

The address of a list of contiguous longword cross-facility item values. The number of elements in this list is indicated by the EPC$K_NUM_CROSS_FAC_ ITEMS, currently 14. The list of cross-facility item values is returned from the call to EPC$GET_CF_ITEMS by the client portion of the facility.

context-variable

VMS Usage: context
type: longword (unsigned)
access: read only
mechanism: by reference

The address of the longword used to identify the context variable. When specified, the value of the longword corresponds to the **context-variable** argument returned from the call to EPC$SET_CONTEXT.

flags

VMS Usage: flags
type: longword (unsigned)
access: read only
mechanism: by value

If the EPC$M_NOEF flag bit is set, then the **efn** is not cleared upon calling or upon completion of the Oracle Trace service routine.

Description

The EPC$SET_CF_ITEMS service routine sets all values for the cross-facility items for this process or thread of execution (if the **context-variable** argument is specified). The EPC$SET_CF_ITEMS routine is used by client/server facilities that need to pass the values of the cross-facility items from the client process of the facility to the server process of the facility. The associated EPC$GET_CF_ITEMS routine must be called by the client process of the facility to retrieve a list of cross-facility item values of the client process. It is the responsibility of the facility to transport the cross-facility item values to the server process. If the server process of the facility is multithreaded, it must also call the EPC$SET_CONTEXT routine prior to the call to the EPC$SET_CF_ITEMS routine in order to create context for the thread running in the server process.

If your application sets one of the cross-facility items, Oracle Corporation recommends that you register this usage with the Oracle Trace development group. See Appendix B for information on how to do this.

The EPC$SET_CF_ITEMS routine completes synchronously.

Return Values

EPC$_BADTHREAD	Bad context-variable argument passed.
EPC$_BADCF-ITEMS	Bad list of cross-facility items argument passed.
EPC$_DISABLED	Collection has been disabled.
EPC$_INSARGS	Insufficient arguments specified.
EPC$_NOTINSTALL	Oracle Trace software is not installed.
EPC$_SUCCESS	This call was successful.
EPC$_SYNC	The EPC$M_SYNC flag was set, and the call did not require an I/O.

EPC$SET_CF_VALUE

SET_CF_VALUE service routine assigns a value to a cross-facility item.

Format

EPC$SET_CF_VALUE [efn] ,item-num ,item-val [,context-variable] [,item-adr] [,flags]

Returns

VMS Usage: condition_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

efn

VMS Usage: ef_number
type: longword (unsigned)
access: read only
mechanism: by value

The number of the OpenVMS event flag to be set when EPC$SET_CF_VALUE completes. The **efn** argument is a longword containing this number. If the EPC$M_NOEF flag is set in the **flags** longword, then this flag is not set.

item-num

VMS Usage: cross_fac_item_number
type: longword (unsigned)
access: read only
mechanism: by value

The **item-num** contains the cross-facility item number whose value is to be set to the value indicated by the **item-val** argument.

item-val

VMS Usage: item_value
type: longword (unsigned)
access: read only
mechanism: by value

The **item-val** contains the numeric value (in decimal) to set the cross-facility item number, indicated by the **item-num** argument.

context-variable

VMS Usage: context
type: longword (unsigned)
access: read only
mechanism: by reference

The address of the longword used to identify the context variable. When specified, the value of the longword corresponds to the **context-variable** argument returned from the call to EPC$SET_CONTEXT.

item-adr

VMS Usage: cross_fac_addr
type: longword (unsigned)
access: write only
mechanism: by reference

If specified, **item-adr** will contain a pointer to the location where the corresponding cross-facility item resides. The location will be user-mode accessible memory, created by Oracle Trace during image activation time (and thus accessible during the entire image execution to avoid any potential access violations). The facility can subsequently assign a value to the item without having to issue a call to the EPC$SET_CF_VALUE service routine (thus avoiding call overhead).

flags

VMS Usage: flags
type: longword (unsigned)
access: read only
mechanism: by value

If the EPC$M_NOEF flag bit is set, then the **efn** is not cleared upon calling or upon completion of the Oracle Trace service routine.

Description

The EPC$SET_CF_VALUE service routine sets the value of the corresponding item. It optionally returns the address of the cross-facility item in the **item-adr** argument, so that it can be assigned later (more efficiently) by the facility.

If your application sets one of the cross-facility items, Oracle Corporation recommends that you register this usage with the Oracle Trace development group. See Appendix B for information on how to do this.

The EPC$SET_CF_VALUE service completes in a synchronous nature.

Return Values

EPC$_BADITEM-ADR	Bad item-adr argument passed.
EPC$_BADITEM-NUM	Bad item-num argument passed.
EPC$_BADITEM-VAL	Bad item-num argument passed.
EPC$_BADTHREAD	Bad context-variable argument passed.
EPC$_DISABLED	Collection has been disabled.
EPC$_INSARGS	Insufficient arguments specified.
EPC$_NOTINSTALL	Oracle Trace software is not installed.
EPC$_SUCCESS	This call was successful.
EPC$_SYNC	The EPC$M_SYNC flag was set, and the call did not require an I/O.

EPC$SET_CONTEXT

EPC$SET_CONTEXT records a context variable for a new or existing thread so that resource utilization and cross-facility items can be collected for multithreaded facilities.

Format

EPC$SET_CONTEXT [efn] ,context-variable [,status] [,astadr] [,astparam] [,flags]

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

efn

VMS Usage: ef_number
type: longword (unsigned)
access: read only
mechanism: by value

The OpenVMS event flag to be set when EPC$SET_CONTEXT completes. The **efn** argument is a longword containing the address of this flag. Note that event flag 47 is reserved for Oracle Corporation.

Upon initiation, EPC$SET_CONTEXT clears the specified event flag (or event flag 0 if **efn** was not specified). When EPC$SET_CONTEXT returns, it sets the specified event flag (or event flag 0).

Overhead is significantly reduced if you do not use the **efn** argument. Use the EPC$M_NOEF flag to specify that the event flag should not be cleared or set. See the **flags** argument for information.

context-variable

VMS Usage: context
type: longword(unsigned)
access: modify
mechanism: by reference

The address of the longword that contains the context variable to which context should be set.

If a zero value is passed at the address specified, then EPC$SET_CONTEXT returns a value that identifies the new thread context. If the address contains a non-zero context variable, then Oracle Trace uses this value as the context of an existing thread and tracks resource utilization statistics and cross-facility items for the thread.

status

VMS Usage: io_status_block
type: quadword (unsigned)
access: write only
mechanism: by reference

When you specify the **status** argument, EPC$SET_CONTEXT sets the first longword to zero at initiation. The second longword is always zero. Upon completion, a condition value returns in the first longword.

astadr

VMS Usage: ast_procedure
type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of the entry mask of the AST service routine to be executed when EPC$SET_CONTEXT completes. The **astadr** is the address of the entry mask of this routine.

If you specify the **astadr**, the AST routine executes in the same access mode as the caller of EPC$SET_CONTEXT.

astparam

VMS Usage: user_arg
type: longword (unsigned)
access: read only
mechanism: by value

The AST parameter to be passed to the AST service routine specified by the **astadr** argument. The **astparam** argument is a longword parameter.

Include this argument only when you specify the **astadr** argument.

flags

VMS Usage: vector_longword_unsigned
type: longword (unsigned)
access: read only
mechanism: by value

The **flags** argument is used to specify whether the event flag should be cleared upon service initiation and set upon completion. If not, the overhead of clearing and setting the event flag is eliminated. This bit is defined as the constant: EPC$M_NOEF.

The **flags** argument is also used to determine if an AST routine should be called upon service completion. If there is no need to set up an AST, the overhead is eliminated. Oracle Trace only calls the AST routine if the Oracle Trace command does not complete synchronously (for example, if it had to perform some I/O). This bit is defined as the constant: EPC$M_SYNCSTS.

If the EPC$M_SYNCSTS flag is set and the routine completes asynchronously, then EPC$SUCCESS is returned. If the flag is set and the routine completes synchronously (no I/O occurred), then EPC$_SYNC is returned.

Note

The following values must be set for the EPC$M_xxx bits in the **flags** argument. Oracle Trace provides include files for many languages containing the values of these bits (see Section 4.4.3). If no EPC$DEFINITIONS file exists for your language, you need to define the following constants in your instrumented code:

- EPC$M_SYNCSTS = 1
- EPC$M_NOEF = 2
- EPC$M_BITMAP = 4
- EPC$M_LONGREC = 8

Description

The EPC$SET_CONTEXT routine sets context for a new thread or an existing thread so that resource utilization and cross-facility items can be collected for multithreaded facilities. This routine is called by a facility when a thread context is initially created or reentered (by way of a context switch) when resource utilization items are to be collected on a per-thread basis.

If a zero value is passed at the address specified, then EPC$SET_CONTEXT returns a value that identifies the new thread context.

The associated EPC$DELETE_CONTEXT routine must be called by a multithreaded facility when a thread executes to its entirety in order to delete context information that Oracle Trace maintains on behalf of the facility.

Return Values

EPC$_BADASTADR	Bad AST address specified.
EPC$_BADASTPRM	Bad AST parameter specified.
EPC$_BADSTATUS	Bad status argument passed.
EPC$_BADTHREAD	Bad context-variable argument passed.
EPC$_DISABLED	Collection has been disabled.
EPC$_INSARGS	Insufficient arguments specified.
EPC$_NOTHREAD	No context-variable argument was passed.
EPC$_NOTINSTALL	Oracle Trace software is not installed.
EPC$_SUCCESS	This call was successful.
EPC$_SYNC	The EPC$M_SYNC flag was set, and the call did not require an I/O.

EPC$START_EVENT

EPC$START_EVENT records the start of an event to the data collection file.

The EPC$START_EVENT routine completes asynchronously. It returns to the caller after initiating the $START_EVENT processing without waiting for I/O processing to complete.

Format

EPC$START_EVENT [efn] ,facility ,event-id ,handle [,context-variable] [,user-defined-buffer] [,status] [,astadr] [,astparam] [,flags]

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

efn

VMS Usage: ef_number
type: longword (unsigned)
access: read only
mechanism: by value

The OpenVMS event flag to be set when EPC$START_EVENT completes. The **efn** argument is a longword containing the address of this flag. Note that event flag 47 is reserved for Oracle Corporation.

Upon initiation, EPC$START_EVENT clears the specified event flag (or event flag 0 if **efn** was not specified). When EPC$START_EVENT returns, it sets the specified event flag (or event flag 0).

Overhead is significantly reduced if you do not use the **efn** argument. Use the EPC$M_NOEF flag to specify that the event flag should not be cleared or set. See the **flags** argument for information.

facility

VMS Usage: longword_unsigned
type: longword (unsigned)
access: read only
mechanism: by value

The facility number for the calling facility. Facilities 1 through 2047 are registered with Oracle Corporation and are guaranteed to be unique across all systems. Facilities 2048 through 4097 are user-defined.

event-id

VMS Usage: longword_unsigned
type: longword (unsigned)
access: read only
mechanism: by value

The event identifier for an event. Event IDs range from 1 to 128.

handle

VMS Usage: context
type: longword (unsigned)
access: write only
mechanism: by reference

The address of the longword to store a process unique event handle for this particular event. The **handle** is used in the corresponding EPC$END_EVENT call for an event.

context-variable

VMS Usage: context
type: longword (unsigned)
access: read only
mechanism: by reference

The address of the longword used to identify the context in which this call appears. When specified, the value of the longword corresponds to the **context-variable** argument returned from the call to EPC$SET_CONTEXT.

user-defined-buffer

VMS Usage: user-defined-buffer
type: any binary or ASCII data
access: read only
mechanism: by descriptor—fixed length descriptor or buffer address

If the record buffer size value fits in a word (less than 65535 bytes), then the **user-defined-buffer** can contain the address of a descriptor pointing to a record buffer containing the item values for an event. If the record buffer is too large to fit in a descriptor, then **user-defined-buffer** can contain the address of a descriptor defined as a longword containing the size, followed by a pointer to the actual string, providing the EPC$M_LONGREC flag is set.

You must define facility-specific items using ASCIC data types, as described in Section 4.7.3.

Note that the item values are facility-specific; they do not include any resource utilization items. Oracle Trace handles the resource item values transparently. As a result, you do not need to specify the **user-defined-buffer** on the EPC$START_EVENT call if you want only the set of standard resource utilization items.

status

VMS Usage: io_status_block
type: quadword (unsigned)
access: write only
mechanism: by reference

When you specify the **status** argument, EPC$START_EVENT sets the first longword to zero at initiation. The second longword is always zero. Upon completion, a condition value returns in the first longword.

astadr

VMS Usage: ast_procedure
type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of the entry mask of the AST service routine to be executed when EPC$START_EVENT completes.

If you specify the **astadr**, the AST routine executes in the same access mode as the caller of EPC$START_EVENT.

astparam

VMS Usage: user_arg
type: longword (unsigned)
access: read only
mechanism: by value

The AST parameter to be passed to the AST service routine specified by the **astadr** argument. The **astparam** argument is a longword parameter.

Include this argument only when you specify the **astadr** argument.

flags

VMS Usage: vector_longword_unsigned
type: longword (unsigned)
access: read only

mechanism: by value

The **flags** argument is used to specify whether the event flag should be cleared upon service initiation and set upon completion. If not, the overhead of clearing and setting the event flag is eliminated. This bit is defined as the constant: EPC$M_NOEF.

The FLAGS longword is also used to determine if an AST routine should be called upon service completion. If there is no need to set up an AST, the overhead is eliminated. Oracle Trace only calls the AST routine if the Oracle Trace command does not complete synchronously (for example, if it had to perform some I/O). This bit is defined as the constant: EPC$M_SYNCSTS.

If the EPC$M_SYNCSTS flag is set and the routine completes asynchronously, then EPC$SUCCESS is returned. If the flag is set and the routine completes synchronously (no I/O occurred), then EPC$_SYNC is returned.

The bit EPC$M_LONGREC indicates that the **user-defined-buffer** is the address of a longword buffer, not a descriptor.

Note

The following values must be set for the EPC$M_xxx bits in the **flags** argument. Oracle Trace provides include files for many languages containing the values of these bits (see Section 4.4.3). If no EPC$DEFINITIONS file exists for your language, you need to define the following constants in your instrumented code:

- EPC$M_SYNCSTS = 1
- EPC$M_NOEF = 2
- EPC$M_BITMAP = 4
- EPC$M_LONGREC = 8

Description

The EPC$START_EVENT routine enables a given facility to log the start of an event to the data collection file. This routine is called at the beginning of a duration event that can have various facility-specific items associated with it. The items recorded are determined by the item flags list for the facility. A call to EPC$END_EVENT follows this call where the event ends.

For performance reasons, Oracle Trace does not write every record directly to the data collection file. Instead, records are buffered and flushed to disk when the buffer is full. The EPC$START_EVENT routine completes asynchronously. That is, it returns to the caller after copying the start event record to the buffer without waiting for any buffer flushes to complete.

Return Values

EPC$_BADASTADR	Bad AST address specified.
EPC$_BADASTPRM	Bad AST parameter specified.
EPC$_BADEVNT	Bad event ID argument passed.
EPC$_BADEVNTREC	Bad event record argument passed.
EPC$_BADFAC	Bad facility code passed.
EPC$_BADHANDL	Bad handle argument passed.
EPC$_BADSTATUS	Bad status argument passed.
EPC$_BADTHREAD	Bad context-variable argument passed.
EPC$_DCFCORRUPT	Data capture file is corrupt.
EPC$_DISABLED	Collection has been disabled.
EPC$_EVTNOTCOL	Event is not being collected.
EPC$_FACNOTCOL	Facility is not being collected.
EPC$_ILLBUFLEN	The record buffer was larger than the maximum.
EPC$_INSARGS	Insufficient arguments specified.
EPC$_NODCFEXISTS	The specified data capture file does not exist.
EPC$_NOEVNT	No event ID argument specified.
EPC$_NOFAC	No facility argument specified.
EPC$_NOHANDL	No handle argument passed.
EPC$_NOTINSTALL	Oracle Trace software is not installed.
EPC$_SUCCESS	This call was successful.
EPC$_SYNC	The EPC$M_SYNC flag was set, and the call did not require an I/O.

EPC$START_EVENTW

EPC$START_EVENTW records the start of a duration event to the data collection file.

The EPC$START_EVENTW routine completes synchronously. It returns to the caller after all $START_EVENTW processing completes.

Format

EPC$START_EVENTW [efn] ,facility ,event-id ,handle [,context-variable] [,user-defined-buffer] [,status] [,astadr] [,astparam] [,flags]

Description

For asynchronous completion, use the EPC$START_EVENT service; EPC$START_EVENT returns control to the caller after initiating the processing, without waiting for I/O processing to complete.

For performance reasons, Oracle Trace does not write every record directly to the data collection file. Instead, records are buffered and flushed to disk when the buffer is full. The EPC$START_EVENTW routine completes synchronously; if a buffer flush needs to be performed, it waits for the I/O to complete before returning to the caller.

In all other respects, EPC$START_EVENTW is identical to EPC$START_EVENT. Refer to EPC$EVENT for all other information about the EPC$START_EVENTW routine.

8

System Management Tasks

This chapter describes the system management functions associated with using Oracle Trace.

8.1 Required Account and Process Quotas

You must make sure that the appropriate user accounts have sufficient quotas to be able to use Oracle Trace. If you typically format very large collection files (over 50,000 blocks) and generate reports based on large formatted databases, you can improve performance by increasing several of your account quotas. Table 8–1 summarizes the required and optional user account quotas.

Table 8–1 User Account Quotas for Using Oracle Trace

Account Quota	Normal Use	Formatting and Reporting on Large Files
ASTLM	24	
BIOLM	20	
BYTLM	20,480	34,810
DIOLM	20	
ENQLM	1,800	10,000
FILLM	50	
PGFLQUO	20,000	75,000
PRCLM	1	
WSEXTENT	2,048	
WSQUOTA	1,024	

User account quotas are stored in the file SYSUAF.DAT. Use the OpenVMS Authorize Utility to verify and change user account quotas. First set your directory to SYS$SYSTEM and then run AUTHORIZE:

```
$ SET DEFAULT SYS$SYSTEM
$ RUN AUTHORIZE
UAF>
```

At the UAF> prompt, use the SHOW command with an account name to check a particular account. For example:

```
UAF>SHOW SMITH
```

To change a quota, use the MODIFY command at the UAF> prompt. MODIFY has the following format:

MODIFY account-name /quota-name=NNN

The following example changes the FILLM quota for the SMITH account and then exits from the utility:

```
UAF>MODIFY SMITH /FILLM=50
UAF>EXIT
```

After you exit from the utility, the system displays messages indicating whether or not changes were made. Once you have made the changes, users must log out and log in again for the new quotas to take effect.

For more information on modifying account quotas, see the OpenVMS system management documentation.

8.2 Controlling Oracle Trace

A part of Oracle Trace called the **Registrar process** is designed to be active on your system or cluster at all times. This process handles all communication between your application programs and Oracle Trace. Without this process, applications instrumented with Oracle Trace calls cannot register with Oracle Trace. In addition, you cannot schedule or stop data collection without the Registrar process.

In a VMScluster environment, the Registrar process should be present on each node in the cluster. A log file exists in the EPC$HOME_DIR directory for the Registrar on each node in a VMScluster: EPC$REGISTRAR_node.LOG.

8.2.1 Starting Oracle Trace

Generally, you should start the Oracle Trace Registrar process as part of your normal system startup procedure. Note that you must start the Registrar process *after* activating the Oracle Rdb monitor process on your system. If you have multiversion Oracle Rdb installed, be sure to start Oracle Trace after the appropriate Oracle Rdb version.

Add the Oracle Trace startup procedure to the system startup command file on each node that you want Oracle Trace to run on:

```
$! Start the Oracle Rdb monitor process
$ @SYS$STARTUP:RMONSTART
$! Start Oracle Trace
$ @SYS$STARTUP:EPC$STARTUP
```

You can also restart the Registrar process interactively with the following command:

```
$ @SYS$STARTUP:EPC$STARTUP
```

In a multiversion Oracle Rdb environment, it is important for the Registrar process to use the appropriate version of Oracle Rdb to access the history, administration, and formatted databases. The EPC$STARTUP.COM procedure executes the file EPC$HOME_DIR:EPC$REGISTRAR_INIT.COM. This file in turn executes SYS$SHARE:RDBVMS_SETVER.COM to automatically set the Oracle Rdb version to the same version specified at installation time. If for some reason you manually convert your Oracle Trace databases, you must edit EPC$REGISTRAR_INIT.COM and set the appropriate version.

8.2.2 Stopping Oracle Trace

Oracle Corporation recommends that you stop Oracle Trace as part of your normal system shutdown procedure. Note that you must stop the Oracle Trace Registrar process *before* stopping the Oracle Rdb monitor process. If you attempt to stop the Oracle Rdb monitor first, both processes will hang and you will have to use the OpenVMS STOP/ID command to stop the EPC$REGISTRAR process.

The STOP SYSTEM command sends a message to the Registrar process. This process has to perform some rundown operations before exiting, and in a large VMScluster, you may need to add a short delay before shutting down the Oracle Rdb monitor. Add the Oracle Trace shutdown command to SYS$MANAGER:SYSHUTDWN.COM for each node that Oracle Trace is running on:

```
$! Shutdown Oracle Trace and abort all active data collection
$ COLLECT STOP SYSTEM/ABORT
$! Wait for Registrar communication to complete
$ WAIT 00:00:30
$! Shutdown the Oracle Rdb monitor process
$ @SYS$MANAGER:RMONSTOP
```

You can also stop the Registrar process interactively with the following command:

```
$ COLLECT STOP SYSTEM/ABORT
```

Note

Oracle Corporation recommends that you always use the Oracle Trace STOP SYSTEM command to stop collections. If you use the OpenVMS STOP/ID command instead, the Oracle Trace process stops; however, collections continue until they terminate naturally.

8.2.3 Recovering from a System Crash

After a system crash, the Oracle Trace registrar process should restart automatically as part of your normal system startup procedure.

If a collection is active when the system goes down, data collection resumes as long as the registrar process is restarted before the scheduled end-time of the collection. If a collection is pending when the system goes down, data collection begins as scheduled so long as the registrar process restarts before the scheduled end-time. Also, any images that had registered with Oracle Trace will have to be restarted. For example, consider the following collections scheduled on node MYVAX1:

```
24-JUL-1995 08:45          Scheduled Collections                  Page 1
Brief Report                                          Oracle Trace V2.2-0

Collection Schedule for node MYVAX1

    Selection Name   Collection Name  Start           End
    --------------   --------------  --------------  --------------
->  MY_SELECTION     MORNING_DATA    24-JUL-95 08:00  24-JUL-95 10:00
    TOOLS_SELECTION  JOB123          24-JUL-95 10:00  24-JUL-95 12:00
    MY_SELECTION     AFTERNOON_DATA  24-JUL-95 12:00  24-JUL-95 16:00
```

If MYVAX1 crashes at 09:00 and comes back at 09:30, the collection MORNING_DATA resumes data collection for the remaining 15 minutes of the scheduled collection interval. The pending collections: JOB123 and AFTERNOON_DATA are unaffected by the system crash and start as scheduled.

If MYVAX1 crashes at 09:00 and comes back at 10:15, the collection JOB123 starts and runs until its normal completion time. The pending collection: AFTERNOON_DATA is unaffected by the system crash. Note that the collection MORNING_DATA successfully collected data from 08:00 until the crash occurred at 09:00. If no process was flushing its data buffer into

the data capture file at the time of the crash, the data files produced by the collection can be formatted and reported on as if the collection had run to completion.

8.3 Installing Facilities on Your System

When you perform a product installation on your system, there are several cases where this could affect your use of Oracle Trace.

8.3.1 Installing New Facilities

When you install a product that has just added Oracle Trace support or when you install a supporting product for the first time, there is nothing that you need to do for Oracle Trace. The product's installation procedure usually inserts a new facility definition into the Oracle Trace administration database automatically. However, if the Oracle Rdb monitor is not running, the facility definition will be written to the Oracle Trace facility library SYS$SHARE:EPC$FACILITY.TLB. See Section 8.3.3 for information about extracting these definitions and inserting them into the administration database.

8.3.2 Installing New Versions of Existing Facilities

When you install a product whose previous version also supported Oracle Trace, you need to update any of your facility selections which specifically reference the previous version of the product. The product's installation procedure automatically inserts a new facility definition into the Oracle Trace administration database, but unless you change your facility selections to use the correct version number for the new product version, you will not be able to collect any data for that product.

Note that if you create your facility selections without specifying a version for the facilities, Oracle Trace always uses the latest version of the facility installed on your system. You will not have to change your facility selections when you perform an installation. However, you must recognize that any data collected from the new version may not be compatible with data collected before the installation. You would not be able to merge new data into an existing formatted database or file.

8.3.3 Installing a New Version of Oracle Rdb

Before installing a new version of Oracle Rdb, you *must* perform a full RMU[1] backup of the Oracle Trace administration and history databases (as well as any other Oracle Rdb databases on your system, including Oracle Trace-formatted databases produced with the FORMAT command).

To back up the Oracle Trace administration database, use the following command:

```
$ RMU/BACKUP EPC$ADMIN_DB EPC$ADMIN_DB.RBF
```

To back up the Oracle Trace history database, use the following command:

```
$ RMU/BACKUP EPC$HISTORY_DB EPC$HISTORY_DB.RBF
```

To back up an Oracle Trace formatted database, use the following command:

```
$ RMU/BACKUP OLD_FORMATTED_DATABASE OLD_FORMATTED_DATABASE.RBF
```

After installing the new version of Oracle Rdb you must restore your databases. The RMU/RESTORE operation restores the database to its original location.

To restore the Oracle Trace administration database, use the following command:

```
$ RMU/RESTORE/NEW_VERSION EPC$ADMIN_DB.RBF
```

To restore the Oracle Trace history database, use the following command:

```
$ RMU/RESTORE/NEW_VERSION EPC$HISTORY_DB.RBF
```

To restore an Oracle Trace formatted database, use the following command:

```
$ RMU/RESTORE/NEW_VERSION  OLD_FORMATTED_DATABASE.RBF
```

Because the Oracle Trace administration database is not available during the installation, Oracle Rdb inserts its facility definition into the Oracle Trace facility library SYS$SHARE:EPC$FACILITY.TLB. You must extract this definition and insert it into the Oracle Trace administration database. For example:

```
$ LIBRARY /TEXT /EXTRACT=RDBVMSV3.1-0 /OUT=RDBVMS.EPC$DEF -
_$ SYS$SHARE:EPC$FACILITY.TLB
$ COLLECT INSERT DEFINITION RDBVMS.EPC$DEF /REPLACE
```

Note

See the *Oracle Rdb Release Notes* for the version number of your Oracle

[1] RMU is the Oracle Rdb Management Utility

Rdb software.

After the installation, be sure to restart both the Oracle Rdb monitor and the Oracle Trace Registrar process.

If you do not perform the appropriate backups or perform the backups and fail to perform the restores, Oracle Trace displays the following error whenever anyone enters an Oracle Trace command.

```
RDB_F_WRONG_ODS the on-disk structure of database filename is not supported
  by version of facility being used
EPC_F_DBVER-Admin, history, or formatted database does not match current
  Rdb Version
```

Whenever you encounter this error, you must determine whether or not you performed the backup, or performed the backup and failed to perform the restore, which files were involved, and back up and/or restore the appropriate files accordingly. To do this perform the following tasks.

1. Narrow the problem down to the affected files, by entering the following commands and evaluating the responses.

Command	Displays the error message if . . .
SHOW DEFINITION	there is no updated version of the administration database
SHOW HISTORY	there is no updated version of the history database
REPORT formatted_ database.rdb	there is no updated version of the formatted database name that you specified

2. Display a directory of the suspicious file. The following command displays a directory of the history database file, to learn if you performed a backup and restore on the file.

   ```
   $ DIR/DATE EPC$DATABASE_DIR:EPC$HISTORY_DB.*
   ```

3. Examine the dates and file extensions to determine if you performed a backup and restore, or forgot to perform either or both operations.
4. Perform the necessary backups and restores.

8.4 Managing the History Database

Oracle Trace maintains a user-visible history database which contains a record of all informational and error messages encountered during data collection. There is no automatic purging function and over time, the history database can become very large. You should periodically create a new (empty) database and either offload or delete the old version.

Use the SET HISTORY /NEW_FILE command to create a new version of the history database. For example:

```
$ COLLECT SET HISTORY /NEW_FILE
$ PURGE/LOG EPC$HISTORY_DB
%PURGE-I-FILPURG, EPC$DATABASE_DIR:EPC$HISTORY_DB.RDB;1 deleted
(22640 blocks)
```

One method of regularly creating a new history database is to put the Oracle Trace SET HISTORY/NEW_FILE command into your site-specific startup procedure. Each time your system reboots, the old history database is closed and a new one is created. If you choose to do this, you should put the command before the execution of EPC$STARTUP so that no history data is lost. For example:

```
$ COLLECT SET HISTORY/NEW_FILE
$ PURGE/NOLOG/NOCONFIRM/KEEP=2 -
    SYS$SYSDEVICE:[EPC.DATABASES]EPC$HISTORY_DB.RDB
$ @SYS$STARTUP:EPC$STARTUP
```

The Oracle Trace Registrar process binds to the history database once when it is first created, and unbinds when you issue either the STOP SYSTEM or the SET HISTORY/NEW_FILE command. The modify date of the history database is updated only when the Registrar unbinds. Because of this, incremental backups of your disks may not copy the history database (or the Oracle Trace administration database).

One solution is to create a new history database prior to performing your incremental backups. Alternately, you could perform an Oracle Rdb online backup using the RMU/BACKUP/ONLINE command. Finally, note that changing the protection of a file updates the modify date, even if you set the protection to the existing protection scheme. You could set up a nightly command procedure to automatically reset the protection on the database files before you begin your incremental backup.

8.5 Disabling Data Collection

You can assign the logical, EPC$DISABLED, to turn off process registration within the scope of the logical name. To specify the logical name table (which determines the scope) where you want to enter a logical name, use the /PROCESS, /JOB, /GROUP, /SYSTEM, or /TABLE qualifier. For example, the following command disables all process registrations in your UIC group:

```
$ ASSIGN/GROUP EPC$DISABLED EPC$DISABLED
```

See the ASSIGN command in the OpenVMS DCL documentation for more information about creating logical names.

8.6 Using Oracle Trace Efficiently

Oracle Trace is designed to operate with minimal performance impact on your system. The following suggestions will ensure that you use Oracle Trace in the most efficient manner.

Facility developers can:

- Define several classes of events that are useful to users. Generally, you should have one class for each purpose. This limits the amount of data the user collects.
- Define separate development and production versions of a facility definition for debugging purposes.
- Use the optional Oracle Trace flags argument on the Oracle Trace service routine calls. For example, the EPC$M_NOEF bit provides for a more efficient execution because it eliminates the overhead of clearing and setting the event flag. The EPC$M_SYNC bit allows you to eliminate an AST routine call upon service completion.

  See Section 4.5 and the description of each service routine in Chapter 7 for more information about the flags argument.

General users can:

- Use a facility selection to specify only those facilities for which you want to collect data.
- Specify in a facility selection the collection class which contains the fewest number of events that you require.
- Limit the duration of collections, especially the first time you run it. Data collection files can get very large if you are collecting for a long collection interval (several hours) or have a high rate of events/second.

For the Debit/Credit workload for Oracle Rdb and Oracle CODASYL DBMS, the following collection rates apply:

- 20,000+ blocks per 1 TPS/Hour for ALL class data
- 20,000 blocks per 1 TPS/Hour for PERFORMANCE class data
- 15,000 blocks per 1 TPS/Hour for RDBEXPERT class data

- Use the /REGISTRATION_ID qualifier to the SCHEDULE COLLECTION command to limit images and processes collecting data to those with the characteristics that you specify. You can collect data on a per-process, per-image, or per-user basis. Note that the process ID (EPID), process name, image name, and user name are all registered automatically when an image instrumented with Oracle Trace activates. Use the SHOW REGISTER command to display process characteristics.

  Note that values that you provide to the REGISTRATION_ID qualifier must be exactly as they appear in the SHOW REGISTER display for the processes to which you want to limit collections. You cannot use abbreviations or default values.

- Use the /FILELIST qualifier to the SCHEDULE COLLECTION command to distribute data collection files across several disks. This reduces potential I/O bottlenecks. This is especially useful if you have several images collecting data or if you are collecting for a long time interval.

- For better user interface performance, enter the Oracle Trace command environment by entering the COLLECT command with no arguments. This eliminates binding to the history and administration databases for each command.

- Improve the performance of report generation by using SQL or interactive RDO to add indices to the database relations in your formatted database. For information about creating indices and about the layout of the formatted database, see the sections on using Oracle Trace in the documentation sets of the individual layered products. Also, appendixes of the *Oracle Trace Reporter User's Guide* describe the formats of both the Oracle Rdb formatted database and the VAX RMS file.

- Improve the formatting of very large data collection files by changing the logical name RDMS$BIND_WORK_FILE. This logical specifies temporary tables that Oracle Rdb uses on a disk structure other than the disk that contains the database. This reduces the disk I/O operations on the disk where the database resides. The default location for the files is SYS$LOGIN. For example:

```
$ ! Assign the work area to another disk with read-write access
$ DEFINE RDMS$BIND_WORK_FILE WORK$DISK:[RDB.WORK]
```

System managers can:

- Adjust system parameters to enhance performance or lower the use of some system resources. One recommendation is to increase process working set parameters to speed up Oracle Trace formatting operations and report generations.
- Increase users' BYTLM quota and increase the logical EPC$SRG_MAX_NETWORK_LINK. The maximum number of network links that the SHOW REGISTER command will use can be set using this logical. For example:

  ```
  $ DEFINE EPC$SRG_MAX_NETWORK_LINK 20
  ```

  The current default is 15, set according to the minimum recommended BYTLM quota of 20,480. It should be set lower if the following error is encountered when using the SHOW REGISTER command:

  ```
  %SYSTEM-F-EXBYTLM, exceeded byte count quota
  ```

- Set the EPC$BE_REGTIMEOUT and EPC$FE_REGTIMEOUT logicals to increase or decrease the time a process waits when communicating with the Registrar process.

  EPC$BE_REGTIMEOUT defines the time a process waits for the Registrar to respond to an EPC$INIT call. EPC$FE_REGTIMEOUT defines the time a user waits for the Registrar to respond to a SCHEDULE COLLECTION, CANCEL COLLECTION, or STOP SYSTEM command before the command fails. The timers can be set to any whole number of seconds from 1 to $2^{32} - 1$. The default for each timer is 90 seconds.

9
Troubleshooting Oracle Trace

This chapter describes some typical problems that you might encounter when using Oracle Trace on your system. Table 9–1 shows the symptoms, causes, and cures of many common problems. Some of the recommendations for verifying errors assume that you have appropriate privileges.

Note that the primary troubleshooting tool is the SHOW HISTORY command which displays error or warning messages that occur during data collection. See Section 3.5.2 for a description of the SHOW HISTORY command.

Table 9–1 Troubleshooting Oracle Trace Problems

Symptom	Possible Cause	Solution
Data collection is active but no data is collected.	No applications contained facilities that were specified in the facility selection.	Use SHOW REGISTER to show all available applications.
	When you created the facility selection, either you did not explicitly specify a facility version number or you specified an incorrect one. The CREATE SELECTION command defaults to the latest facility version. You can verify this by comparing the facility version number displayed by the SHOW SELECTION command to the version displayed by the SHOW REGISTER command.	Create another facility selection and specify an appropriate version number. See Section 2.3 for more information.

(continued on next page)

Table 9–1 (Cont.) Troubleshooting Oracle Trace Problems

Symptom	Possible Cause	Solution
	Oracle Trace could not find a match between characteristics you supplied with the /REGISTRATION_ID qualifier and characteristics associated with processes.	Compare the characteristics you supplied to the /REGISTRATION_ID qualifier with the characteristics displayed by the SHOW REGISTER command for the processes you want to collect for. See Section 3.1 for more information.
	The Oracle Trace Registrar process has stopped.	Restart the registrar process. See Section 8.2.1.
	Processes have insufficient privilege to write to the data collection file.	Use the PROTECTION parameter to the /COLLECTION_FILES qualifier to the SCHEDULE COLLECTION command.
	The process attempted to register before the Registrar was available.	Check the SHOW REGISTER display to ensure that the process has registered. If necessary, exit from the facility image and restart it.
Output disks fill up too quickly.	You are collecting too much data.	Schedule shorter collection intervals, use the registration ID, and use a collection class to limit the amount of data collected. See Sections 2.1 and 3.1.
	Insufficient disk space to begin with.	Spread data files over several disks using the /FILELIST qualifier to the SCHEDULE COLLECTION command. Also use /COLLECTION_FILES to set the maximum allowable size for the collection files.
You cannot delete facility definitions or facility selections.	You are not the creator of the specified definition, selection or collection.	You must have OpenVMS BYPASS OR SYSPRV privilege to delete other users' definitions or selections.
	You do not have access to the administration or history databases.	See the SET ACCESS command in Chapter 6.
	You attempted to delete a facility definition that is referenced by a facility selection.	You must delete any facility selections that reference the facility definition.
	You attempted to delete a selection that is referenced by an active or pending collection.	You must cancel any active or pending collections that reference the facility selection.

(continued on next page)

Table 9–1 (Cont.) Troubleshooting Oracle Trace Problems

Symptom	Possible Cause	Solution
You cannot schedule, show, cancel a collection, or delete a facility.	You do not have write access to the administration database or read or write access to the History database.	See the Set ACCESS command in Chapter 6.
Collections only run on the local node (in a cluster environment).	Oracle Trace is not running on the rest of the nodes in the cluster.	You must start Oracle Trace on every node in the cluster. See Section 8.2.1.
	The EPC$ADMIN_DB and EPC$HISTORY_DB logicals do not reference the same databases on all nodes in the cluster. You can verify this by using the SHOW LOGICAL command on each node and comparing them.	Reinstall Oracle Trace on all nodes that have the wrong logical definitions. When prompted for EPC$ROOT, specify a directory that is common across all nodes in the cluster.
Elapsed time information for duration events is negative.	The system time was changed during the execution of the event.	Reschedule data collection and do not change system time during active collections.
	The start event and end event service routines are switched in your facility. You can verify this by reviewing your code.	Move the EPC$START_EVENT and EPC$END_EVENT service routine calls to their proper location and recompile your program.
Start or End events are missing.	Collection interval started or ended in the middle of a duration event.	Activate images after starting collections, or ignore mismatched events.
	Event may have occurred too soon after the call to EPC$INIT.	See Section 4.8.2 for information on guaranteeing data collection.
You cannot stop the Registrar process (the STOP SYSTEM command fails).	The Oracle Rdb monitor process is not running. You can verify this by using the RMU/SHOW SYSTEM command.	Use the OpenVMS STOP/ID command to stop the EPC$REGISTRAR process. Note that active data collection continues. See Section 8.2.2.

(continued on next page)

Table 9–1 (Cont.) Troubleshooting Oracle Trace Problems

Symptom	Possible Cause	Solution
Collections are stuck in the “aborting” state.	The Registrar is not running on all nodes in the VMScluster. You can verify this by using the SHOW SYSTEM command for each node in the cluster. You can also use the SHOW COLLECTION /FORMAT=FULL command to display a list of nodes that have not stopped the collection.	Restart the Registrar process on those nodes that appear in the output of the SHOW COLLECTION/FORMAT=FULL display. See Section 8.2.1.
	Collection files are full on some nodes.	Wait for the collection interval to end. Alternatively, you can reschedule the collection and then cancel it.
Processes unregister or never register.	The process matches the criteria for more than five concurrent collections.	Make sure the process is registered before starting the sixth collection. See Section 3.2.
	The call to EPC$INIT times out before the Registrar has responded.	Set the EPC$BE_REGTIMEOUT logical. See Section 8.6.
All Oracle Trace commands produce an "-RDB-F-WRONG_ODS, the on-disk structure of database filename is not supported by version of facility being used" error message.		
The secondary message is "EPC_C_DBVER, Admin, history, formatted database does not match current Rdb Version"	You have installed a new version of Oracle Rdb without backing up and restoring Oracle Trace databases.	See Section 8.3.3 for details.
The secondary message is "RDMS-F-ROOTMAJVER, database format nn is not compatible with software version xx".	Your are running multiple versions of Oracle Rdb and your default Oracle Rdb version is different from the version under which the Oracle Trace administration and history databases were created.	Enter this command: `@SYS$SHARE:RDBVMS_SETVER nn`. Note that nn corresponds to the Oracle Rdb version under which your Oracle Trace versions were collected.

9.1 Error Messages and Recovery Procedures

The file SYS$HELP:EPC$MSG.DOC contains a listing of a subset of the Oracle Trace error messages, with explanations and user actions. The messages are arranged in alphabetical order by error code. You can use a text editor or the DCL SEARCH command to locate specific error codes or text strings.

Within the Oracle Trace command environment, you can use the HELP command to display the explanation and user action associated with an error message. For example, the following command displays information about the error code FACCRE_NOEVTOPTS:

```
Trace> HELP ERROR FACCRE_NOEVTOPTS

Error_Messages_and_Recovery

  FACCRE_NOEVTOPTS

    Must specify list of events or event options

       Explanation: User attempted to create a facility definition
       which did not contain any events.

       User Action: You must specify at least one event when creating
       a facility definition.
```

Note that an error encountered by a process which has registered with Oracle Trace is not related to any collection for which the process might be recording event data. If you use the COLLECTION-NAME parameter to the SHOW HISTORY command, you will not see errors or messages generated by the individual processes. For example, if a process is unable to record data due to a file protection violation on the data collection file, the message will be on the SHOW HISTORY /FORMAT=ALL report, not the report for a specific collection.

A

Example of the ATM-Sample Facility Instrumented

This appendix contains an example of a facility written in VAX C that is instrumented for Oracle Trace. See Example A–1.

Example A–1 EPC$ATM-SAMPLE-EXTENDED Facility

```
/*AUTHOR. Oracle Corporation
*
* PROGRAM ABSTRACT:
*
*       This is a sample application that illustrates the features of
*       Oracle Trace.
*
*       The application mimics an automatic teller machine (ATM),
*       prompting for an account number, then allowing the user to
*       select one of three tasks:
*
*            -  Withdrawing funds
*            -  Depositing funds
*            -  Displaying the current balance
*
*   The program logs events for the above tasks as well as a signin event
*   which is the duration from the time the user specifies the account until
*   exitting.  The account identifier is logged as cross-facility 14 in both
*   the atm facility (balance, deposit and withdrawal events) and all of the
*   Rdb events.  After collecting data for both ATM and Rdb it is possible to
*   see the resources used by each ATM event as well as what portion of those
*   resources were used by Rdb.  This could help determine whether a change
*   in the physical database needs to be made, the SQL queries need to be
*   changed or if the program flow needs to be modified.
*
*   For example, all of the events utilize a RW transaction.  It might make
*   more sense for all to first use a Read Only transaction while getting the
*   data from the database and then utilize a RW when updating programatically
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
*   ensure atomicity.  This would depend on how often database contention is
*   a factor and can best be easily determined by analyzing the logged data
*   over a period of peak utilization.
*
*  REQUIRED FILES:
*
*       There are no other files necessary to compile and link this
*       application.  However, the database epc$atm_sample is required.
* The file epc$atm-sample-db.sql is used to create the sample db.
* The location of the file is not important as long as a logical
* EPC$ATM_SAMPLE is defined to point to the file.
*
*
*
* MODIFICATION HISTORY:
*
*       Created:  May-1989
*      Modified:  Jan 1991        -- translated to C from Pascal, not much
*         error checking.
*    Mar 1991   -- Added a User definable item buffer with
*         a representative sample of item types.
*    Jun 1992   -- Use Rdb to show cross-facility
*******************************************************************************/

#include <stdio.h>
#include <ssdef.h>
#include <epc$definitions.h>
#include <descrip.h>

#define     TRUE 1
#define     FALSE 0
#define     RETRY_TIMES 5   /* # of times flag checked after registering    */

/* If you want to run this program using the debugger, then uncomment this  */
/* set of defines and comment the ones below.  Otherwise, your screen will  */
/* get cleared underneath you making it hard to follow the code.     */
/*
#define   CLEAR_SCREEN  printf("")
#define   NORM          printf("")
#define   BOLD          printf("")
#define   UNDERSCORE    printf("")
#define   BLINK         printf("")
#define   REVERSE       printf("")
*/
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
#define   CLEAR_SCREEN  printf("%c[2J%c[2H",27,27)
#define   NORM          printf("%c[0m",27)
#define   BOLD          printf("%c[1m",27)
#define   UNDERSCORE    printf("%c[4m",27)
#define   BLINK         printf("%c[5m",27)
#define   REVERSE       printf("%c[7m",27)

/*
*  The following constants are used for the Oracle Trace service routines.
*                                                                          */
#define FACILITY_NUMBER  4094
#define MIN_EVENT  1
#define ERROR_DISPLAY  1
#define BALANCE_EVENT  2
#define DEPOSIT_EVENT  3
#define WITHDRAW_EVENT  4
#define SIGNIN_EVENT  5
#define MAX_EVENT  5
#define MIN_ITEM  1
#define MAX_ITEM  128

#define ACCOUNT_ID_LENGTH 9
#define CUST_NAME_LENGTH 25
#define THIS_MACHINE  1
#define MACHINE_LOC_LENGTH 40


/* Number of Bytes used to hold size for length of user defined items     */
#define FIXED_ASCIC_LENGTH_BYTES  1
#define ASCIC_LENGTH_BYTES        1
#define ASCIW_LENGTH_BYTES        2

#define WORD_LENGTH_BYTES         2
#define LONGWORD_LENGTH_BYTES     4

#define SIGSQL(sqlcode)         \
    if (sqlcode != 0)         \
      {           \
        int temp_sqlcode;        \
        epc$$atm_rollback(temp_sqlcode)              \
        printf("\nError accessing db. exitting..."); \
 exit(1);         \
      }
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
extern struct
  {
        int rdb$lu_num_arguments;
 int rdb$lu_status;
 int rdb$lu_arguments[18];
  } rdb$message_vector;

/*
* The following constants are used as headings
*/
char  yourbank_heading[]   = {"YOURBANK -- Automatic Teller Machine"};
char  withdrawal_heading[] = {"Withdrawal Form"};
char  deposit_heading[]    = {"Deposit Form"};
char  balance_heading[]    = {"Customer Balance Form"};

/*
 * The following constants are used as informational and warning messages
*/
char error_message[]=
 {"                                                          "};
char invalid_selection[]  = {" Unrecognized selection. Please try again..."};
char no_such_account[]    =
 {" Unrecognized account number. Please try again.."};
char overdraft[]          =
 {" Withdrawal exceeds current balance. Please contact the bank."};
char transaction_failed[] =
 {" An error has occurred. The transaction cannot be completed..."};
char thank_you[]          = {" Thank you. Please come again..."};
char acknowledge_message[]= {" (Press RETURN to continue.)"};

char signin_instruction_1[]  ={" Please enter your account number."};
char signin_instruction_2[]  ={" (Enter 0 to exit the application.)"};
char main_instruction_1[]    ={" Please enter the number for your selection."};
char withdrawal_instruction[]={" Please enter the amount of the withdrawal."};
char deposit_instruction[]   ={" Please enter the amount of the deposit."};
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
typedef  struct     signin_id_s
    {
 char     size;
 char        string[ACCOUNT_ID_LENGTH];
    } SIGNIN_ID_T;

typedef  struct     record_buffer_s
    {
 SIGNIN_ID_T    signin_id;      /* fixed ascic     */
 short        machine_number;      /* word       */
 char        machine_location
        [MACHINE_LOC_LENGTH+ASCIC_LENGTH_BYTES];    /* ascic */
    } RECORD_BUFFER_T;

typedef struct item_flags_s
  {
    int account_id  : 1;    /* first bit = account_id       */
    int machine_no  : 1;    /* second = machine number       */
    int machine_loc : 1;    /* third = machine location       */
    int restofmine1 : 29;   /* rest of available item flags for this longword*/
    int restofmine2 : 32;   /* rest of available item flags for this longword*/
    int restofmine3 : 32;   /* rest of available item flags for this longword*/
    int restofmine4 : 4;    /* I can use first 4 bits of last longword      */
    int Oracle Traces   : 28;   /* Oracle Trace resource items and reserved flags   */
  } ITEM_FLAGS_T;

/* Note: I am using the max events and item flags.  The item flags needs to */
/* be max because we are collecting resource items, to use VM more     */
/* efficiently the event_flags array should only be MAX-MIN+1 long     */
static int s_event_flags[128];
static ITEM_FLAGS_T s_event_item_flags[128];

static $DESCRIPTOR(s_fac_ver,"V1.2-0");
static $DESCRIPTOR(s_reg_id,"ATM APPLICATION EXT");

/* The following variables are used to store the user's selection and
   to calculate the result of the transaction.
*/
  int     account_num;
  int     user_choice;
  char     acknowledged;

  char     account_id[ACCOUNT_ID_LENGTH];
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
/*****************************************************************************/
close_and_exit_program()
 {
   int     sqlcode;

   CLEAR_SCREEN;
   NORM;
   printf("                      ");
   UNDERSCORE;
   printf("\n\n\n%s",yourbank_heading);
   NORM;
   printf("\n\n\n");
   printf("                       ");
   BLINK;
   printf("Have a nice day");
   NORM;
   printf("\n\n\n");

   epc$$atm_finish(&sqlcode);
   if ((sqlcode) != 0)
     {
       printf("\nError disconnecting for bank database file");
     }

 };
/*****************************************************************************/
withdrawal(int account_id)
  {
    float  transaction_amount;
    char  temp_char;
    long  cond_status;
    float   account_bal;
    int   sqlcode;

    CLEAR_SCREEN;
    printf("\n                   ");
    UNDERSCORE;
    printf("\n\n\n%s",yourbank_heading);
    NORM;
    printf("\n                        ");
    REVERSE;
    printf("",withdrawal_heading);
    NORM;
    printf("\n\n\n%s",withdrawal_instruction);
    printf("\n        Amount of withdrawal: ");
    scanf("%f",&transaction_amount);
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
    /* Start a RW transaction     */
    start_db_trans();
    epc$$atm_get_account_info(&account_id, &account_bal, &sqlcode);
    SIGSQL(sqlcode);

    if ((account_bal - transaction_amount) >= 0 )
     {
      account_bal = account_bal - transaction_amount;
      epc$$atm_upd_account_info(&account_id, &account_bal, &sqlcode);
      SIGSQL(sqlcode);
      epc$$atm_commit(&sqlcode);
      SIGSQL(sqlcode);
     }
    else
     {
     printf("\n\n\n%s",overdraft);
      if (s_event_flags[ERROR_DISPLAY - 1] != FALSE)
        {
          cond_status = epc$event(0,      /* No efn used     */
      FACILITY_NUMBER,  /* facility number REQ. */
      ERROR_DISPLAY,    /* Event ID REQUIRED    */
      0,      /* Context not used     */
      0,      /* No user defined items*/
      0,      /* No status needed     */
      0,      /* No AST       */
      0,      /* No AST parameters    */
      EPC$M_NOEF     /* Don't clear/set efn  */
      );
        };
     }
    while ((temp_char = getchar()) != '\n');
    printf("\n\n\n%s",acknowledge_message);
    scanf("%c",&temp_char);
  }
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
/*****************************************************************************/
deposit(int account_id)
  {
    float  transaction_amount;
    char  temp_char;
    float  account_bal;
    int   sqlcode;

    CLEAR_SCREEN;
    printf("\n                   ");
    UNDERSCORE;
    printf("\n\n\n%s",yourbank_heading);
    NORM;
    printf("\n                         ");
    REVERSE;
    printf("%s",deposit_heading);
    NORM;
    printf("\n\n\n%s",deposit_instruction);
    printf("\n        Amount of deposit: ");
    scanf("%f",&transaction_amount);

    while ((temp_char = getchar()) != '\n');
    printf("\n\n\n%s",acknowledge_message);
    scanf("%c",&temp_char);

    /* Start a RW transaction     */
    start_db_trans();
    epc$$atm_get_account_info(&account_id, &account_bal, &sqlcode);
    SIGSQL(sqlcode);

    account_bal = account_bal + transaction_amount;
    epc$$atm_upd_account_info(&account_id, &account_bal, &sqlcode);
    SIGSQL(sqlcode);
    epc$$atm_commit(&sqlcode);
    SIGSQL(sqlcode);
  }
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
/*****************************************************************************/
balance(int account_id)
  {
    char  temp_char;
    float  account_bal;
    int   sqlcode;

    CLEAR_SCREEN;
    printf("\n                   ");
    UNDERSCORE;
    printf("\n\n\n%s",yourbank_heading);
    NORM;
    printf("\n                         ");
    REVERSE;
    printf("%s",balance_heading);
    NORM;

    printf("\n\n\n              Your balance is:");

    /* Start a RW transaction     */
    start_db_trans();
    epc$$atm_get_account_info(&account_id, &account_bal, &sqlcode);
    SIGSQL(sqlcode);
    epc$$atm_commit(&sqlcode);
    SIGSQL(sqlcode);

    printf("%.2f",account_bal);

    while ((temp_char = getchar()) != '\n');
    printf("\n\n\n%s",acknowledge_message);
    scanf("%c",&temp_char);

  }
/*****************************************************************************
the routine to bind to the atm database and start a read write transaction
*****************************************************************************/
start_db_trans()
 {

   int     sqlcode;

   epc$$atm_trans_rw(&sqlcode);
   if (sqlcode != 0)
     {
       printf("\n Error attaching to atm database.  bye...");
       exit(1);
     }

 }
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
/*****************************************************************************/
main()
 {
  char      temp_char;
  float      delay_time = 2.0; /* delay time for registering     */
  long      si_event_handle;
  long      event_handle;
  long      status;
  long      cond_status;
  short      i;
  int      new_account = TRUE;
  struct     dsc$descriptor_s record_desc;
  char      atm_location[10];
  char      bank_name[40];
  char      bank_location[40];
  char      *record_buffer;
  int      account_id_number;
  int      *cf_14_loc;

  strcpy(atm_location,"Nashua, NH");
  strcpy(bank_name,"My Favorite Bank");
  strcpy(bank_location,"Nashua, NH");

/* Call EPC$INIT to register the ATM_SAMPLE facility with Oracle Trace */
  cond_status = epc$init(0,      /* No VMS efn used      */
    FACILITY_NUMBER,   /* facility number REQUIRED     */
    &s_fac_ver,     /* Facility version REQUIRED    */
                         &s_reg_id,     /* registration ID      */
                         &s_event_flags,    /* Oracle Trace event flags     */
    &s_event_item_flags,/* Item flags buffer      */
    0,      /* No Status block needed     */
    0,      /* No AST       */
    0,      /* No AST parameter      */
    EPC$M_NOEF,     /* Don't bother to clear and    */
         /* set efn, I'm not using it    */
    MIN_EVENT,     /* Min. event for this faciity  */
    MAX_EVENT,     /* Max. event for this facility */
    MIN_ITEM,     /* Min. item for this facility  */
    MAX_ITEM     /* Max. item for this facility  */
    );

/* To guarantee that no events get missed, we could put a loop here to wait */
/* for the EPC$REGISTRAR to activate this process.  Make sure the event     */
/* is always collected.           */
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
   for(i=0;i<RETRY_TIMES && s_event_flags[ERROR_DISPLAY] != 1;i++)
     {
       status = lib$wait(&delay_time);
       if (status != SS$_NORMAL)
         {
    printf("\nError returned from lib$wait...");
    exit(1);
  }
     }
  /* Get account number and then process client's requests      */
  while (new_account == TRUE)
   {
      new_account = FALSE;
      CLEAR_SCREEN;
      printf("                    ");
      UNDERSCORE;
      printf("\n\n\n%s",yourbank_heading);
      NORM;
      printf("\n\n\n%s\n%s",signin_instruction_1,signin_instruction_2);
      printf("\n\n                  Account number:    ");
      BOLD;

      strcpy(account_id,"         ");
      scanf("%9s",account_id);

      /* Convert the string read in to an integer and then       */
      /* log it as CF item 14.          */
      account_id_number = atoi(account_id);


      if ((strcmp(account_id,"0") != 0) &&
   (s_event_flags[SIGNIN_EVENT - 1] != FALSE) )
       {
         /*****************************************************************/
         /* The design of this sample can assume that signin_event will    */
         /* always be logged if any ATM Oracle Trace collection is active.  To */
         /* guarantee this, the signin_event must be part of ALL classes of*/
         /* events in the ATM facility definition.  If we are assured of   */
   /* this then only set a cf_item if a collection is active.     */
         /*****************************************************************/
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
   /* By setting the CF item, it can be logged by Rdb as well    */
   cond_status = epc$set_cf_value(
      0,      /* No efn used      */
      CROSS_FAC_14,    /* define CF 14      */
      SIGNIN_EVENT,    /* set its value to the act id*/
      0,      /* Not on a thread      */
      &cf_14_loc,     /* the location of the cf item */
      EPC$M_NOEF);     /* Oracle Trace don't clear/set efn */
   /*****************************************************************/
   /* Sample lay out of user defined buffer       */
   /* The user defined items are:        */
   /* signin_id:     FIXED_ASCIC 9 characters      */
   /* machine_number:     longword        */
   /* machine_lacation:   ASCIC        */
   /* Buffer looks like:          */
   /*  +--------------------------+        */
   /*  |a|    b    | c  |d|     e...        */
   /*  +--------------------------+        */
   /*  where:           */
   /*    a = fixed_ascic length (1 byte)        */
   /* b = actual fixed_ascic data (9 bytes)       */
   /* c = longword (4 bytes)         */
   /* d = ascic length (1 byte)        */
   /* e = actual ascic data (c bytes)        */
   /*****************************************************************/

   record_buffer = (char *) malloc(
   FIXED_ASCIC_LENGTH_BYTES + ACCOUNT_ID_LENGTH +
          LONGWORD_LENGTH_BYTES +
   ASCIC_LENGTH_BYTES + sizeof(atm_location));
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
        /* replace null with a space for index       */
        account_id[strlen(account_id)] = ' ';
  record_buffer[0] = ACCOUNT_ID_LENGTH;
  memcpy(&(record_buffer[FIXED_ASCIC_LENGTH_BYTES]),
 account_id, ACCOUNT_ID_LENGTH);
  record_buffer[FIXED_ASCIC_LENGTH_BYTES+ACCOUNT_ID_LENGTH] =
    (long) THIS_MACHINE;
  record_buffer[FIXED_ASCIC_LENGTH_BYTES+
                      ACCOUNT_ID_LENGTH+
                      LONGWORD_LENGTH_BYTES] =
    strlen(atm_location);
  memcpy(&(record_buffer[FIXED_ASCIC_LENGTH_BYTES+
                             ACCOUNT_ID_LENGTH+
                             LONGWORD_LENGTH_BYTES+
                             ASCIC_LENGTH_BYTES]),
   atm_location, strlen(atm_location));
        /* length = signin id size (1byte) + signin id + machine no. +    */
  /*          machine loc size (1 byte) + machine loc str size     */
        record_desc.dsc$w_length = FIXED_ASCIC_LENGTH_BYTES+
                                   ACCOUNT_ID_LENGTH+
       LONGWORD_LENGTH_BYTES+
                                   ASCIC_LENGTH_BYTES+
                                   sizeof(atm_location);
  record_desc.dsc$b_class = DSC$K_CLASS_S;
  record_desc.dsc$a_pointer = record_buffer;
        cond_status = epc$start_event(0,  /* No efn used      */
    FACILITY_NUMBER,  /* Facility number REQUIRED     */
    SIGNIN_EVENT,     /* Event ID REQUIRED     */
    &si_event_handle, /* Event occurance handle REQ   */
    0,      /* No Context, single threaded  */
    &record_desc,     /* Data buffer      */
    0,      /* No status required     */
    0,      /* No AST       */
    0,      /* No AST parameter      */
    EPC$M_NOEF     /* Don't clear/set efn     */
    );
      }
     NORM;
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
      while ((strcmp(account_id,"0") != 0) & (new_account == FALSE))
       {
    /* In this example its a small bank with only 9 accounts.     */
           if (account_id_number > 0 && account_id_number < 10)
            {
               CLEAR_SCREEN;
               printf("\n                 ");
        UNDERSCORE;
               printf("\n\n\n%s",yourbank_heading);
        NORM;
        printf("\n\n\n                       ");
        BOLD;
        printf("1.");
        NORM;
        printf("  Withdrawal");
        printf("\n\n                      ");
        BOLD;
        printf("2.");
        NORM;
        printf("  Deposit");
               printf("\n\n                           ");
        BOLD;
               printf("3.");
        NORM;
        printf("  Balance");
        printf("\n\n\n                       ");
        BOLD;
        printf("4.");
        NORM;
        printf("  Exit");

        printf("\n\n\n    Choice: ");
        BOLD;
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
  scanf("%d",&user_choice);
        NORM;
 switch(user_choice)
  {
   case 1:
     /*   withdrawal; first check if event should be collected  */
           if (s_event_flags[WITHDRAW_EVENT - 1] != FALSE)
{
  /* change the cf_14 value to be logged by Oracle Trace   */
  /* to signify the Rdb events occurred during an ATM  */
  /* WITHDRAWAL event.  This is cheaper than calling   */
  /* EPC$SET_CF_VALUE.        */
  *cf_14_loc = WITHDRAW_EVENT;

               epc$start_event(0,  /* No efn used     */
    FACILITY_NUMBER, /* facility number  */
    WITHDRAW_EVENT, /* Event ID     */
    &event_handle, /* Event handle     */
    0,  /* No Context, single*/
       /* threaded          */
    0,  /* No user Data buffer*/
    0,  /* No status required*/
    0,  /* No AST     */
    0,  /* No AST parameter */
    EPC$M_NOEF /* Don't clear/set efn*/
    );
             };
           withdrawal(account_id_number);
           if (s_event_flags[WITHDRAW_EVENT - 1] != FALSE)
{
               epc$end_event(0,   /* No efn used     */
  FACILITY_NUMBER, /* Facility Number  */
  WITHDRAW_EVENT, /* Event ID     */
  &event_handle, /* Distiguish this  */
/* occurrence with  */
/* the above start  */
  0,   /* No Context, single*/
                                                 /* threaded          */
  0,   /* No user Data buffer*/
  0,   /* No status required*/
  0,   /* No AST     */
  0,   /* No AST parameter */
  EPC$M_NOEF  /* Don't clear/set efn*/
  );
       };
            break;
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
    case 2:
/*  deposit event; 1st check if event should be collected */
            if (s_event_flags[DEPOSIT_EVENT - 1] !=FALSE)
 {
  /* change the cf_14 value to be logged by Oracle Trace   */
  /* to signify the Rdb events occurred during an ATM  */
  /* DEPOSIT event.  This is cheaper than calling     */
  /* EPC$SET_CF_VALUE.        */
  *cf_14_loc = DEPOSIT_EVENT;
               epc$start_event(0,  /* No efn used     */
  FACILITY_NUMBER, /* Facility Number  */
  DEPOSIT_EVENT, /* Event ID     */
  &event_handle, /* Distiguish this  */
 /* occurrence with  */
 /* the above start  */
  0,   /* No Context, single*/
 /* threaded       */
  0,   /* No user Data buffer*/
  0,   /* No status required*/
  0,   /* No AST     */
  0,   /* No AST parameter */
  EPC$M_NOEF  /* Don't clear/set efn*/
  );
             };
            deposit(account_id_number);
            if (s_event_flags[DEPOSIT_EVENT - 1] != FALSE)
 {
               epc$end_event(0,   /* No efn used      */
  FACILITY_NUMBER, /* Facility Number   */
  DEPOSIT_EVENT, /* Event ID      */
  &event_handle, /* Distiguish this   */
 /* occurrence with   */
 /* the above start   */
  0,   /* No Context, single*/
 /* threaded          */
  0,   /* No user Data buffer*/
  0,   /* No status required*/
  0,   /* No AST      */
  0,   /* No AST parameter  */
  EPC$M_NOEF  /* Don't clear/set efn*/
  );
             };
     break;
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
  case 3:
     /* balance event; check event first */
           if (s_event_flags[BALANCE_EVENT - 1] != FALSE)
{
  /* change the cf_14 value to be logged by Oracle Trace   */
  /* to signify the Rdb events occurred during an ATM  */
  /* SIGNIN event.  This is cheaper than calling     */
  /* EPC$SET_CF_VALUE.        */
  *cf_14_loc = BALANCE_EVENT;

               epc$start_event(0,  /* No efn used     */
  FACILITY_NUMBER, /* Facility Number  */
  BALANCE_EVENT, /* Event ID     */
  &event_handle, /* Distiguish this  */
/* occurrence with  */
/* the above start  */
  0,   /* No Context, single */
/* threaded       */
  0,   /* No user Data buffer*/
  0,   /* No status required*/
  0,   /* No AST     */
  0,   /* No AST parameter */
  EPC$M_NOEF  /* Don't clear/set efn*/
  );
             };
           balance(account_id_number);
           if (s_event_flags[BALANCE_EVENT - 1] != FALSE)
{
               epc$end_event(0,   /* No efn used     */
  FACILITY_NUMBER, /* Facility Number  */
  BALANCE_EVENT, /* Event ID     */
  &event_handle, /* Distiguish this  */
/* occurrence with  */
/* the above start  */
  0,   /* No Context, single*/
/* threaded        */
  0,   /* No user Data buffer*/
  0,   /* No status required*/
  0,   /* No AST     */
  0,   /* No AST parameter */
  EPC$M_NOEF  /* Don't clear/set efn*/
  );
             };
     break;
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
        case 4:
                if (s_event_flags[SIGNIN_EVENT - 1] != FALSE)
    {
       /* change the cf_14 value to be logged by Oracle Trace   */
       /* to signify the Rdb events occurred during an ATM  */
       /* BALANCE event.  This is cheaper than calling     */
       /* EPC$SET_CF_VALUE.        */
       *cf_14_loc = SIGNIN_EVENT;
      /*****************************************************/
      /* Sample lay out of user defined buffer      */
      /* The user defined items are:       */
      /* signin_id:     FIXED_ASCIC 9 characters*/
      /* machine_number:     longword      */
      /* machine_lacation:   ASCIC      */
      /* Buffer looks like:        */
      /*  +--------------------------+      */
      /*  |a |  b...   |c |   d...     |      */
      /*  +--------------------------+      */
      /*  where:         */
      /*    a = Asciw Length (2 bytes)       */
      /*    b = Asciw Data (upto 40 bytes)      */
      /*    c = Asciw length (2 bytes)       */
      /*    d = Asciw data (upto 40  bytes)      */
      /*****************************************************/
             record_buffer = (char *) malloc(
ASCIW_LENGTH_BYTES +
sizeof(bank_name)  +
ASCIW_LENGTH_BYTES +
sizeof(bank_location));
     record_buffer[0] = sizeof(bank_name);
     record_buffer[1] = 0;
     memcpy(&(record_buffer[ASCIW_LENGTH_BYTES]),
    bank_name, sizeof(bank_name));
     record_buffer[ASCIW_LENGTH_BYTES+sizeof(bank_name)] =
sizeof(bank_location);
     record_buffer[ASCIW_LENGTH_BYTES+sizeof(bank_name)+1] =
0;
     memcpy(&(record_buffer[ASCIW_LENGTH_BYTES +
       sizeof(bank_name)  +
       ASCIW_LENGTH_BYTES]),
    bank_location, strlen(bank_location));
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
        /*****************************************************/
        /* total length =         */
        /*    ASCIW length size (2 bytes) +      */
        /*    size of the bank name       +      */
        /*    ASCIW length size (2 bytes) +      */
        /*    size of bank location.       */
        /*****************************************************/
        record_desc.dsc$w_length = ASCIW_LENGTH_BYTES +
       sizeof(bank_name)  +
       ASCIW_LENGTH_BYTES +
       sizeof(bank_location);
        record_desc.dsc$b_class = DSC$K_CLASS_S;
        record_desc.dsc$a_pointer = record_buffer;
                     cond_status = epc$end_event(0, /* No efn used     */
         FACILITY_NUMBER, /* Facility Number  */
         SIGNIN_EVENT, /* Event ID     */
         &si_event_handle, /* Distiguish this  */
       /* occurrence with  */
       /* the above start  */
         0,   /* No Context, single*/
       /* threaded         */
         &record_desc, /* Data buffer     */
         0,   /* No status required*/
         0,   /* No AST     */
         0,   /* No AST parameter */
         EPC$M_NOEF  /* Don't clear/set efn*/
         );
        }
     new_account = TRUE;
       break;
```

(continued on next page)

Example A–1 (Cont.) EPC$ATM-SAMPLE-EXTENDED Facility

```
        default:
                  if (s_event_flags[ERROR_DISPLAY - 1] != FALSE)
      {
                     cond_status = epc$event(0, /* No efn used     */
        FACILITY_NUMBER, /* Facility Number  */
        ERROR_DISPLAY, /* Event ID     */
        0,   /* No Context, single*/
      /* threaded          */
        0,   /* No user Data buffer*/
        0,   /* No status required*/
        0,   /* No AST     */
        0,   /* No AST parameter */
        EPC$M_NOEF  /* Don't clear/set efn*/
        );
                    };
                  printf("\n%s",invalid_selection);
                  printf("\n%s",acknowledge_message);
          scanf("%c",&temp_char);
                  break;
        } /* end case */
     } /* end if seek into file worked */
           else
            {
      /* We don't need to check the event ID and these are the    */
      /* minimum events that need to be passd to epc$event    */
             cond_status = epc$event(0,FACILITY_NUMBER,ERROR_DISPLAY);
             printf("\n\n\n%s",no_such_account);
             printf("%s",acknowledge_message);
      scanf("%c",&temp_char);
      new_account = TRUE;
            }
      } /* end while account != 0) */
    } /* end while new_account TRUE */
  close_and_exit_program();
  }; /* end main */
```

B

Registering a Cross-Facility Item

If you plan to instrument a widely distributed application, registering cross-facility items that the application uses ensures that they are unique in all Oracle Trace environments. To register cross-facility items, please complete this application and FAX or mail it to:

```
Oracle Corporation
New England Development Center
Oracle Trace Development Group
One Oracle Drive
Nashua, NH 03062

FAX: 603-897-3334
```

Glossary

administration database

A single Oracle Rdb database that contains the Oracle Trace facility definitions, facility selections, and schedule information. There is one Oracle Trace administration database per system or VMScluster.

application program

A sequence of instructions and routines, not part of the basic operating system, designed to serve the specific needs of a user. An application program can be instrumented with Oracle Trace service routine calls. Also referred to as a **facility**, especially after the Oracle Trace calls have been added.

AST

See **asynchronous system traps**.

asynchronous system traps

A software-simulated interrupt to a user-defined service routine. ASTs enable a user process to be notified asynchronously, with respect to the process, of the occurrence of a specific event. Many of the Oracle Trace service routines use ASTs to reduce execution time.

buffered I/O

Buffered I/O used during processing. This indicates that a device is transferring data to or from a buffer in the nonpaged pool.

collection class

A set (or group) of events and items that can be collected for a facility. Classes are associated with a particular purpose. Classes for a facility are specified in the facility definition. Users refer to a class when creating a facility selection. There can be one or more classes for each facility. Typical classes include CAPACITY_PLANNING, PERFORMANCE, and WORKLOAD.

collection interval

See **interval**.

collection name

A 1- to 32-character string that represents the name of a particular collection.

CPU usage

CPU time charged to the process.

cross-facility items

A set of common items that can be set and read by separate facilities. Oracle Trace can create reports relating these items.

data collection

The process of collecting data on a system or VMScluster. Criteria in the scheduling of data collection include when to collect data, where to put the output, and which facility selection to use. Data collection must be scheduled on a system in order to collect data. Multiple collections may be active on a node or VMScluster at any time.

data collection file

A file that contains raw data gathered during data collection. A single data collection file stores data for one or more OpenVMS processes. The file is created by the SCHEDULE COLLECTION command and is written to by processes which are running instrumented applications.

data formatting

Organizing collected data into an Oracle Rdb database or a formatted VAX RMS file. Collected data must be formatted before Oracle Trace can generate reports based on it.

data merging

Combining several data files into a single formatted database during data formatting.

direct I/O

Direct I/O used during processing. In direct I/O operation, a device transfers directly to or from memory using the users address space.

duration event

See **event**.

elapsed time

Time interval between the start and end of a duration event.

end event

The end of a duration event. See also **event**.

event

An occurrence of some activity within a facility. There are two types of events: **duration** and **point**. Duration events have logical beginning and ending points. Point events occur instantaneously. You can define up to 128 events for each facility.

event flag

A Boolean value corresponding to a particular event for a facility. If an event flag is set, it indicates that data capture is enabled for that event.

event collection flags list

A 128-element array of event flags for all events within a facility. Each facility has its own event collection flags list.

event handle

A unique numeric identifier generated by Oracle Trace on the EPC$START_EVENT routine call that identifies the start and end for a particular instance of a duration event.

event ID

A numeric representation of a predefined event. Valid event identifiers range from 1 to 128.

event name

A 1- to 15-character string that names an event.

event pair

The matching start and end events that indicate the occurrence of a duration event.

event record

A record buffer used to pass the facility-specific items for an event to the Oracle Trace service routines.

facility

Software, usually referred to as a layered product, that serves a particular purpose and operates under OpenVMS. See also **application program**.

facility definition

A description of the events and items that Oracle Trace can capture for a particular facility. Each facility for which Oracle Trace can collect data must have a facility definition stored in the Oracle Trace administration database.

facility library

A text library that is referenced during Oracle Trace reinstallations. Facilities installed on your system before the installation of Oracle Trace can store their facility definitions in the facility library. The file specification for the library is SYS$COMMON:[SYSLIB]EPC$FACILITY.TLB.

facility selection

A description of what to collect during data collection. Facility selections include a list of facilities and their collection classes. One or more data collections can use the same facility selection.

file list

A file containing a list of file specifications to use as data capture files. Each file specification must be on a separate line within the file list. The default file type for the file list is TXT.

formatted data file

A file that contains data organized for reporting. The formatted data file can contain data from one or more data files.

frequency

How many times an event occurred in the collection interval.

history database

A single Oracle Rdb database that contains all of the informational and error messages associated with data collection. There is one history database per system or VMScluster. You reference the history database with the logical name EPC$HISTORY_DB.

instrumenting

The act of adding Oracle Trace service routine calls to an application program so that event data can be collected.

interval

A time period when data collection takes place.

item

A numeric or string value that can be collected for an occurrence of an event. Up to 128 items can be captured for each event.

item flag

A 128-bit element of the item flags list. Each bit corresponds to an item that can be captured for a particular event for a facility. Each event has a 128-bit item flag associated with it.

item flags list

A 128-element array of item flags for all events for the facility. Each facility has its own item flags list.

item ID

A numeric representation of an item. Valid item identifiers range from 1 to 128.

item name

A 1- to 15-character string representing the name of an item.

local collection

A collection that is active or pending either on a standalone system or on one node in a VMScluster as opposed to data collection scheduled cluster-wide.

local node

The system you are currently logged in to.

non-registered facility

A facility whose facility ID number is not registered with Oracle Corporation. The facility ID for a non-registered facility is in the range of 2048 to 4095. Non-registered facilities usually represent customer-created applications. See also **facility**.

Oracle Trace service routines

See **service routines**.

page faults

Total number of hard and soft page faults used during processing. A page fault occurs when a program refers to an address that has not yet been mapped into physical memory.

page fault I/O

Number of hard page faults. That is, page faults that require disk I/O.

point event

See **event**.

registered facility

A facility whose facility ID number is registered with Oracle Corporation. The facility ID for a registered facility is in the range of 1 to 2047. Registered facilities usually represent either Oracle Corporation or Digital Equipment Corporation layered products. See also **facility**.

registrar process

A detached process that handles all communication between the applications instrumented with Oracle Trace routine calls and the Oracle Trace administration database.

registration identifier

A 1- to 255-character string that is useful in distinguishing separate images that use the same facilities. You use the /REGISTRATION_ID qualifier to the SCHEDULE COLLECTION command to collect data from processes with a specific registration ID.

remote

Pertaining to or originating from another node in the network.

reporting

The process of creating reports based on a formatted file or database. Oracle Trace can create reports based on data stored in an Oracle Rdb database.

resource utilization items

A set of standard items that Oracle Trace collects for all facilities. The items are referenced by the item group name RESOURCE_ITEMS. See Table 4–6 for a list of these items.

service routines

A set of predefined Oracle Trace routines whose calls are instrumented in the source code of an application program so that event data can be collected.

source program

A program that expresses an algorithm in a programming language such as FORTRAN, COBOL, or Pascal. After a source program is instrumented with Oracle Trace service routine calls, it is referred to as a facility.

start event

The beginning of a duration event. See **event**.

Index

D

E

F

G

T

U

V

W