EDB OCL Connector

Version 17

Built at 2025-03-21T14:50:56

1 EDB OCL Connector

The EDB OCL Connector provides an API similar to the Oracle Call Interface. You can use EDB's OCL Connector to compile applications that are written to use the Oracle Call Interface to interact with an EDB Postgres Advanced Server database server.

> **Note**
>
> EDB doesn't support the use of the Open Client Library with Oracle Real Application Clusters (RAC) and Oracle Exadata. These Oracle products aren't evaluated or certified with this EDB product.

2 EDB OCL Connector release notes

The EDB OCL connector documentation describes the latest version of the EDB OCL connector.

Release notes describe what's new in a release. When a minor or patch release introduces new functionality, indicators in the content identify the version that introduced the new feature.

Version	Release date
17.2.0.1	22 Nov 2024

2.1 EDB OCL Connector 17.2.0.1 release notes

Released: 22 Nov 2024

The EDB OCL Connector provides an API similar to the Oracle Call Interface.

New features, enhancements, bug fixes, and other changes in the EDB OCL Connector 17.2.0.1 include:

Type	Description
Enhancement	Added support for EDB Postgres Advanced Server version 17.2.

3 Supported platforms

The EDB OCL Connector is supported on the same platforms as EDB Postgres Advanced Server. To determine the platform support for the EDB OCL Connector, you can refer either to the platform support for EDB Postgres Advanced Server on the Platform Compatibility page on the EDB website or to Installing EDB OCL Connector.

4 libpq cross-version compatibility

EDB OCL installation always uses the latest libpq. When upgrading to a new major release of EDB Postgres Advanced Server, the different scenarios supported under libpq cross-version compatibility are as follows:

- If the latest libpq is installed on the machine, OCL uses it.
- If the latest libpq isn't already installed, OCL installs it. It doesn't use the existing libpq of older versions even if it's installed.
- If you upgrade the OCL version, then libpq is also upgraded to its latest version.

If you're upgrading to a minor release, you need to manually upgrade libpq.

Upgrading libpq for minor releases of EDB Postgres Advanced Server

For minor releases of EDB Postgres Advanced Server, you might need to upgrade libpq to a required version on the client machine where you installed EDB OCL Connector. (Any new libpq version dependencies are listed in the release notes.) If you need to upgrade libpq, run the appropriate command for your operating system.

For Ubuntu/Debian

```
sudo apt-get install edb-as15-libpq5
```

For RHEL and SLES

```
sudo <package-manager> install edb-as15-server-libs
```

Where `<package-manager>` is the package manager used with your operating system:

Package manager	Operating system
dnf	RHEL 8/9 and derivatives
zypper	SLES

5 Installing EDB OCL Connector

Select a link to access the applicable installation instructions:

Linux x86-64 (amd64)

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9, RHEL 8
- Oracle Linux (OL) 9, Oracle Linux (OL) 8
- Rocky Linux 9, Rocky Linux 8
- AlmaLinux 9, AlmaLinux 8

SUSE Linux Enterprise (SLES)

- SLES 15

Debian and derivatives

- Ubuntu 22.04, Ubuntu 20.04
- Debian 12, Debian 11

Linux IBM Power (ppc64le)

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9, RHEL 8

SUSE Linux Enterprise (SLES)

- SLES 15

Linux AArch64 (ARM64)

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9
- Oracle Linux (OL) 9

Debian and derivatives

- Debian 12

Windows

- Windows Server 2019

5.1 Installing EDB OCL Connector on Linux IBM Power (ppc64le)

Operating system-specific install instructions are described in the corresponding documentation:

Red Hat Enterprise Linux (RHEL)

- RHEL 9
- RHEL 8

SUSE Linux Enterprise (SLES)

- SLES 15

5.1.1 Installing EDB OCL Connector on RHEL 9 ppc64le

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Install the EPEL repository:

  ```
  sudo dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
  ```
- Refresh the cache:

  ```
  sudo dnf makecache
  ```

Install the package

```
sudo dnf -y install edb-oci
sudo dnf -y install edb-oci-devel
```

5.1.2 Installing EDB OCL Connector on RHEL 8 ppc64le

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Install the EPEL repository:

  ```
  sudo dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
  ```
- Refresh the cache:

  ```
  sudo dnf makecache
  ```

Install the package

```
sudo dnf -y install edb-oci
sudo dnf -y install edb-oci-devel
```

5.1.3 Installing EDB OCL Connector on SLES 15 ppc64le

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  zypper lr -E | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Activate the required SUSE module:

  ```
  sudo SUSEConnect -p PackageHub/15.4/ppc64le
  ```
- Refresh the metadata:

  ```
  sudo zypper refresh
  ```

Install the package

```
sudo zypper -n install edb-oci
sudo zypper -n install edb-oci-devel
```

5.2 Installing EDB OCL Connector on Linux x86 (amd64)

Operating system-specific install instructions are described in the corresponding documentation:

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9
- RHEL 8
- Oracle Linux (OL) 9
- Oracle Linux (OL) 8
- Rocky Linux 9
- Rocky Linux 8
- AlmaLinux 9
- AlmaLinux 8

SUSE Linux Enterprise (SLES)

- SLES 15

Debian and derivatives

- Ubuntu 22.04
- Ubuntu 20.04
- Debian 12
- Debian 11

5.2.1 Installing EDB OCL Connector on RHEL 9 or OL 9 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Install the EPEL repository:

  ```
  sudo dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
  ```

Install the package

```
sudo dnf -y install edb-oci
sudo dnf -y install edb-oci-devel
```

5.2.2 Installing EDB OCL Connector on RHEL 8 or OL 8 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Install the EPEL repository:

  ```
  sudo dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
  ```

Install the package

```
sudo dnf -y install edb-oci
sudo dnf -y install edb-oci-devel
```

5.2.3 Installing EDB OCL Connector on AlmaLinux 9 or Rocky Linux 9 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Install the EPEL repository:

  ```
  sudo dnf -y install epel-release
  ```
- Enable additional repositories to resolve dependencies:

  ```
  sudo dnf config-manager --set-enabled crb
  ```

Install the package

```
sudo dnf -y install edb-oci
sudo dnf -y install edb-oci-devel
```

5.2.4 Installing EDB OCL Connector on AlmaLinux 8 or Rocky Linux 8 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Install the EPEL repository:

  ```
  sudo dnf -y install epel-release
  ```
- Enable additional repositories to resolve dependencies:

  ```
  sudo dnf config-manager --set-enabled powertools
  ```

Install the package

```
sudo dnf -y install edb-oci
sudo dnf -y install edb-oci-devel
```

5.2.5 Installing EDB OCL Connector on SLES 15 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  zypper lr -E | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Activate the required SUSE module:

  ```
  sudo SUSEConnect -p PackageHub/15.4/x86_64
  ```
- Refresh the metadata:

  ```
  sudo zypper refresh
  ```

Install the package

```
sudo zypper -n install edb-oci
sudo zypper -n install edb-oci-devel
```

5.2.6 Installing EDB OCL Connector on Ubuntu 22.04 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

```
sudo apt-get -y install edb-oci
sudo apt-get -y install edb-oci-dev
```

5.2.7 Installing EDB OCL Connector on Ubuntu 20.04 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

```
sudo apt-get -y install edb-oci
sudo apt-get -y install edb-oci-dev
```

5.2.8 Installing EDB OCL Connector on Debian 12 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

```
sudo apt-get -y install edb-oci
sudo apt-get -y install edb-oci-dev
```

5.2.9 Installing EDB OCL Connector on Debian 11 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

```
sudo apt-get -y install edb-oci
sudo apt-get -y install edb-oci-dev
```

5.3 Installing EDB OCL Connector on Linux AArch64 (ARM64)

Operating system-specific install instructions are described in the corresponding documentation:

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9
- Oracle Linux (OL) 9

Debian and derivatives

- Debian 12

5.3.1 Installing EDB OCL Connector on RHEL 9 or OL 9 arm64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Install the EPEL repository:

  ```
  sudo dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
  ```

Install the package

```
sudo dnf -y install edb-oci
sudo dnf -y install edb-oci-devel
```

5.3.2 Installing EDB OCL Connector on Debian 12 arm64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

```
sudo apt-get -y install edb-oci
sudo apt-get -y install edb-oci-dev
```

5.4 Installing on Windows

EDB provides a graphical interactive installer for Windows. You can access it in two ways:

- Download the graphical installer from the Downloads page, and invoke the installer directly. See Installing directly.
- Use StackBuilder Plus with EDB Postgres Advanced Server to download the EDB installer package and invoke the graphical installer. SeeUsing StackBuilder Plus.

Installing directly

After downloading the graphical installer, to start the installation wizard, assume sufficient privileges (superuser or administrator) and double-click the installer icon. If prompted, provide a password.

In some versions of Windows, to invoke the installer with administrator privileges, you need to right-click the installer icon and select **Run as Administrator** from the context menu.

Proceed to Using the graphical installer.

Using StackBuilder Plus

If you're using EDB Postgres Advanced Server, you can invoke the graphical installer with StackBuilder Plus. SeeUsing StackBuilder Plus.

1. In StackBuilder Plus, follow the prompts until you get to the module selection page.

   On the Welcome page, from the list of available servers, select the target server installation. If your network requires you to use a proxy server to access the internet, select **Proxy servers** and specify a server. Select **Next**.

2. Expand the **Database Drivers** node, and select **EnterpriseDB OCI Connector**.

3. Proceed to Using the graphical installer.

Using the graphical installer

1. Select the installation language and select **OK**.

2. On the Setup OCI page, select **Next**.

3. Browse to a directory where you want to install OCI, or leave the directory set to the default location. Select **Next**.

4. On the Ready to Install page, select **Next**.

   An information box shows the installation progress of the selected components.

5. When the installation is complete, select **Finish**.

5.5 Upgrading a Linux installation

If you have an existing OCL Connector installation on a Linux platform, you can upgrade your repository configuration file, which enables access to the current EDB repository. Then you can upgrade to a more recent version of OCL Connector.

To update the `edb.repo` file:

```
# Update your repository configuration file
sudo <package-manager> upgrade edb-repo

# Upgrade the installed product
sudo <package-manager> upgrade edb-oci

sudo <package-manager> upgrade edb-oci-devel
```

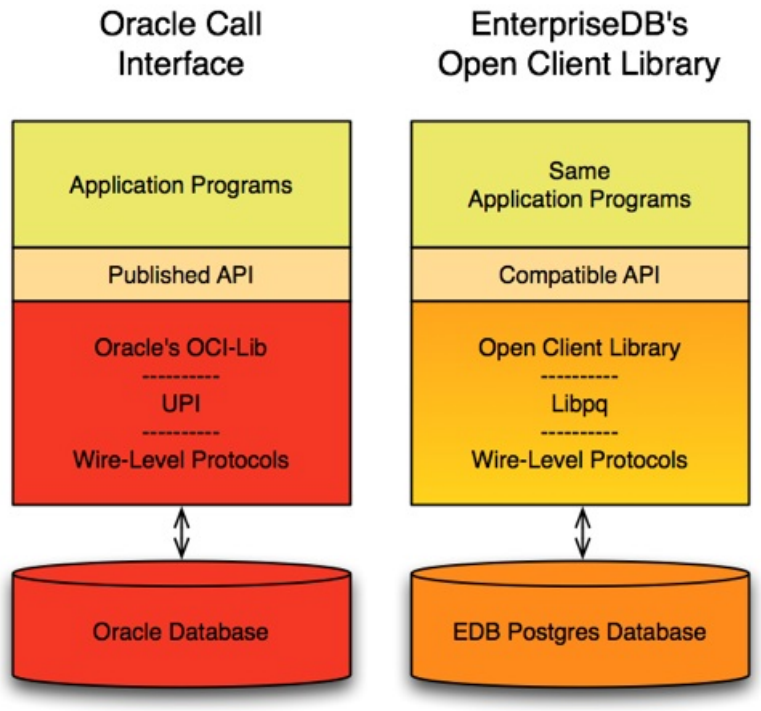
Where `<package-manager>` is the package manager used with your operating system:

Package manager	Operating system
dnf	RHEL 8/9 and derivatives
zypper	SLES
apt-get	Debian and Ubuntu

6 Open Client Library

The Open Client Library provides application interoperability with the Oracle Call Interface. An application that was formerly locked in can now work with either an EDB Postgres Advanced Server or an Oracle database with minimal to no changes to the application code.

The following diagram compares the Open Client Library and Oracle Call Interface application stacks.



The EDB implementation of the Open Client Library is written in C.

6.1 Forming a connection string

The EDB OCL Connector accepts both Oracle-style and Postgres-style connection URIs. A connection string can take the following Oracle-style form:

```
[//][host][:port][/dbname]
```

Or it can take the following Postgres-style forms:

```
postgres://[user[:password]@][host][:port][/dbname]
[?param1=value1&...]
```

```
postgresql://[user[:password]@][host][:port][/dbname]
[?param1=value1&...]
```

You can also use a Postgres-style URI to specify multiple host components, each with an optional port component, in a single URI. A multi-host connection string takes the form:

```
postgresql://<user>:<password>@host1:port1,host2:port2,host3:port3/
```

Where:

`user` is the name of the connecting user.

`password` is the password associated with the connecting user.

`host` is the host name or IP address to which you're connecting. To specify an IPV6 address, enclose the address in square brackets.

`port` is the port number to which you're connecting.

`dbname` is the name of the database with which you're connecting.

`paramx=valuex` pairs specify extra, application-specific connection properties.

For example, each of the following connection strings establishes a connection to the `edb` database on port `5444` of a system with an IP address of `10.0.0.4`:

```
//10.0.0.4:5444/edb
postgres://<user>:<password>@10.0.0.4:5444/edb
postgresql://<user>:<password>@10.0.0.4:5444/edb
```

For more information about using Postgres-style connection strings, see the PostgreSQL core documentation.

6.2 Compiling and linking a program

The EDB Open Client Library allows applications written using the Oracle Call Interface API to connect to and access an EDB database with minimal changes to the C source code. The EDB Open Client Library files are named:

- On Linux: `libedboci.so`
- On Windows: `edboci.dll`

The files are installed in the `oci/lib` subdirectory.

Compiling and linking a sample program

This example compiles and links the sample program `edb_demo.c` in a Linux environment. The `edb_demo.c` file is located in the `oci/samples` subdirectory.

1. Set `ORACLE_HOME` to the complete pathname of the Oracle home directory, for example:

   ```
   export ORACLE_HOME=/usr/lib/oracle/xe/app/oracle/product/10.2.0/server
   ```

2. Set `EDB_HOME` to the complete pathname of the home directory, for example:

   ```
   export EDB_HOME=/usr/edb
   ```

3. Set `LD_LIBRARY_PATH` to the complete path of `libpthread.so`. By default, `libpthread.so` is located in `/lib64`.

   ```
   export LD_LIBRARY_PATH=/lib64/lib:$LD_LIBRARY_PATH
   ```

4. Set `LD_LIBRARY_PATH` to include the EDB Postgres Advanced Server Open Client library. By default, `libedboci.so` is located in `$EDB_HOME/oci/lib`.

   ```
   export LD_LIBRARY_PATH=$EDB_HOME/oci:$EDB_HOME/oci/lib:$LD_LIBRARY_PATH
   ```

5. Compile and link the OCL API program:

   ```
   cd $EDB_HOME/oci/samples
   make
   ```

6.3 Ref cursor support

The EDB Postgres Advanced Server Open Client Library supports the use of `REF CURSOR` as `OUT` parameters in PL/SQL procedures that are compatible with Oracle. Support is provided through the following APIs:

- `OCIBindByName`
- `OCIBindByPos`
- `OCIBindDynamic`
- `OCIStmtPrepare`
- `OCIStmtExecute`
- `OCIStmtFetch`
- `OCIAttrGet`

The EDB OCL Connector also supports the `SQLT_RSET` data type.

This example invokes a stored procedure that opens a cursor and returns a `REF CURSOR` as an output parameter. The code sample assumes that a PL/SQL procedure named `openCursor`, with an `OUT` parameter of type `REF CURSOR`, was created on the database server and that the required handles were allocated:

```
char* openCursor = "begin
\
     openCursor(:cmdRefCursor);
\
   end;";
OCIStmt* stmtOpenRefCursor;
OCIStmt*
stmtUseRefCursor;
```

Allocate handles for executing a stored procedure to open and use the `REF CURSOR`:

```
/* Handle for the stored procedure to open the ref cursor
*/
OCIHandleAlloc((dvoid *) envhp,
               (dvoid **) &stmtOpenRefCursor,

OCI_HTYPE_STMT,
               0,
               (dvoid **) NULL));
```

```
/* Handle for using the Ref Cursor
*/
OCIHandleAlloc((dvoid *) envhp,
               (dvoid **) &stmtUseRefCursor,

OCI_HTYPE_STMT,
               0,
               (dvoid **) NULL));
```

Then, prepare the PL/SQL block that's used to open the `REF CURSOR`:

```
OCIStmtPrepare(stmtOpenRefCursor,
                errhp,
                (text *) openCursor,
                (ub4)
strlen(openCursor),

OCI_NTV_SYNTAX,

OCI_DEFAULT));
```

Bind the PL/SQL `openCursor OUT` parameter:

```
OCIBindByPos(stmtOpenRefCursor,
                &bndplrc1,
                errhp,
                1,
                (dvoid*) &stmtUseRefCursor,
                        /* the returned ref cursor
*/
                0,
                SQLT_RSET,
                     /* SQLT_RSET type representing cursor
*/
                (dvoid *) 0,
                (ub2 *) 0,
                (ub2) 0,
                (ub4) 0,
                (ub4 *) 0,

OCI_DEFAULT));
```

Use the `stmtOpenRefCursor` statement handle to call the `openCursor` procedure:

```
OCIStmtExecute(svchp,
                stmtOpenRefCursor,
                errhp,
                1,
                0,
                0,
                0,

OCI_DEFAULT);
```

At this point, the `stmtUseRefCursor` statement handle contains the reference to the cursor. To obtain the information, define output variables for the ref cursor:

```
/* Define the output variables for the ref cursor
*/

OCIDefineByPos(stmtUseRefCursor,
                   &defnEmpNo,
                   errhp,
                  (ub4) 1,
                  (dvoid *) &empNo,
                  (sb4) sizeof(empNo),

SQLT_INT,
                  (dvoid *) 0,
                  (ub2 *)0,
                  (ub2 *)0,
                  (ub4)
OCI_DEFAULT));
```

Then, fetch the first row of the result set into the target variables:

```
/* Fetch the cursor data
*/
  OCIStmtFetch(stmtUseRefCursor,
                  errhp,
                  (ub4) 1,
                  (ub4)
OCI_FETCH_NEXT,
                  (ub4)
OCI_DEFAULT))
```

6.4 OCL function reference

The following tables list the functions supported by the EDB OCL connector. You must supply any header files. EDB Postgres Advanced Server doesn't supply header files.

Connect, authorize, and initialize functions

Function	Description
OCIBreak	Abort the specified OCL function.
OCIEnvCreate	Create an OCL environment.
OCIEnvInit	Initialize an OCL environment handle.
OCIInitialize	Initialize the OCL environment.
OCILogoff	Release a session.
OCILogon	Create a logon connection.
OCILogon2	Create a logon session in various modes.
OCIReset	Reset the current operation/protocol.
OCIServerAttach	Establish an access path to a data source.
OCIServerDetach	Remove access to a data source.
OCISessionBegin	Create a user session.
OCISessionEnd	End a user session.
OCISessionGet	Get session from session pool.
OCISessionRelease	Release a session.
OCITerminate	Detach from shared memory subsystem.

Using the tnsnames.ora file

The `OCIServerAttach` and `OCILogon` methods use `NET_SERVICE_NAME` as a connection descriptor specified in the `dblink` parameter of the `tnsnames.ora` file. Use the `tnsnames.ora` file (compatible with Oracle databases) to specify database connection details. OCL searches your home directory for a file named `.tnsnames.ora`. If OCL doesn't find the `.tnsnames.ora` file there, it searches for `tnsnames.ora` on the path specified in `TNS_ADMIN` environment variable.

You can specify multiple descriptors `(NET_SERVICE_NAME)` in the `tnsnames.ora` file.

The sample `tnsnames.ora` file contains:

```
EDBX =
(DESCRIPTION =
 (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 5444))
 (CONNECT_DATA = (SERVER = DEDICATED)(SID = edb))
)
```

Any parameters not included in the files are ignored by the Open Client Library. In the example, `SID` refers to the database named `edb` in the cluster running on `localhost` on port `5444`.

A C program call to `OCIServerAttach` that uses the `tnsnames.ora` file looks like:

```
static text* username =
(text*)"enterprisedb";
static text* password =
(text*)"edb";
static text* attach_str = "EDBX";
OCIServerAttach(srvhp,
                errhp,
                attach_str,

strlen(attach_str),
                0);
```

If you don't have a `tnsnames.ora` file, supply the connection string in the form `//localhost:5444/edbx`.

> **Note**
>
> Multiple descriptors are also supported in `tnsnames.ora`.

Handle and descriptor functions

Function	Description
OCIAttrGet	Get handle attributes. EDB Postgres Advanced Server supports the following handle attributes: OCI_ATTR_USERNAME, OCI_ATTR_PASSWORD, OCI_ATTR_SERVER, OCI_ATTR_ENV, OCI_ATTR_SESSION, OCI_ATTR_ROW_COUNT, OCI_ATTR_CHARSET_FORM, OCI_ATTR_CHARSET_ID, EDB_ATTR_STMT_LEVEL_TX, OCI_ATTR_MODULE
OCIAttrSet	Set handle attributes. EDB Postgres Advanced Server supports the following handle attributes: OCI_ATTR_USERNAME, OCI_ATTR_PASSWORD, OCI_ATTR_SERVER, OCI_ATTR_ENV, OCI_ATTR_SESSION, OCI_ATTR_ROW_COUNT, OCI_ATTR_CHARSET_FORM, OCI_ATTR_CHARSET_ID, EDB_ATTR_STMT_LEVEL_TX, OCI_ATTR_MODULE, OCI_ATTR_PREFETCH_ROWS
OCIDescriptorAlloc	Allocate and initialize a descriptor.
OCIDescriptorFree	Free an allocated descriptor.
OCIHandleAlloc	Allocate and initialize a handle.
OCIHandleFree	Free an allocated handle.
OCIParamGet	Get a parameter descriptor.
OCIParamSet	Set a parameter descriptor.

EDB_ATTR_EMPTY_STRINGS

By default, EDB Postgres Advanced Server treats an empty string as a NULL value. You can use the `EDB_ATTR_EMPTY_STRINGS` environment attribute to control the behavior of the OCL connector when mapping empty strings. To modify the mapping behavior, use the `OCIAttrSet()` function to set `EDB_ATTR_EMPTY_STRINGS` to one of the following.

Value	Description
OCI_DEFAULT	Treat an empty string as a NULL value.
EDB_EMPTY_STRINGS_NULL	Treat an empty string as a NULL value.
EDB_EMPTY_STRINGS_EMPTY	Treat an empty string as a string of zero length.

To find the value of `EDB_ATTR_EMPTY_STRINGS`, query `OCIAttrGet()`.

EDB_ATTR_HOLDABLE

EDB Postgres Advanced Server supports statements that execute as `WITH HOLD` cursors. The `EDB_ATTR_HOLDABLE` attribute specifies the statements that execute as `WITH HOLD` cursors. You can set the `EDB_ATTR_HOLDABLE` attribute to any of the following values:

- `EDB_WITH_HOLD` — Execute as a `WITH HOLD` cursor.
- `EDB_WITHOUT_HOLD` — Execute using a protocol-level prepared statement.
- `OCI_DEFAULT` — See the definition that follows.

You can set the attribute in an `OCIStmt` handle or an `OCIServer` handle. When you create an `OCIServer` handle or an `OCIStmt` handle, the `EDB_ATTR_HOLDABLE` attribute for that handle is set to `OCI_DEFAULT`.

You can change the `EDB_ATTR_HOLDABLE` attribute for a handle by calling `OCIAttrSet()` and retrieve the attribute by calling `OCIAttrGet()`.

When EDB Postgres Advanced Server executes a `SELECT` statement, it examines the `EDB_ATTR_HOLDABLE` attribute in the `OCIServer` handle. If that attribute is set to `EDB_WITH_HOLD`, the query is executed as a `WITH HOLD` cursor.

If the `EDB_ATTR_HOLDABLE` attribute in the `OCIServer` handle is set to `EDB_WITHOUT_HOLD`, the query is executed as a normal prepared statement.

If the `EDB_ATTR_HOLDABLE` attribute in the `OCIServer` handle is set to `OCI_DEFAULT`, EDB Postgres Advanced Server uses the value of the `EDB_ATTR_HOLDABLE` attribute in the `OCIServer` handle. (If the `EDB_ATTR_HOLDABLE` attribute in the `OCIServer` is set to `EDB_WITH_HOLD`, the query executes as a `WITH HOLD` cursor. Otherwise, the query executes as a protocol-prepared statement.)

EDB_HOLD_CURSOR_ACTION

The `EDB_HOLD_CURSOR_ACTION` attribute alters the way `WITH HOLD` cursors are created using the OCL interface. You can set this attribute to any of the following values:

- `EDB_COMMIT_AFTER_CURSOR` — Commit the transaction after creating the cursor.
- `EDB_CURSOR_WITHOUT_XACT_BLK` — Don't begin a new transaction chain.
- `OCI_DEFAULT` — See the definition that follows.

The following describes the attribute values.

`OCI_DEFAULT`

Each time you execute a statement, the OCL examines the transaction state on the database server. If a transaction isn't already in progress, the OCL executes a `BEGIN` statement to create a new transaction block and then executes the statement that you provide. The transaction block remains open until you call `OCITransCommit()` or `OCITransRollback()`.

By default, the database server closes any open cursors when you commit or roll back. If you (or the OCL) declare a cursor that includes the `WITH HOLD` clause, the cursor result set is persisted on the database server, and you can continue to fetch from that cursor. However, the database server doesn't persist open cursors when you roll back a transaction. If you try to fetch from a cursor after a `ROLLBACK`, the database server reports an error.

`EDB_COMMIT_AFTER_CURSOR`

If your application must read from a `WITH HOLD` cursor after rolling back a transaction, you can arrange for the OCL to commit the transaction immediately after creating the cursor by setting `EDB_HOLD_CURSOR_ACTION` to `EDB_COMMIT_AFTER_CURSOR` prior to creating such a cursor. For example:

```
ub4 action =
EDB_COMMIT_AFTER_CURSOR;

OCIAttrSet(stmt,

OCI_HTYPE_STMT,
           &action,
           sizeof(action),
           EDB_ATTR_HOLD_CURSOR_ACTION,

err);

OCIStmtExecute(...);
```

> **Note**
>
> Using `EDB_COMMIT_AFTER_CURSOR` commits any pending changes.

`EDB_CURSOR_WITHOUT_XACT_BLK`

If your application doesn't run properly with the extra commits added by `EDB_COMMIT_AFTER_CURSOR`, you can try setting `EDB_ATTR_HOLD_CURSOR_ACTION` to `EDB_CURSOR_WITHOUT_XACT_BLK`. With this action, the OCL doesn't begin a new transaction chain. If you create a `WITH HOLD` cursor immediately after committing or rolling back a transaction, the cursor is created in its own transaction, the database server commits that transaction, and the cursor persists.

You might still experience errors if the cursor declaration isn't the first statement in a transaction. If you execute some other statement before declaring the cursor, the `WITH HOLD` cursor is created in a transaction block and can be rolled back if an error occurs or if your application calls `OCITransRollback()`.

You can set the `EDB_HOLD_CURSOR_ACTION` on the server level (`OCIServer`) or for each statement handle (`OCIStmt`). If the statement attribute is set to a value other than `OCI_DEFAULT`, the value is derived from the statement handle. Otherwise, if the statement attribute is set to `OCI_DEFAULT`, the value is taken from the server handle. So you can define a server-wide default action by setting the attribute in the server handle and leaving the attribute set to `OCI_DEFAULT` in the statement handles. You can use different values for each statement handle or server handle as you see fit.

EDB_ATTR_STMT_LVL_TX

Unless otherwise instructed, the OCL connector rolls back the current transaction whenever the server reports an error. You can override the automatic `ROLLBACK` with the `edb_stmt_level_tx` parameter, which preserves modifications in a transaction, even if one or more statements raise an error in the transaction.

You can use the `OCIServer` attribute with `OCIAttrSet()` and `OCIAttrGet()` to enable or disable `EDB_ATTR_STMT_LEVEL_TX`. By default, `edb_stmt_level_tx` is disabled. To enable `edb_stmt_level_tx`, the client application must call `OCIAttrSet()`:

```
OCIServer* server =
myServer;
ub1 enabled = 1;

OCIAttrSet(server,

OCI_HTYPE_SERVER,
           &enabled,
           sizeof(enabled),
           EDB_ATTR_STMT_LEVEL_TX,

err);
```

To disable `edb_stmt_level_tx`:

```
OCIServer* server =
myServer;
ub1 enabled = 0;

OCIAttrSet(server,

OCI_HTYPE_SERVER,
           &enabled,
           sizeof(enabled),
           EDB_ATTR_STMT_LEVEL_TX,

err);
```

Bind, define, and describe functions

Function	Description
OCIBindByName	Bind by name.
OCIBindByPos	Bind by position.
OCIBindDynamic	Set additional attributes after bind.
OCIBindArrayOfStruct	Bind an array of structures for bulk operations.
OCIDefineArrayOfStruct	Specify the attributes of an array.
OCIDefineByPos	Define an output variable association.
OCIDefineDynamic	Set additional attributes for define.
OCIDescribeAny	Describe existing schema objects.
OCIStmtGetBindInfo	Get bind and indicator variable names and handle.
OCIUserCallbackRegister	Define a user-defined callback.

Statement functions

Function	Description
OCIStmtExecute	Execute a prepared SQL statement.
OCIStmtFetch	Fetch rows of data (deprecated).
OCIStmtFetch2	Fetch rows of data.
OCIStmtPrepare	Prepare a SQL statement.
OCIStmtPrepare2	Prepare a SQL statement.
OCIStmtRelease	Release a statement handle.

Transaction functions

Function	Description
OCITransCommit	Commit a transaction.
OCITransRollback	Roll back a transaction.

XA functions

Function	Description
xaoEnv	Return OCL environment handle.
xaoSvcCtx	Return OCL service context.

xaoSvcCtx

To use the `xaoSvcCtx` function, provide extensions in the `xaoSvcCtx` or `xa_open` connection string format as follows:

```
Oracle_XA{+<required_fields> ...}
```

Where `required_fields` are the following:

`HostName=host_ip_address` specifies the IP address of the EDB Postgres Advanced Server database.

`PortNumber=host_port_number` specifies the port number on which EDB Postgres Advanced Server is running.

`SqlNet=dbname` specifies the database name.

`Acc=P/username/password` specifies the database username and password. You can omit the password. To do so, specify the field as `Acc=P/username/` .

`AppName=app_id` specifies a number that identifies the application.

The following is an example of the connection string:

```
Oracle_XA+HostName=192.168.1.1+PortNumber=1533+SqlNet=XE+Acc=P/user/password+AppName=1234
```

Date and datetime functions

Function	Description
OCIDateAddDays	Add or subtract a number of days.
OCIDateAddMonths	Add or subtract a number of months.
OCIDateAssign	Assign a date.
OCIDateCheck	Check if the given date is valid.
OCIDateCompare	Compare two dates.
OCIDateDaysBetween	Find the number of days between two dates.

Function	Description
OCIDateFromText	Convert a string to a date.
OCIDateGetDate	Get the date portion of a date.
OCIDateGetTime	Get the time portion of a date.
OCIDateLastDay	Get the date of the last day of the month.
OCIDateNextDay	Get the date of the next day.
OCIDateSetDate	Set the date portion of a date.
OCIDateSetTime	Set the time portion of a date.
OCIDateSysDate	Get the current system date and time.
OCIDateToText	Convert a date to a string.
OCIDateTimeAssign	Perform datetime assignment.
OCIDateTimeCheck	Check if the date is valid.
OCIDateTimeCompare	Compare two datetime values.
OCIDateTimeConstruct	Construct a datetime descriptor.
OCIDateTimeConvert	Convert one datetime type to another.
OCIDateTimeFromArray	Convert an array of size `OCI_DT_ARRAYLEN` to an `OCIDateTime` descriptor.
OCIDateTimeFromText	Convert the given string to Oracle datetime type in the `OCIDateTime` descriptor according to the specified format.
OCIDateTimeGetDate	Get the date portion of a datetime value.
OCIDateTimeGetTime	Get the time portion of a datetime value.
OCIDateTimeGetTimeZoneName	Get the time zone name portion of a datetime value.
OCIDateTimeGetTimeZoneOffset	Get the time zone (hour, minute) portion of a datetime value.
OCIDateTimeSubtract	Take two datetime values as input and return their difference as an interval.
OCIDateTimeSysTimeStamp	Get the system current date and time as a timestamp with time zone.
OCIDateTimeToArray	Convert an `OCIDateTime` descriptor to an array.
OCIDateTimeToText	Convert the given date to a string according to the specified format.

Interval functions

Function	Description
OCIIntervalAdd	Add two interval values.
OCIIntervalAssign	Copy one interval value into another interval value.
OCIIntervalCompare	Compare two interval values.
OCIIntervalGetDaySecond	Extract days, hours, minutes, seconds and fractional seconds from an interval.
OCIIntervalSetDaySecond	Modify days, hours, minutes, seconds and fractional seconds in an interval.
OCIIntervalGetYearMonth	Extract year and month values from an interval.
OCIIntervalSetYearMonth	Modify year and month values in an interval.
OCIIntervalDivide	Divide `OCIInterval` values by `OCINumber` values.
OCIIntervalMultiply	Multiply `OCIInterval` values by `OCINumber` values.
OCIIntervalSubtract	Subtract one interval value from another interval value.
OCIIntervalToText	Extrapolate a character string from an interval.
OCIIntervalCheck	Verify the validity of an interval value.
OCIIntervalToNumber	Convert an `OCIInterval` value into a `OCINumber` value.

Function	Description
OCIIntervalFromNumber	Convert a `OCINumber` value into an `OCIInterval` value.
OCIDateTimeIntervalAdd	Add an `OCIInterval` value to an `OCIDatetime` value, resulting in an `OCIDatetime` value.
OCIDateTimeIntervalSub	Subtract an `OCIInterval` value from an `OCIDatetime` value, resulting in an `OCIDatetime` value.
OCIIntervalFromText	Convert a text string into an interval.
OCIIntervalFromTZ	Convert a time zone specification into an interval value.

Number functions

Function	Description
OCINumberAbs	Compute the absolute value.
OCINumberAdd	Adds NUMBERs.
OCINumberArcCos	Compute the arc cosine.
OCINumberArcSin	Compute the arc sine.
OCINumberArcTan	Compute the arc tangent.
OCINumberArcTan2	Compute the arc tangent of two NUMBERs.
OCINumberAssign	Assign one NUMBER to another.
OCINumberCeil	Compute the ceiling of NUMBER.
OCINumberCmp	Compare NUMBERs.
OCINumberCos	Compute the cosine.
OCINumberDec	Decrement a NUMBER.
OCINumberDiv	Divide two NUMBERs.
OCINumberExp	Raise e to the specified NUMBER power.
OCINumberFloor	Compute the floor of a NUMBER.
OCINumberFromInt	Convert an integer to an Oracle NUMBER.
OCINumberFromReal	Convert a real to an Oracle NUMBER.
OCINumberFromText	Convert a string to an Oracle NUMBER.
OCINumberHypCos	Compute the hyperbolic cosine.
OCINumberHypSin	Compute the hyperbolic sine.
OCINumberHypTan	Compute the hyperbolic tangent.
OCINumberInc	Increment a NUMBER.
OCINumberIntPower	Raise a given base to an integer power.
OCINumberIsInt	Test if a NUMBER is an integer.
OCINumberIsZero	Test if a NUMBER is zero.
OCINumberLn	Compute the natural logarithm.
OCINumberLog	Compute the logarithm to an arbitrary base.
OCINumberMod	Modulo division.
OCINumberMul	Multiply NUMBERs.
OCINumberNeg	Negate a NUMBER.
OCINumberPower	Exponentiation to base e.
OCINumberPrec	Round a NUMBER to a specified number of decimal places.
OCINumberRound	Round a NUMBER to a specified decimal place.
OCINumberSetPi	Initialize a NUMBER to Pi.

Function	Description
OCINumberSetZero	Initialize a NUMBER to zero.
OCINumberShift	Multiply by 10, shifting specified number of decimal places.
OCINumberSign	Obtain the sign of a NUMBER.
OCINumberSin	Compute the sine.
OCINumberSqrt	Compute the square root of a NUMBER.
OCINumberSub	Subtract NUMBERs.
OCINumberTan	Compute the tangent.
OCINumberToInt	Convert a NUMBER to an integer.
OCINumberToReal	Convert a NUMBER to a real.
OCINumberToRealArray	Convert an array of NUMBER to a real array.
OCINumberToText	Converts a NUMBER to a string.
OCINumberTrunc	Truncate a NUMBER at a specified decimal place.

String functions

Function	Description
OCIStringAllocSize	Get allocated size of string memory in bytes.
OCIStringAssign	Assign string to a string.
OCIStringAssignText	Assign text string to a string.
OCIStringPtr	Get string pointer.
OCIStringResize	Resize string memory.
OCIStringSize	Get string size.

Cartridge services and file I/O interface functions

Function	Description
OCIFileClose	Close an open file.
OCIFileExists	Test to see if the file exists.
OCIFileFlush	Write buffered data to a file.
OCIFileGetLength	Get the length of a file.
OCIFileInit	Initialize the `OCIFile` package.
OCIFileOpen	Open a file.
OCIFileRead	Read from a file into a buffer.
OCIFileSeek	Change the current position in a file.
OCIFileTerm	Terminate the `OCIFile` package.
OCIFileWrite	Write buflen bytes into the file.

LOB functions

Function	Description
OCILobRead	Return a LOB value (or a portion of a LOB value).
OCILOBWriteAppend	Add data to a LOB value.
OCILobGetLength	Return the length of a LOB value.
OCILobTrim	Trim data from the end of a LOB value.
OCILobOpen	Open a LOB value for use by other LOB functions.
OCILobClose	Close a LOB value.

Miscellaneous functions

Function	Description
OCIClientVersion	Return client library version.
OCIErrorGet	Return error message.
OCIPGErrorGet	Return native error messages reported by libpq or the server. The signature is: sword OCIPGErrorGet(dvoid *hndlp, ub4 recordno, OraText *errcodep,ub4 errbufsiz, OraText *bufp, ub4 bufsiz, ub4 type)
OCIPasswordChange	Change password.
OCIPing	Confirm that the connection and server are active.
OCIServerVersion	Get the Oracle version string.

Supported data types

Function	Description
ANSI_DATE	ANSI date
SQLT_AFC	ANSI fixed character
SQLT_AVC	ANSI variable character
SQLT_BDOUBLE	Binary double
SQLT_BIN	Binary data
SQLT_BFLOAT	Binary float
SQLT_BOL	Boolean
SQLT_CHR	Character string
SQLT_DAT	Oracle date
SQLT_DATE	ANSI date
SQLT_FLT	Float
SQLT_INT	Integer
SQLT_LBI	Long binary
SQLT_LNG	Long
SQLT_LVB	Longer long binary
SQLT_LVC	Longer longs (character)
SQLT_NUM	Oracle numeric
SQLT_ODT	OCL date type

Function	Description
SQLT_STR	Zero-terminated string
SQLT_TIMESTAMP	Timestamp
SQLT_TIMESTAMP_TZ	Timestamp with time zone
SQLT_TIMESTAMP_LTZ	Timestamp with local time zone
SQLT_UIN	Unsigned integer
SQLT_VBI	VCS format binary
SQLT_VCS	Variable character
SQLT_VNU	Number with preceding length byte
SQLT_VST	OCL string type

6.5 OCL error codes (reference)

The following table lists the error code mappings defined by the OCL Connector. When the database server reports an error code or condition (shown in the first or second column), the OCL converts the value to the compatible value displayed in the third column.

Error code	Condition name	Oracle error code
42601	syntax_error	ORA-16945
42P01	undefined_table	ORA-00942
02000	no_data	ORA-01403
08000	connection_exception	ORA-12545
08003	connection_does_not_exist	ORA-12545
08006	connection_failure	ORA-12545
08001	sqlclient_unable_to_establish_sqlconnection	ORA-12545
08004	sqlserver_rejected_establishment_of_sqlconnection	ORA-12545
25000	invalid_transaction_state	ORA-01453
08007	transaction_resolution_unknown	ORA-01453
0A000	feature_not_supported	ORA-03001
22012	division_by_zero	ORA-01476
2200B	escape_character_conflict	ORA-01424
22019	invalid_escape_character	ORA-00911
2200D	invalid_escape_octet	ORA-01424
22025	invalid_escape_sequence	ORA-01424
22P06	nonstandard_use_of_escape_character	ORA-01424
2200C	invalid_use_of_escape_character	ORA-01424
22004	null_value_not_allowed	ORA-01400
23000	integrity_constraint_violation	ORA-00001
23505	unique_violation	ORA-00001
40P01	t_r_deadlock_detected	ORA-00060
42701	duplicate_column	ORA-01430
53000	insufficient_resources	ORA-01659
53100	disk_full	ORA-01659
53200	out_of_memory	ORA-82100
42P07	duplicate_table	ORA-00955
21000	cardinality_violation	ORA-01427
22003	numeric_value_out_of_range	ORA-01426
22P02	invalid_text_representation	ORA-01858
28000	invalid_authorization_specification	ORA-01017
28P01	invalid_password	ORA-01017
2200F	zero_length_character_string	ORA-01425
42704	undefined_object	ORA-01418
2BP01	dependent_objects_still_exist	ORA-02429
22027	trim_error	ORA-30001
22001	string_data_right_truncation	ORA-01401
22002	null_value_no_indicator_parameter	ORA-01405
22008	datetime_field_overflow	ORA-01800

Error code	Condition name	Oracle error code
44000	with_check_option_violation	ORA-01402
01007	warning_privilege_not_granted	ORA-00000
01006	warning_privilege_not_revoked	ORA-00000
02001	no_additional_dynamic_result_sets_returned	ORA-00000
03000	sql_statement_not_yet_complete	ORA-00000
08P01	protocol_violation	ORA-00000
23001	restrict_violation	ORA-00000
23502	not_null_violation	ORA-00000
23505	foreign_key_violation	ORA-00000
23514	check_violation	ORA-00000
24000	invalid_cursor_state	ORA-01001
26000	invalid_sql_statement_name	ORA-00000
42830	invalid_foreign_key	ORA-00000
55006	object_in_use	ORA-00000
55P03	lock_not_available	ORA-00054
72000	snapshot_too_old	ORA-01555

For more information about Postgres error codes, see the PostgreSQL core documentation.

6.6 Multithreading support

OCL is supported in a multithreaded environment. You can enable and use multithreading in a multithreaded environment by making an `OCIEnvNlsCreate()` call with `OCI_THREADED` as the value of the mode parameter.

```
retCode = OCIEnvNlsCreate( &envp,
                           OCI_THREADED,
                           NULL,
                           NULL,
                           NULL,
                           NULL,
                           0,
                           NULL,
                           0,
                           0 );
```

All subsequent calls to `OCIEnvNlsCreate()` must also be made with `OCI_THREADED`.

OCI library manages mutexes for the application for each environment handle if a multithreaded application is running on a thread-safe operating system.

6.7 OTL support

Oracle Template Library (OTL) is a C++ library for database access. It consists of a single header file. To know more about OTL, see the Oracle, Odbc and DB2-CLI Template Library Programmer's Guide.

OTL certification

The EDB OCL Connector, version 13.1.4.2, is certified with OTL 4.0. To use OTL-supported data types and for other OTL-specific behavior, define the OTL environment variable (the value isn't important) on the shell before running an OTL-based app. For example, you can export `OTL=TRUE` for conditional execution of scenarios that are related to OTL.

EDB OCL Connector is certified with the following OTL features:

- Connect, disconnect, commit, and roll back using `otl_connect`.
- Constant SQL statements using the static function `otl_cursor::direct_exec`. (A SQL statement is constant if it doesn't have any bind variables.) It includes most DDL statements like `CREATE TABLE` and `CREATE PROCEDURE/FUNCTION`.
- SQL statements with bind variable using `otl_stream class`. It includes most DML statements like `SELECT`, `UPDATE`, `DELETE`, `INSERT`, and `PROCEDURE/FUNCTION` calls.
- Date/Time data types using `otl_datetime`.
- Raw/Long Raw data types using `otl_long_string`.
- Ref cursors using `otl_refcur_stream`.

Connect and log in

This example initializes OCL and connects to a database using a `tnsnames.ora`-based connection string:

```
otl_connect db;
otl_connect::otl_initialize();

db.rlogon("enterprisedb/edb@EDBX");
if(db.connected)
          cout<<"Connected to Database"<<endl;
```

CREATE TABLE, INSERT, and SELECT

This example uses `otl_cursor::direct_exec` to create a table and then insert a row in the table. You can then use `otl_stream` to retrieve the inserted row.

```
char* createstmt =
  "create table testtable(c1 VARCHAR2(15), c2
DATE)";
char* insertstmt =
  "insert into testtable values('test_data123',
"
  "TO_DATE('2005-12-31 23:59:59','YYYY-MM-DD
HH24:MI:SS'))";
char* selectstmt = "select c1, c2 from
testtable";

otl_cursor::direct_exec(db, createstmt); // create
table
db.commit();

otl_cursor::direct_exec(db, insertstmt); // Insert
data.

char strData[100];
otl_datetime
dtData;
otl_stream otlCur(50, sqlstmnt,
db);
while (!otlCur.eof()) {
  otlCur >> strData >>
dtData;

  cout << "Retrieved Value: " << data <<
endl;
  cout << "Retrieved Value: " << data.month <<
"/"
       << data.day << "/" << data.year << " " <<
data.hour
       << ":" << data.minute << ":" << data.second << endl;
}
```

UPDATE

This example uses bind parameters in an `UPDATE` statement:

```
char* updatestmt = "UPDATE testtable SET c1=:c1<char[49]>
"
                   "WHERE c1=:c2<char[49]>";

char whereValue[50] = "test_data123";
char data[50] = "otl test";
otl_stream otlCur(80, updatestmt,
db);
otlCur.set_commit(0);
otlCur << data << whereValue;
```

Stored procedure

This example creates a stored procedure using `otl_cursor::direct_exec` and then calls it using `otl_stream`:

```
otl_cursor::direct_exec(
  db,
  "CREATE OR REPLACE PROCEDURE my_procOneIntOut
"
  "  (A IN NUMBER, B OUT
NUMBER)"
  "IS
"
  "BEGIN
"
  "   B :=
A;"
  "END;");

otl_stream
otlCur(
  1,
  "begin my_procOneIntOut(:A<int,in>, :B<int,out>);end;",
  db);
otlCur.set_commit(0);

int a =
10;
otlCur <<
a;

int
b;
otlCur >>
b;
cout << "B: " << b <<
endl;
```

Function

This example creates a function using `otl_cursor::direct_exec` and then calls it using `otl_stream`:

> **Note**
>
> This example is using the `emp` table in the `edb` sample database.

```
otl_cursor::direct_exec(
  db,
  "CREATE OR REPLACE FUNCTION get_no_int(e_name character
"
  "varying(10))
"
  "RETURNS int AS $$
"
  "DECLARE retval int;
"

  "BEGIN
"
  "SELECT empno FROM emp WHERE ename = e_name INTO retval;
"
  "RETURN retval;
"
  "END;
"

  "$$  LANGUAGE
plpgsql;");

char ename[50] = "SCOTT";
otl_stream
otlCur(
  1,
  "begin
"
  " :rc<int,out> := get_no_int(:c1<char[11],in>);"
  "end;",
  db);
otlCur << ename;

int
eno;
otlCur >>
eno;

cout << "Retrieved Value: " << eno <<
endl;
```

REF CURSOR

This example creates a package with a procedure that returns three ref cursors as `OUT` parameters and then calls it.

> **Note**
>
> This example is using the `emp` table in the `edb` sample database.

```
otl_cursor::direct_exec(
  db,
  "CREATE OR REPLACE PACKAGE
ref_test
  IS TYPE p_cursor IS REF
CURSOR;
  PROCEDURE getdata(empc OUT
p_cursor,
                    salc OUT
p_cursor,
```

```
                    comc OUT
p_cursor);
  END
ref_test;
  "
);

otl_cursor::direct_exec(
  db,
  "CREATE OR REPLACE PACKAGE BODY ref_test
\
         IS
\
         PROCEDURE getdata(empc OUT p_cursor, salc OUT p_cursor, comc OUT p_cursor) IS
\
         BEGIN
\
            open empc for select empno, ename from EMP;
\
            open salc for select ename, sal from EMP;
\
            open comc for select ename, comm from EMP;
\
         END;
\
         END
ref_test;");

otl_stream
otlCur(1,
                  "BEGIN
\
        ref_test.getdata(:cur1<refcur,out[50]>, :cur2<refcur,out[50]>, :cur3<refcur,out[50]>);
\
        END;",
                  db);
otlCur.set_commit(0);

otl_refcur_stream
  s1; // reference cursor streams for reading
rows.
otl_refcur_stream
  s2; // reference cursor streams for reading
rows.
otl_refcur_stream
  s3; // reference cursor streams for reading
rows.

otlCur >> s1;
otlCur >> s2;
otlCur >> s3;

int e_no;
char name[11];
double
sal;
double comm;

cout << "=====> Reading :cur1..." << endl;
while (!s1.eof()) { // while not end-of-
data
  s1 >> e_no >> name;
  cout << "e_no=" << e_no << "\tname: " << name <<
endl;
```

```
}

cout << "=====> Reading :cur2..." << endl;
while (!s2.eof()) { // while not end-of-
data
  s2 >> name >>
sal;
  cout << "name=" << name << "\tsalary: " << sal <<
endl;
}

cout << "=====> Reading :cur3..." << endl;
while (!s3.eof()) { // while not end-of-
data
  s3 >> name >> comm;
  cout << "name=" << name << "\tcommission: " <<
comm
       << endl;
}

s1.close();
s2.close();
s3.close();
```

7 Generating the OCL trace

The OCL tracing option logs direct communication (queries, updates, and so on) with the backend in the specified `OCI_DEBUG_LOG file`. It also logs the functions/APIs that were invoked. The trace files are generated in the default working directory (`oci_log_file_name`). If you append the path with a file name (`directory path/oci_log_file_name`), then the trace files are generated at that specific location.

A trace file is generated for each connection in text-file (readable) format.

> **Note**
>
> OCL tracing is disabled by default.

To generate the OCL trace:

1. Enable the EDB client-side tracing for OCL. You can enable the OCL tracing by setting these environment variables:
    - `export OCI_DEBUG_LEVEL=4`
    - `export OCI_DEBUG_LOG=oci_log_file`
2. After you export the environment variables, run the application. The OCL trace files are generated in the specified directory.

8 Using SSL

EDB Postgres Advanced Server provides native support for using SSL connections to encrypt client/server communications for increased security. In OCL, it's controlled by setting the `sslmode` parameter to `verify-full` or `verify-ca` and providing the system with a root certificate to verify against.

Steps of SSL configuration

1. Configure the server and client-side certificates. For detailed information about configuring SSL client and server-side certificates, refer to the PostgreSQL SSL documentation.

2. Enable the SSL OCL connection. In an OCL client application, you can enable SSL mode by setting the `EDB_ATTR_SSL` attribute in `Session`:

```
char* sslmode = "verify-full";
retValue = OCIAttrSet((dvoid*)authp,
                      (ub4)OCI_HTYPE_SESSION,
                      (dvoid*)sslmode,
                      (ub4)strlen((char*)sslmode),
                      (ub4)EDB_ATTR_SSL,
                      errhp);
```

> **Note**
>
> `EDB_ATTR_SSL` is defined in the `edboci.h` header file available in the installation directory.

3. After setting the SSL attribute, you can use the `OCILogon` function to create a connection:

```
OCILogon(pEnv,pError,&pSvc,(OraText*)pUsername,ub4)UsernameLen,
        (OraText*)pPassword,(ub4)PasswordLen,
        (OraText*)pDatabase,(ub4)DatabaseLen);
```

Once the server is authenticated, then the client is ready to pass sensitive data.

For more information about the supported SSL mode options, see the PostgreSQL SSL documentation.

9 Scram compatibility

The EDB OCL driver provides SCRAM-SHA-256 support for EDB Postgres Advanced Server version 11 and later. This support is available in EDB OCL 11.0.1 release and later.