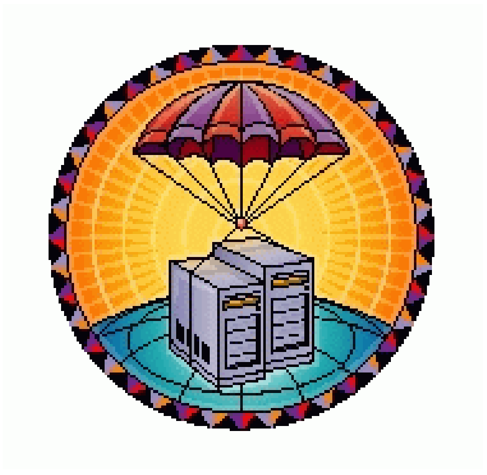
Cluster-Aware ODBC and OCI Client Applications for Oracle Fail Safe Solutions

An Oracle Technical White Paper
October, 2001



Laurence Clarke
Oracle New England Development Center
Server Technologies
failsafe_us@oracle.com

Table of Contents

Introduction ---------- 4
Client Issues During Failover ---------- 5
ODBC Client Applications ---------- 6
OCI Client Applications ---------- 7
OCI Programming for Failover ---------- 9
Updating the TNSNAMES.ORA file ---------- 11
Multitier and Non-OCI Solutions ---------- 13
Conclusion ---------- 13
Appendix ---------- 14
Sample Program: FsFailover.c ---------- 14

Introduction

This white paper describes how to create cluster-aware ODBC and OCI client applications for use with Oracle database servers configured with Oracle Fail Safe on Windows NT and Windows 2000 Microsoft Cluster Server clusters. Sample code is included in the Appendix for an Oracle OCI client application that demonstrates transparent failover for application users. The target audience is application designers, developers, and database administrators.

System downtime can happen at any time (planned maintenance or unplanned outages); however, the business and economic impact of downtime can be minimized by combining a highly available database solution, such as Oracle9*i* with Oracle Fail Safe, with "cluster-aware" client applications that automatically reconnect to the database after a failure and replay any transactions that were rolled back during database recovery. In many cases, downtime can be made completely invisible to users.

The first step toward deploying highly available client applications is to ensure fast data server failover. Failover times on well configured clusters are typically under a minute, with many customers reporting failover times of 30 seconds or less. Data server failover times can be minimized by optimizing the cluster hardware configuration, network configuration, failover policies, and database tuning.

The second step is to make client applications cluster-aware, as described in this paper. In general, most "out-of-the-box" client applications are not automatically cluster-aware. That is, application users will notice the effects of a database failover from one cluster node to the other. To client applications, a failover appears like an instant system reboot. The application (or end user) must then reconnect to the database and recover the pre-failure state, including any transactions rolled back during database recovery. Cluster-aware applications automate these steps and can make the effects of database failover invisible to end users.

Starting with Oracle8 release 8.0.5, the Oracle ODBC driver is enhanced to automatically make all Oracle ODBC clients cluster-aware without the need for any additional application coding. OCI client applications that use Net8 or Oracle Net to access databases configured with Oracle Fail Safe can also be made cluster-aware. Client applications that use Oracle8 release 8.0.6, Oracle8*i* release 8.1.6 and later, or Oracle9*i* OCI (including SQL*Plus, Oracle Objects for OLE, or Thick JDBC clients) all can take advantage of automatic transparent application failover without any additional coding.

Although much of the technical content of this paper also is valid for Oracle Real Application Clusters client applications, the focus of this paper is specific to client applications that connect to single-instance databases configured with the Oracle Fail Safe. For more information on other Oracle9*i* high availability and disaster recovery features, refer to the materials available online at http://www.oracle.com/ip/deploy/database/oracle9i/index.html?ha_home.html.

Client Issues During Failover

All Oracle Fail Safe solutions gain the following benefits over standalone database solutions without requiring any changes to existing client applications:

Benefit	Description
Highly Available	Oracle Fail Safe databases are highly available to clients at all times. Failover timing is optimized by performing a checkpoint before all planned failovers. Incremental checkpoints optimize recovery for unplanned failovers. Accelerated database reconnections allow clients to resume read-only work quickly.
Easy to Use and Support	Oracle Fail Safe Manager extends Oracle Enterprise Manager operations to Oracle databases running on MSCS clusters. It includes wizards to automate clusterwide configuration, extensive online help, a tutorial, and an integrated family of tools that automatically diagnose and fix common configuration problems. Oracle Fail Safe ensures that all clusterwide operations succeed, or that the entire operation is undone. Databases are not left in an indeterminate state.
Easy to Integrate with Applications	No changes are required for applications to access databases configured with Oracle Fail Safe. These databases are available at a fixed virtual server address, regardless of the physical cluster node hosting the database. In addition, Oracle OCI client applications can take advantage of automatic reconnection after failover, replay of interrupted SELECT statements, and callback functions that help to automate state recovery. The Oracle ODBC driver also supports automatic database reconnection and replay of interrupted SELECT statements without the need for any additional application coding.

Table 1: Oracle Fail Safe Key Features

To client applications, an outage appears an instant system reboot. For both planned outages (for example, during an operating system upgrade) or unplanned outages (for example, as a result of a hardware or software failure), client applications must perform the same recovery steps:

1. Reconnect to the Oracle data server using the node-independent IP address or network name of the Fail Safe group (virtual server) containing the database.

2. Reissue any interrupted SQL queries or uncommitted transactions that were rolled back during recovery (recover the session state). Note that many existing client applications are already coded to reissue transactions to handle deadlock situations.

Client applications that use current releases of the Oracle ODBC driver or Oracle OCI can use transparent application failover features to automatically reconnect to the Oracle data server, and, optionally, replay interrupted SELECT statements without the need for any additional application coding changes. Oracle OCI clients also optionally can register a failover callback function to help automate additional application specific recovery tasks (the sample OCI application in the Appendix uses a callback function to display informational messages during the failover process, for example).

Figure 1 summarizes the three steps in the failover process for such “cluster-aware” database client applications. After the initial results for a query are returned (step 1), the application software reconnects if it encounters errors due to a failover (step 2), and then replays the query and returns the rest of the data to the user (step 3).

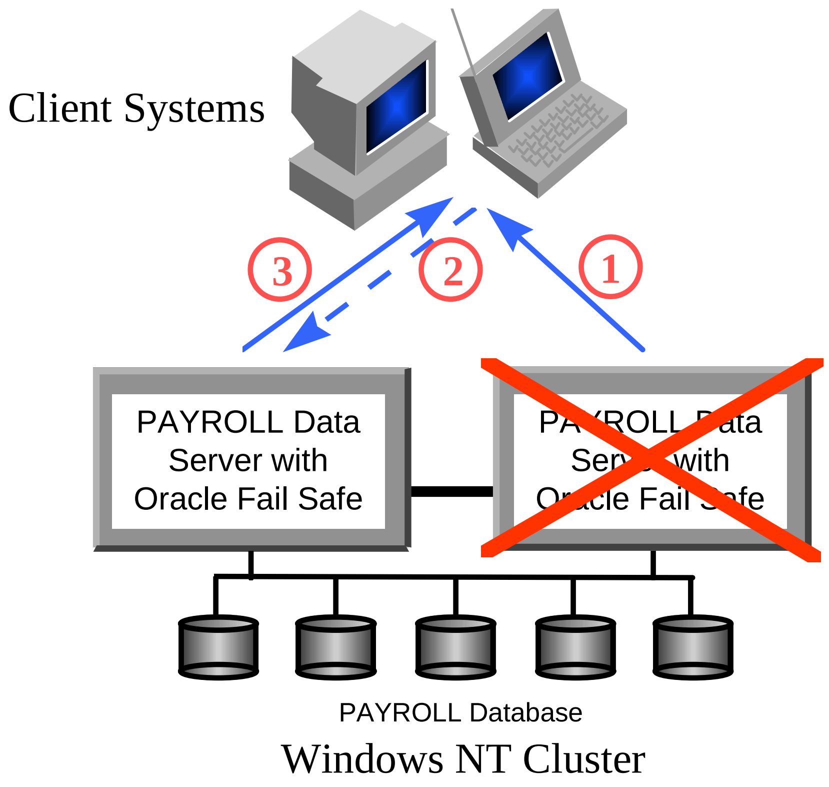

Client Systems
3
2
1
PAYROLL Data Server with Oracle Fail Safe
PAYROLL Data Server with Oracle Fail Safe
PAYROLL Database
Windows NT Cluster

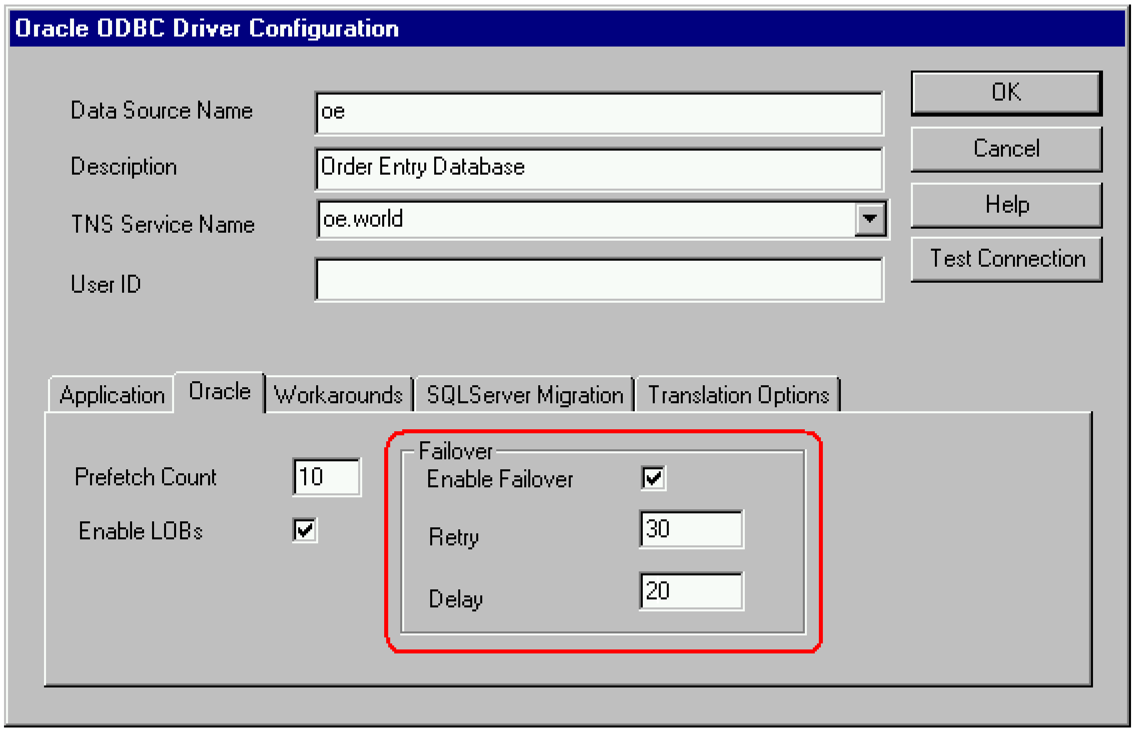
Figure 1: Transparent failover

Note that database applications have state (which may include a set of prepared cursors, an outstanding transaction, a set of in-process SELECT statements, and username/password information needed at logon). For client applications with state information, the simple IP address and cluster resource failover solution provided by Microsoft Cluster Server (MSCS) cluster software is not by itself sufficient to ensure transparent client application failover, since none of the state information is recovered. Fortunately, ODBC and OCI client applications that connect to databases configured with Oracle Fail Safe, as described in the next sections, can take full advantage of transparent application failover features. In most cases, pre-failover state can be restored and end users, other than experiencing a brief pause in their work, may never know a failover occurred.

ODBC Client Applications

Current releases of the Oracle ODBC driver layer over the OCI interface and Net8 or Oracle Net. For release 8.0.4 and earlier, none of the OCI transparent failover features are enabled in the ODBC driver, and ODBC clients must manually reconnect and restore state. Starting with release 8.0.5, however, the Oracle ODBC driver is enhanced so that all ODBC clients, without any additional coding, can take advantage of automatic database reconnection and replay of SELECT statements. Oracle ODBC drivers (release 8.0.5 and later) support transparent application failover for clients connected to Oracle8 release 8.0.3 and later databases.

Automatic failover is easily enabled during Oracle ODBC driver data source setup, as shown in Figure 2. The TNSNAMES.ORA file also must be updated, just as for OCI client applications (see section Updating the TNSNAMES.ORA file). No client application coding changes are needed. The latest Oracle ODBC drivers are available online through the Oracle Technology Network at http://technet.oracle.com/software/tech/windows/odbc/content.html.



Data Source Name: oe
Description: Order Entry Database
TNS Service Name: oe.world
User ID:

Prefetch Count: 10
Enable LOBs: ✓

Enable Failover: ✓
Retry: 30
Delay: 20

Figure 2: Enabling application failover during Oracle9*i* ODBC driver data source setup

OCI Client Applications

Although including code specifically to trap and handle all failover related OCI errors is a possible solution for OCI client applications, much of the necessary work can be performed automatically by using OCI transparent failover features. These features work only for client applications built using Oracle8 or later OCI that connect through Net8 or Oracle Net to Oracle8 or later databases; applications that connect to Oracle7 databases or that do not use Net8 or Oracle Net to access the database cannot use this approach.

Transparent failover allows client application users to continue working after the application loses its connection to a data server. While users may experience a brief pause during the time the database server fails over to a surviving cluster node, the session context is preserved. After failover completes, the application automatically reconnects to the database and, if desired, can continue retrieving data from SELECT statements initiated before the failure. After a failover, only the execution of interupted SELECT statements can be resumed (however, any updates are rolled back and OCI reports an error). All other calls are rolled back and OCI reports an error message that can be trapped and handled by the application.

Applications also need to check the state of transactions that were committed from the client side, but for which no acknowledgment was received back from the server—some client applications, like SAP, log committed transactions in the database and can quickly determine if a given transaction was indeed committed or if it needs to be reissued.

With transparent failover, application users see no loss of connection as long as the database instance successfully fails over to the surviving cluster node and can continue serving the application. To accomplish this, the application should preserve as much of the pre-failover state information for each user session as possible. Figure 3 illustrates the steps involved in the OCI failover process (also illustrated in the code example included in the Appendix).

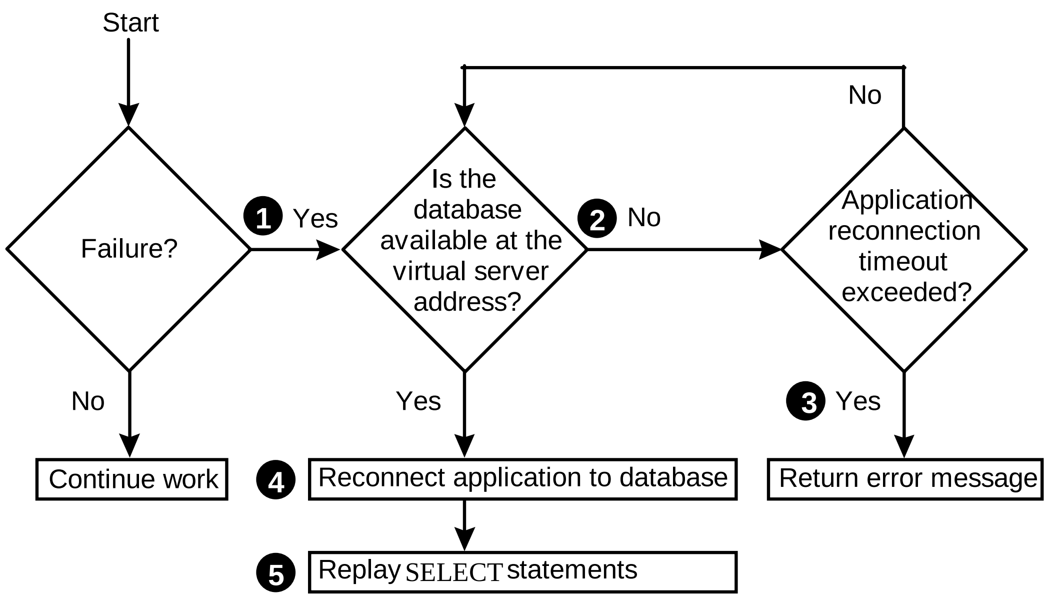

Start
Failure?
1 Yes
No
Continue work
Is the database available at the virtual server address?
2 No
Yes
4 Reconnect application to database
5 Replay SELECT statements
No
Application reconnection timeout exceeded?
3 Yes
Return error message

Figure 3: OCI transparent failover

The following list describes the steps shown in Figure 3:

1. Work proceeds normally until a failure occurs, causing the client application to lose its database connection. (The application discovers the lost connection when it next requests work by the Oracle database.)

2. The OCI library routines first invoke the application's failover callback function (if implemented) when the loss of connection is detected. Before returning from the callback function, the application can issue a message informing the user of possible delays during failover. Starting with Oracle8 release 8.0.6 (or with Oracle8*i* release 8.1.6), use of an explicit failover callback function is optional (the delay required to ensure the database instance has restarted on the failover node can be configured in the TNSNAMES.ORA file).

3. If the database connection cannot be restored within the designated time period, the application can return a status message to the user.

4. OCI reconnects the application to the database at the virtual server address.

5. When the database connection is reestablished, OCI can reexecute interrupted SELECT statements after a failover. The same number of rows as were fetched before the failure are retrieved and discarded, so that the next fetch continues from where the initial query was interrupted. The original query snapshot time is used when the query is reissued to ensure that the same set of rows will be returned. This feature is activated if the application connection to the database specifies a FAILOVER_MODE with TYPE = SELECT in the CONNECT_DATA string of the TNSNAMES.ORA file.

Two factors must be considered carefully whenever using TYPE = SELECT:

- The ordering of rows retrieved by a SELECT statement is not fixed; for this reason, queries that might be replayed should contain an ORDER BY clause. However, even without an ORDER BY clause, rows returned by the reissued query are nearly always returned in the initial order—known exceptions are queries that execute using the hash join or parallel query features. If an ORDER BY clause is not used, OCI will check that the set of discarded rows matches those previously retrieved to ensure that the application does not generate incorrect results.

- Recovery time after a failover can be significantly longer when using TYPE = SELECT. For example, if a query that retrieves 100,000 rows is interrupted by a failover after 99,990 rows have been fetched, then the client application will not be available for new work after a failover until 99,990 rows first have been refetched and discarded and the last 10 rows of the query have been retrieved.

Note that all other interrupted transactions are rolled back as part of database instance recovery after the failover. OCI applications can be coded to take the additional step of trapping the associated error messages and replaying these transactions.

In addition, if there are multiple sessions using a connection when failover occurs, OCI can fail over all of the sessions. However, if there were any ALTER SESSION statements executed for any of these sessions, OCI cannot replay them on the failover node. Any ALTER SESSION statement must be replayed by the application or user on the failover node.

OCI Programming for Failover

For complete OCI programming information, refer to the Oracle Call Interface documentation included with your Oracle database server. The following discussion assumes that you already have basic knowledge of OCI programming.

Oracle8 (and later) OCI support the use of handles (an opaque pointer to a storage area allocated by the OCI library). Handles are used to store information about contexts, connections and other OCI functions. Handles are used to store information about the failover callback. Handles make programming easier, because the OCI library, rather than the application, maintains the data. The use of handles in Oracle OCI is illustrated in the code example “FsFailover.c” given in the Appendix.

OCI Program Structure

Typically, an OCI application has the following flow of operations:

1. Initialize the OCI programming environment and processes.
2. Allocate necessary handles.
3. Establish server connection and user session.
4. Issue SQL statements to the server and perform application data processing.
5. Free statements and handles, and terminate user session and server connection.

All the handles except the bind and define handles are allocated with respect to a particular environment handle. Figure 4 shows the hierarchy of the handles.

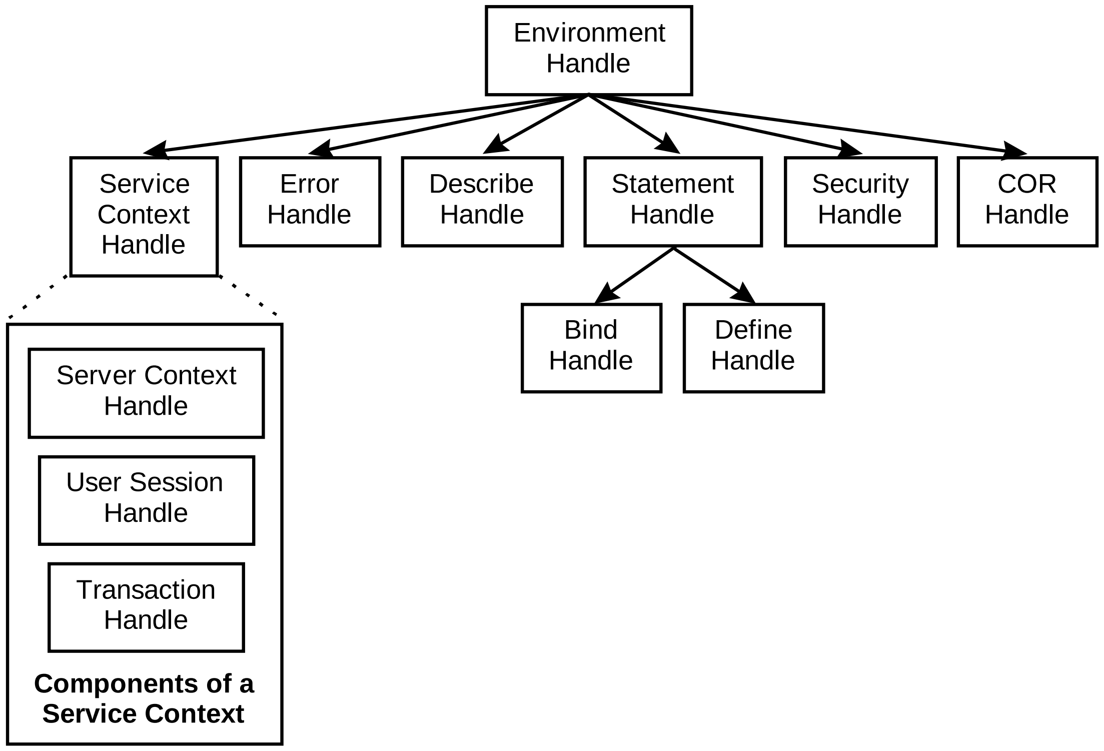


Figure 4: Hierarchy of Oracle OCI Handles

Application Failover Callbacks

Application failover callbacks are used to notify applications when failover is happening and to give applications a chance to handle failover-related processing. The basic structure of a failover callback function is as follows:

```
sb4 callback_fn       (dvoid   *svchp,
                       dvoid   *envhp,
                       dvoid   *fo_ctx,
                       ub4       fo_type,
                       ub4       fo_event);
```

The function arguments are used as follows:

svchp	service context handle
envhp	environment handle
fo_ctx	pointer in memory to store any necessary state or context
fo_type	indicates the type of failover, SESSION or SELECT
fo_event	indicates why the function has been called

Programming a failover callback function requires the following two steps:

1. Writing a failover callback function.
2. Registering the failover callback function in the service context handle.

The failover callback function can be used to notify the user about the failover, save the current state, and perform additional failover related tasks. The callback function is invoked multiple times during the course of reestablishing the user's session. The first call occurs when OCI detects the loss of connection. This callback is intended to allow the application to inform the user about the possible delay. If the failover is successful, a second callback occurs when the connection is reestablished. At this point, the application can reissue any interrupted database transactions that were rolled back during instance recovery and replay any “ALTER SESSION” statements as described previously. If failover is unsuccessful, the callback is called to inform the application that the failover will not occur and the application can then issue an appropriate error message.

Failover Callback Registration

The failover callback must be registered on the server context handle for it to be used. The registration is done by creating a callback definition structure and setting the OCI_ATTR_FOBCK attribute of the server handle to this structure. A structure type **OCIFocbkStruct** has been defined for the callback structure:

```
typedef struct
{
  OCICallbackFailover callback_function;
  dvoid *fo_ctx;
}
OCIFocbkStruct;
```

The structure element **callback_function** contains the address of the function to call while **fo_ctx** contains the address of the client context.

Refer to the sample program in the Appendix for examples of a callback function and of callback registration.

Updating the TNSNAMES.ORA file

To enable automatic reconnection for client applications, the TNSNAMES.ORA file must be updated. The changes described below are required to enable automatic reconnection for both Oracle8 (and later) ODBC clients and Oracle8 (and later) OCI clients. Note that some of the possible reconnection options are specific to Oracle Real Application Clusters and do not work with single-instance Oracle Fail Safe high-availability solutions.

The goal is to make the new connection appear as much like the old connection as possible. Ideally, the client application (and hence the user) will not notice that anything has changed. To accomplish transparent failover, it is essential to preserve as much of the application's pre-failover state as possible.

Reconnection is configured in the CONNECT_DATA string of the DESCRIPTION definition in either the TNSNAMES.ORA file or the Names Server for use by Oracle Names. Refer to the Net8 Administrator's Guide for complete information and syntax for connect operations. A sample connection descriptor in TNSNAMES.ORA looks like the following:

```
PAYROLL =
    (DESCRIPTION =
        (ADDRESS =
            (COMMUNITY = TCP.world)
            (PROTOCOL = TCP)
            (HOST = VSERVER)
            (PORT = 1521)
        )
        (CONNECT_DATA =
            (SID = HR)
            (FAILOVER_MODE =
                    (TYPE = SELECT)
                    (METHOD = BASIC)
                    (RETRIES = 20)
                    (DELAY = 15)
            )
        )
    )
```

Before application failover can occur, the application must reconnect to the database. The HOST parameter in the ADDRESS string controls client reconnection. Because Oracle Fail Safe already uses a virtual server name in the HOST parameter, you do not need to make any changes to the HOST parameter for reconnection to occur. After a failure, client applications will always use the fixed network address of the virtual server to reconnect to the database, and need not be concerned with which physical cluster node hosts the Oracle data server.

The FAILOVER_MODE parameter in the CONNECT_DATA string specifies the type and method of application failover. The TYPE keyword indicates whether or not to reestablish user sessions, and what application operations can continue if sessions are reestablished. The allowed options are NONE, SELECT, or SESSION; these are described further in Table 2.

Options	**Reconnection?**	**Application Queries**	**Comments**
NONE	Does not reestablish user sessions.	Failover functions are not applied.	Returns error message to the user without attempting to fail over. This is the default.
SELECT	Reestablishes user sessions on the new client connection.	SELECT statements started before the failure are reexecuted automatically on the new connection using the initial query snapshot time to ensure consistent data. Active transactions are rolled back and are not reexecuted on the new connection.	Incurs overhead on the client side for all SELECT statements, even if a failure does not occur. After a failover, the same number of rows as already retrieved are first fetched and discarded; then the next fetch from the client application continues retrieving rows. For large queries, this can increase recovery time after a failure.
SESSION	Reestablishes user sessions on the new client connection.	SELECT statements that were started before the failure are not reexecuted on the new connection.	Avoids the overhead that is incurred when you choose the TYPE=SELECT option.

Table 2: TYPE Keyword for the FAILOVER_MODE Parameter

The FAILOVER_MODE parameter METHOD keyword is used to optimize failover performance. With Oracle Fail Safe, there is only one active instance, so the BASIC failover method must be used. This option establishes the new connection at failover time. Because METHOD=BASIC is the default, its use in the CONNECT_DATA string is optional.

The RETRIES = <number> and DELAY = <time in seconds> parameters were introduced starting with Oracle8 release 8.0.6 and for Oracle8*i* release 8.1.6. In this example, the transparent application failover will attempt to reconnect every 15 seconds for 20 tries. If the reconnection is not established, an error will be returned.

Multitier and Non-OCI Solutions

Oracle Fail Safe provides high availability support not only to data tier components but also to application tier components such as Oracle HTTP Server, Oracle Forms and Reports Services, and Oracle Applications Release 11*i* components. When an application server is located between the database server and the client, transparent application failover can automatically restore connections between the application server and the database server, reestablish all package and session states on the database server, and replay any queries that were in progress at the time of failover. Using OCI transparent application failover features, developers can create multitier solutions that completely mask data server failures from other tiers in the environment.

In addition, some non-OCI clients (such as a Netscape browser, for example) contain automatic connection retry logic to accommodate network interruptions or delays. During read-only operations, this logic can effectively hide both application and data tier failovers from end users, who experience only a brief pause before being able to continue work. More typically, users will need to click on the reload or refresh button of their client application to reestablish a connection after a failover, and will need to reenter uncommitted data. Since all application and data tier components configured for high availability with Oracle Fail Safe use system-independent virtual addresses, client applications have a fixed access point (IP address or URL) and do not require any coding changes (for example, to support connections to alternate physical systems).

Conclusion

Starting with Oracle8 release 8.0.5, all Oracle8 ODBC clients can be made cluster-aware with no additional coding changes. In addition, starting with Oracle8 release 8.0.6 and Oracle8*i* release 8.1.6 and later, OCI clients can configure the needed failover delay in the TNSNAMES.ORA file. This enables transparent application failover support with no additional application coding changes for all client applications layered over OCI (including Oracle Objects for OLE, Thick JDBC, and SQL*Plus) that connect to databases configured with Oracle Fail Safe.

The Appendix provides a sample Oracle OCI program that illustrates the registration and use of an OCI failover callback function. During a failover, Oracle applications can use a failover callback function to alert users that a failover is in progress and to perform application specific tasks such as deleting and resubmitting a partially written report or recovering from an interrupted update operation. Oracle Fail Safe release 3.2 also ships with a transparent application failover demonstration application (TAFDemoGui.exe) located in the <ORACLE_HOME>\fs\fsmgr\sample directory.

With some multitier high availability solutions configured with Oracle Fail Safe, it is also possible to hide failovers from non-OCI clients (such as Netscape). In most cases, simply clicking on the browser reload button allows a client to reconnect after a failover and continue working with minimal downtime.

Deploying cluster-aware client applications with highly available application servers and databases configured with Oracle Fail Safe is a cost effective and efficient way to minimize or eliminate downtime. Often, the effects of failover are made completely invisible to end users.

Appendix

Sample Program: FsFailover.c

```
/*
Copyright (c) 1997, 2001 by Oracle Corporation. All rights reserved.
Name:               FsFailover.c
Date:               October 8, 2001
Description:        Demo program for transparent application failover.
Notes:              Demonstration code only; error handling, boundary cases, and other program areas  are not covered completely.
                    For 8.0.4 databases, use SECTION A and leave SECTION B commented out as in the printed example.
                    For 8.0.5 and later databases, you can use the example as written, or comment out SECTION A and use SECTION B.
                    For Oracle8 release 8.0.6, Oracle8i release 8.1.6 and later, and Oracle 9i databases, use of a callback function is optional
                    because the reconnection delay required to accommodate failover can be configured in the CONNECT_DATA section of
                    TNSNAMES.ORA file.  Note that some earlier database releases may not be supported by the very latest Oracle Fail Safe
                    releases; for example, Oracle Fail Safe release 3.2 supports Oracle database releases 8.0.6, 8.1.7, and 9.0.1.
*/

/* Includes */

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>
#ifdef      _STDC_
#include <ociap.h>
#else
#include <ocikp.h>
#endif

/* Local Variables */

static LPSTR pGUserName     = "internal";
static LPSTR pGPassword     = "internal";
static LPSTR pGDbName       = "payroll";

static LPSTR pGSelectQuery = "Select tablespace_name from all_users, all_tables order by tablespace_name";

static OCIEnv               *pGEnvh;
static OCIServer            *pGSrvh;
static OCIError             *pGErrh;
static OCISvcCtx            *pGSvch;
static OCIStmt              *pGStmth;
static OCIDefine            *pGDefine = (OCIDefine *) 0;
static OCIBind              *pGBind = (OCIBind *) 0;

/* Function prototypes */

void CheckOCIError(
                    OCIError            *pErrh,
                    sword               OCIError);

/* Subroutines */

/* Application Failover Callback */
DWORD  ApplicationFailoverCallback(
                    dvoid               *pSvch,
                    dvoid               *pEnvh,
                    dvoid               *pFailoverContext,
                    ub4                 FailoverType,
                    ub4                 FailoverEvent)
{
        switch (FailoverEvent)
        {
        case OCI_FO_BEGIN:
                {
                        OCISvcCtx                       *pLocSvch;
                        sword                           Status;
                        printf("Fail Safe Oracle8 database is failing over ... Please stand by\n");
                        printf("Failover type was found to be %s\n",
                                ((FailoverType == OCI_FO_SESSION) ? "SESSION" :
                                 (FailoverType == OCI_FO_SELECT) ? "SELECT" : "UNKNOWN!!"));
                        printf("Failover Context is : %s\n",
                                pFailoverContext ? (CHAR *) pFailoverContext : "NULL POINTER!");

                        //*** START SECTION A ***//
                        //Earlier Oracle8/8i OCI releases make only one attempt to reconnect after a failure is detected.
                        // OCI releases 8.0.6 and 8.1.6 or later contain enhancements to allow multiple reconnection attempts.
                        // When using earlier OCI libraries, the application needs to ensure that sufficient time elapses so
                        // that when it returns from the failover callback function, the OCI library will be able to successfully
                        // reconnect to the virtual server for the Oracle database.  A quick way to do this is to sleep a "reasonable"
                        // period of time to allow failover on the cluster to complete. A better solution (used below) is to write a loop
                        // in which the application repeatedly  attempts to reconnect to the database at the virtual server address and
                        // returns either after making (and then disconnecting from) a successful connection or after a specified time
                        // interval. For the most recent OCI releases, you can simply add RETRIES and DELAY parameters to the
                        // FAILOVER_MODE clause in the TNSNAMES.ORA file and eliminate the need for a callback function.
                        //
                        // Begin loop that tries to connect to the failover database until successfully connected.
                        while (1)
                        {
                                // Sleep for 10 seconds before retrying to connect to the failover database.
                                Sleep(10000);
                                printf("Retrying to connect to failover database after waiting 10 seconds...\n");

                                Status = OCILogon(pEnvh, pGErrh, &pLocSvch,
                                                pGUserName, strlen(pGUserName),
                                                pGPassword, strlen(pGPassword),
                                                pGDbName, strlen(pGDbName));
                                if (Status != OCI_SUCCESS)
                                {
                                        CheckOCIError(pGErrh, Status);
                                }
                                else
                                {
                                        printf("Successfully reconnected to failover database\n");
                                        CheckOCIError(pGErrh, OCILogoff(pLocSvch, pGErrh));
                                        break;
                                }
                        }

                        // If callback returns now immediately after reconnecting, the query that was executing may fail if the
                        // database is still performing recovery. To avoid possible failure, sleep for an arbitrary time of 10 seconds.
                        // The optimal time period will vary from application to application.
                        Sleep(10000);
                        //*** END SECTION A ***//
                        break;
                }
        case OCI_FO_ABORT:
                        printf("Failover aborted. Failover will not take place.\n");
                        break;
        case OCI_FO_END:
                        printf("Failover ended ... Resume services\n");
                        break;
        case OCI_FO_REAUTH:
                        printf("Failed over client connection… Resume services\n");
                        break;
//*** START SECTION B ***//
//
//      Starting with  Oracle8 release 8.0.5, the preceding call to OCILogon and the associated logic of section A are no longer needed.
//      Section A above can be commented out and Section B can be used instead. A new return code (currently 25410) tells OCI
//      to retry the database connection. Applications can determine what connection retry logic best makes sense (for example,. an
//      exponential backoff in retry frequency, with retry attempts stopped after some number of unsuccessful attempts). The
//      following sample code will retry indefinitely and should be modified appropriately for actual deployment. Also, consult the
//      Oracle8/8i/9i OCI documentation to verify the syntax and return values used by your OCI release.  For OCI releases 8.0.6 and
//      8.1.6 and later, you can, as previously noted, include RETRIES and DELAY parameters in the FAILOVER_MODE clause in
```

```
//	the TNSNAMES.ORA file and eliminate the need for a callback function.
//
//	case OCI_FO_ERROR:
//			printf("Reconnection attempt failed; waiting 10 seconds...\n");
//			Sleep(10000);
//			printf("Retrying to connect to failover database after waiting 10 seconds...\n");
//			return (25410);
//			break;
//*** END SECTION B ***//
	default :
			printf("Bad Failover Event: %d\n", FailoverEvent);
			return -1; /* Error File Oracle Bug report */
	} // end of Switch

	return 0;
} // ApplicationFailoverCallback */


/* Registration of the Application Callback */
sword	RegisterApplicationFailoverCallback(
				dvoid		*pSrvh,
				OCIError	*pErrh
				)
{
	OCIFocbkStruct	FailoverInfo;		/* Information for callback */

	/* Allocate Memory for context and fill the information */

	if (!(FailoverInfo.fo_ctx =
		(dvoid *)malloc(strlen("Application Masking Failover Errors"))))
		return -1;
	strcpy((LPSTR)FailoverInfo.fo_ctx, "Application Masking Failover Errors");
	FailoverInfo.callback_function = &ApplicationFailoverCallback;

	/* Do the registration with OCI */

	return OCIAttrSet(pSrvh, (ub4) OCI_HTYPE_SERVER,
					(dvoid *) &FailoverInfo, (ub4) 0,
					(ub4) OCI_ATTR_FOCBK, pErrh);

} /* end of RegisterApplicationFailoverCallback */

/* CheckOCIError */

void CheckOCIError(
			OCIError *pErrh,
			sword		OCIError
			)
{
	switch (OCIError)
	{
	case OCI_SUCCESS:
		break;
	case OCI_SUCCESS_WITH_INFO:
		printf("Error - OCI_SUCCESS_WITH_INFO \n");
		break;
	case OCI_NEED_DATA:
		printf("Error - OCI_NEED_DATA\n");
		break;
	case OCI_NO_DATA:
		printf("Error - OCI_NO_DATA\n");
		break;
	case OCI_ERROR:
		{
		text	ErrorBuffer[1024];
		sb4		ErrorCode = 0;
		(void) OCIErrorGet((dvoid *)pErrh, (ub4) 1, (text *)NULL, &ErrorCode, ErrorBuffer, 1024, OCI_HTYPE_ERROR);
		printf("Error - %.*s\n", 1024, ErrorBuffer);
		break;
		}
```

```
        case OCI_INVALID_HANDLE:
                printf("Error - OCI_INVALID_HANDLE\n");
                break;
        case OCI_STILL_EXECUTING:
                printf("Error - OCI_STILL_EXECUTING\n");
                break;
        case OCI_CONTINUE:
                printf("Error - OCI_CONTINUE\n");
                break;
        default:
                printf("Unknown Error! %d\n", OCIError);
                break;
        } /* switch */
} /* End of CheckOCIErrors */

/* Application OCI Cleanup */

void ApplicationOCICleanup()
{
        if (pGStmth)
                CheckOCIError(pGErrh, OCIHandleFree((dvoid *)pGStmth, OCI_HTYPE_STMT));
        if (pGErrh)
                (void) OCIServerDetach(pGSrvh, pGErrh, OCI_DEFAULT);
        if (pGSrvh)
                CheckOCIError(pGErrh, OCIHandleFree(pGSrvh, OCI_HTYPE_SERVER));
        if (pGSvch)
                OCIHandleFree(pGSvch, OCI_HTYPE_SVCCTX);
        if (pGErrh)
                OCIHandleFree(pGErrh, OCI_HTYPE_ERROR);
        return;
}/* End of ApplicationOCICleanup */

/* Application OCI Initialization */

void ApplicationOCIInitialization()
{
        (void) OCIInitialize((ub4) OCI_DEFAULT, (dvoid *)0,
                                                (dvoid *(*)(dvoid *, size_t))0,
                                                (dvoid *(*)(dvoid *, dvoid *, size_t))0,
                                                (void (*)(dvoid *, dvoid *))0);
        (void) OCIEnvInit(&pGEnvh, OCI_DEFAULT, (size_t)0,(dvoid **)0);

        (void) OCIHandleAlloc(pGEnvh, &pGErrh, OCI_HTYPE_ERROR, (size_t)0, (dvoid **)0);

        /* Set up the Server Contexts */

        (void) OCIHandleAlloc(pGEnvh, &pGSrvh, OCI_HTYPE_SERVER, (size_t)0, ( dvoid **)0);

        (void) OCIHandleAlloc(pGEnvh, &pGSvch, OCI_HTYPE_SVCCTX, (size_t)0, (dvoid **)0);

        CheckOCIError(pGErrh, OCIServerAttach(pGSrvh, pGErrh, pGDbName, strlen(pGDbName), 0));

        (void) OCIAttrSet(pGSvch, OCI_HTYPE_SVCCTX, pGSrvh, (ub4) 0, OCI_ATTR_SERVER, (OCIError *)pGErrh);

} /* end of ApplicationOCIInitialization */

void ApplicationOCISession(
                        OCISession **pOCISession
                        )

{
        (void) OCIHandleAlloc(pGEnvh, pOCISession, OCI_HTYPE_SESSION, (size_t) 0, (dvoid **)0);
        (void) OCIAttrSet((*pOCISession), (ub4) OCI_HTYPE_SESSION, (dvoid *) pGUserName, (ub4) strlen(pGUserName),
                                        (ub4) OCI_ATTR_USERNAME, pGErrh);
        (void) OCIAttrSet((*pOCISession), (ub4) OCI_HTYPE_SESSION, (dvoid *) pGPassword, (ub4) strlen(pGPassword),
                                        (ub4) OCI_ATTR_PASSWORD, pGErrh);
        CheckOCIError(pGErrh, OCISessionBegin(pGSvch, pGErrh, (*pOCISession), OCI_CRED_RDBMS, (ub4) OCI_DEFAULT));

        (void) OCIAttrSet(pGSvch, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)(*pOCISession), (ub4) 0, (ub4) OCI_ATTR_SESSION, pGErrh);
```

```
} /* ApplicationOCISession */

/* Main Application Program */

int main (
                    int       argc,
                    char      *argv[]
                    )
{
          OCISession *pSession = (OCISession *) 0;
          char                  pTablespaceName[512];
          sword                 OCIStatus;
          int                             i = 0;

          /* Initialize OCI and then establish the User Session */

          ApplicationOCIInitialization();

          ApplicationOCISession(&pSession);

          /* Now register the Application Callback for this User Session */

          CheckOCIError(pGErrh, (RegisterApplicationFailoverCallback(pGSrvh, pGErrh)));

          CheckOCIError(pGErrh, OCIHandleAlloc(pGEnvh, &pGStmth, OCI_HTYPE_STMT, 0, (dvoid **) 0));

          /* Prepare the Query to fetch */

          CheckOCIError(pGErrh, OCIStmtPrepare(pGStmth, pGErrh, pGSelectQuery,
                              (ub4) strlen(pGSelectQuery), (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

          CheckOCIError(pGErrh, OCIDefineByPos(pGStmth, &pGDefine, pGErrh, 1, (dvoid *)pTablespaceName,
                              512, SQLT_STR, (dvoid *)0, (ub2 *) 0, (ub2 *) 0, OCI_DEFAULT));

          CheckOCIError(pGErrh, OCIStmtExecute(pGSvch, pGStmth, pGErrh, (ub4) 1, (ub4) 0, (CONST OCISnapshot *) NULL,
                              (OCISnapshot *)NULL, OCI_DEFAULT));
          /* Now fetch the query data the Failover should be transparent during fetch */

          OCIStatus = 0;
          for (;;)
          {
                    if (OCIStatus)
                    {
                              if (OCIStatus == OCI_NO_DATA) break;
                              else
                              {
                                        CheckOCIError(pGErrh, OCIStatus);
                                        ApplicationOCICleanup();
                                        return OCI_ERROR;
                              }
                    }
                    i += 1;
                    printf("Tablespace_name : %s Count = %d\n", pTablespaceName, i);
                    OCIStatus = OCIStmtFetch(pGStmth, pGErrh, (ub4) 1, (ub4) 0, (ub4) 0);

          } /* for */

          /* Cleanup all OCI Handles */

          ApplicationOCICleanup();

          return OCI_SUCCESS;
}
```

Oracle Corporation World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.
Phone 650.506.7000
Fax 650.506.7200

International Inquiries:
Phone 44.932.872.020
Telex 851.927444(ORACLEG)
Fax 44.932.874.625
http://www.oracle.com/