EDB OCL Connector

Version 13.1.4.2

Built at 2021-07-15T05:03:37

1 What's New

The following features are added to create the EDB OCL Connector `13.1.4.2`:

- EDB OCL now provides support for OTL 4.0 Certification. For more information, refer to OTL Support.
- EDB OCL installation always uses latest libpq. For more information, refer to libpq Cross-version Compatibility.

2 Supported Platforms

The EDB OCL Connector is certified with Advanced Server version 9.6 and above. The EDB OCL Connector native packages are supported on the following 64-bit platforms:

- Red Hat Enterprise Linux (x86_64) 7.x and 8.x
- CentOS (x86_64) 7.x and 8.x
- OL Linux 7.x and 8.x
- PPC-LE 8 running RHEL or CentOS 7.x
- SLES 12.x
- Debian 9.x and 10.x
- Ubuntu 18.04 and 20.04 LTS

The EDB OCL Connector graphical installers are supported on the following 64-bit Windows platforms:

- Windows Server 2019
- Windows Server 2016
- Windows Server 2012 R2
- Windows 10
- Windows 8.1

The EDB OCL Connector graphical installers are supported on the following 32-bit Windows platforms:

- Windows 10
- Windows 8.1

3 libpq Cross-version Compatibility

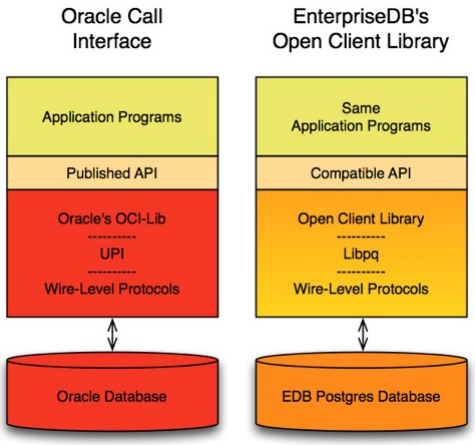
EDB OCL installation always uses the latest libpq. The different scenarios supported under libpq cross-version compatibility are as following:

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

4 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.



Comparison with Oracle Call Interface

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

4.1 Installing and Configuring the OCL Connector

You can use an RPM package, a native package, or a graphical installer to install or update the EDB OCL Connector.

Installing the Connector with an RPM Package

You can install the OCL Connector using an RPM package on the following platforms:

- RHEL 7
- RHEL 8
- CentOS 7
- CentOS 8

On RHEL 7

Before installing the OCL Connector, you must install the following prerequisite packages, and request credentials from EDB:

Install the `epel-release` package:

```
yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

Enable the optional, extras, and HA repositories:

```
subscription-manager repos --enable "rhel-*-optional-rpms" --enable "rhel-*-extras-rpms"  --enable "rhel-ha-for-rhel-*-server-rpms"
```

You must also have credentials that allow access to the EDB repository. For information about requesting credentials, visit:

https://info.enterprisedb.com/rs/069-ALB-339/images/Repository%20Access%2004-09-2019.pdf

After receiving your repository credentials you can:

1. Create the repository configuration file.
2. Modify the file, providing your user name and password.
3. Install `edb-oci`.

Creating a Repository Configuration File

To create the repository configuration file, assume superuser privileges, and invoke the following command:

```
yum -y install https://yum.enterprisedb.com/edbrepos/edb-repo-latest.noarch.rpm
```

The repository configuration file is named `edb.repo`. The file resides in `/etc/yum.repos.d`.

Modifying the file, providing your user name and password

After creating the `edb.repo` file, use your choice of editor to ensure that the value of the `enabled` parameter is `1`, and replace the `username` and `password` placeholders in the `baseurl` specification with the name and password of a registered EDB user.

```
[edb]
name=EnterpriseDB RPMs $releasever - $basearch
baseurl=https://<username>:<password>@yum.enterprisedb.com/edb/redhat/rhel-$releasever-$basearch
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/ENTERPRISEDB-GPG-KEY
```

Installing OCL Connector

After saving your changes to the configuration file, use the following commands to install the OCL Connector:

```
yum install edb-oci

yum install edb-oci-devel
```

When you install an RPM package that is signed by a source that is not recognized by your system, yum may ask for your permission to import the key to your local server. If prompted, and you are satisfied that the packages come from a trustworthy source, enter `y`, and press `Return` to continue.

During the installation, yum may encounter a dependency that it cannot resolve. If it does, it will provide a list of the required dependencies that you must manually resolve.

On RHEL 8

Before installing the OCL Connector, you must install the following prerequisite packages, and request credentials from EDB:

Install the `epel-release` package:

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

Enable the `codeready-builder-for-rhel-8-\*-rpms` repository:

```
ARCH=$( /bin/arch )
subscription-manager repos --enable "codeready-builder-for-rhel-8-${ARCH}-rpms"
```

You must also have credentials that allow access to the EDB repository. For information about requesting credentials, visit:

https://info.enterprisedb.com/rs/069-ALB-339/images/Repository%20Access%2004-09-2019.pdf

After receiving your repository credentials you can:

1. Create the repository configuration file.
2. Modify the file, providing your user name and password.
3. Install `edb-oci`.

Creating a Repository Configuration File

To create the repository configuration file, assume superuser privileges, and invoke the following command:

```
dnf -y https://yum.enterprisedb.com/edbrepos/edb-repo-latest.noarch.rpm
```

The repository configuration file is named `edb.repo`. The file resides in `/etc/yum.repos.d`.

Modifying the file, providing your user name and password

After creating the `edb.repo` file, use your choice of editor to ensure that the value of the `enabled` parameter is `1`, and replace the `username` and `password` placeholders in the `baseurl` specification with the name and password of a registered EDB user.

```
[edb]
name=EnterpriseDB RPMs $releasever - $basearch
baseurl=https://<username>:<password>@yum.enterprisedb.com/edb/redhat/rhel-
$releasever-$basearch
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/ENTERPRISEDB-GPG-KEY
```

Installing OCL Connector

After saving your changes to the configuration file, use the below command to install the OCL Connector:

```
dnf install edb-oci

dnf install edb-oci-devel
```

When you install an RPM package that is signed by a source that is not recognized by your system, yum may ask for your permission to import the key to your local server. If prompted, and you are satisfied that the packages come from a trustworthy source, enter `y`, and press `Return` to continue.

During the installation, yum may encounter a dependency that it cannot resolve. If it does, it will provide a list of the required dependencies that you must manually resolve.

On CentOS 7

Before installing the OCL Connector, you must install the following prerequisite packages, and request credentials from EDB:

Install the `epel-release` package:

```
yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-
7.noarch.rpm
```

!!! Note You may need to enable the `[extras]` repository definition in the `CentOS-Base.repo` file (located in `/etc/yum.repos.d`).

You must also have credentials that allow access to the EDB repository. For information about requesting credentials, visit:

https://info.enterprisedb.com/rs/069-ALB-339/images/Repository%20Access%2004-09-2019.pdf

After receiving your repository credentials you can:

1. Create the repository configuration file.
2. Modify the file, providing your user name and password.
3. Install `edb-oci`.

Creating a Repository Configuration File

To create the repository configuration file, assume superuser privileges, and invoke the following command:

```
yum -y install https://yum.enterprisedb.com/edbrepos/edb-repo-latest.noarch.rpm
```

The repository configuration file is named `edb.repo`. The file resides in `/etc/yum.repos.d`.

Modifying the file, providing your user name and password

After creating the `edb.repo` file, use your choice of editor to ensure that the value of the `enabled` parameter is `1`, and replace the `username` and `password` placeholders in the `baseurl` specification with the name and password of a registered EDB user.

```
[edb]
name=EnterpriseDB RPMs $releasever - $basearch
baseurl=https://<username>:<password>@yum.enterprisedb.com/edb/redhat/rhel-
$releasever-$basearch
enabled=1
gpgcheck=1
```

```
repo_gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/ENTERPRISEDB-GPG-KEY
```

Installing OCL Connector

After saving your changes to the configuration file, use the following command to install the OCL Connector:

```
yum install edb-oci

yum install edb-oci-devel
```

When you install an RPM package that is signed by a source that is not recognized by your system, yum may ask for your permission to import the key to your local server. If prompted, and you are satisfied that the packages come from a trustworthy source, enter `y`, and press `Return` to continue.

During the installation, yum may encounter a dependency that it cannot resolve. If it does, it will provide a list of the required dependencies that you must manually resolve.

On CentOS 8

Before installing the OCL Connector, you must install the following prerequisite packages, and request credentials from EDB:

Install the `epel-release` package:

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

Enable the `PowerTools` repository:

```
dnf config-manager --set-enabled PowerTools
```

You must also have credentials that allow access to the EDB repository. For information about requesting credentials, visit:

https://info.enterprisedb.com/rs/069-ALB-339/images/Repository%20Access%2004-09-2019.pdf

After receiving your repository credentials you can:

1. Create the repository configuration file.
2. Modify the file, providing your user name and password.
3. Install `edb-oci`.

Creating a Repository Configuration File

To create the repository configuration file, assume superuser privileges, and invoke the following command:

```
dnf -y install https://yum.enterprisedb.com/edbrepos/edb-repo-latest.noarch.rpm
```

The repository configuration file is named `edb.repo`. The file resides in `/etc/yum.repos.d`.

Modifying the file, providing your user name and password

After creating the `edb.repo` file, use your choice of editor to ensure that the value of the `enabled` parameter is `1`, and replace the `username` and `password` placeholders in the `baseurl` specification with the name and password

of a registered EDB user.

```
[edb]
name=EnterpriseDB RPMs $releasever - $basearch
baseurl=https://<username>:<password>@yum.enterprisedb.com/edb/redhat/rhel-
$releasever-$basearch
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/ENTERPRISEDB-GPG-KEY
```

Installing OCL Connector

After saving your changes to the configuration file, use the following command to install the OCL Connector:

```
dnf install edb-oci

dnf install edb-oci-devel
```

When you install an RPM package that is signed by a source that is not recognized by your system, yum may ask for your permission to import the key to your local server. If prompted, and you are satisfied that the packages come from a trustworthy source, enter `y`, and press `Return` to continue.

During the installation, yum may encounter a dependency that it cannot resolve. If it does, it will provide a list of the required dependencies that you must manually resolve.

Updating an RPM Installation

If you have an existing `OCL Connector` RPM installation, you can use yum or dnf to upgrade your repository configuration file and update to a more recent product version. To update the `edb.repo` file, assume superuser privileges and enter:

- On RHEL or CentOS 7:

  `yum upgrade edb-repo`

- On RHEL or CentOS 8:

  `dnf upgrade edb-repo`

yum or dnf will update the `edb.repo` file to enable access to the current EDB repository, configured to connect with the credentials specified in your `edb.repo` file. Then, you can use yum to upgrade any installed packages:

- On RHEL or CentOS 7:

  `yum upgrade edb-oci`

  `yum upgrade edb-oci-devel`

- On RHEL or CentOS 8:

  `dnf upgrade edb-oci`

  `dnf upgrade edb-oci-devel`

Installing the Connector on an SLES 12 Host

You can use the zypper package manager to install the connector on an SLES 12 host. zypper will attempt to satisfy package dependencies as it installs a package, but requires access to specific repositories that are not hosted at EDB. Before installing the connector, use the following commands to add EDB repository configuration files to your SLES host:

```
zypper addrepo https://zypp.enterprisedb.com/suse/edb-sles.repo
```

After creating the repository configuration files, use the `zypper refresh` command to refresh the metadata on your SLES host to include the EDB repositories.

When prompted for a `User Name` and `Password`, provide your connection credentials for the EDB repository. To request credentials for the repository, visit the EDB website.

Before installing EDB Postgres Advanced Server or supporting components, you must also add SUSEConnect and the SUSE Package Hub extension to the SLES host, and register the host with SUSE, allowing access to SUSE repositories. Use the commands:

```
zypper install SUSEConnect
SUSEConnect -r 'REGISTRATION_CODE' -e 'EMAIL'
SUSEConnect -p PackageHub/12.4/x86_64
SUSEConnect -p sle-sdk/12.4/x86_64
```

For detailed information about registering a SUSE host, visit the SUSE website.

Then, you can use the zypper utility to install the connector:

```
zypper install edb-oci
```

```
zypper install edb-oci-devel
```

Installing the Connector on a Debian or Ubuntu Host

To install a DEB package on a Debian or Ubuntu host, you must have credentials that allow access to the EDB repository. To request credentials for the repository, visit the EDB website.

The following steps will walk you through on using the EDB apt repository to install a DEB package. When using the commands, replace the `username` and `password` with the credentials provided by EDB.

1. Assume superuser privileges:

   ```
   sudo su -
   ```

2. Configure the EDB repository:

   On Debian 9:

   ```
   sh -c 'echo "deb https://username:password@apt.enterprisedb.com/$(lsb_release -
   cs)-edb/ $(lsb_release -cs) main" > /etc/apt/sources.list.d/edb-$(lsb_release -
   cs).list'
   ```

   On Debian 10:

   1. Set up the EDB repository:

```
sh -c 'echo "deb [arch=amd64] https://apt.enterprisedb.com/$(lsb_release -cs)-
edb/ $(lsb_release -cs) main" > /etc/apt/sources.list.d/edb-$(lsb_release -
cs).list'
```

2. Substitute your EDB credentials for the `username` and `password` in the following command:

```
sh -c 'echo "machine apt.enterprisedb.com login <username> password
<password>" > /etc/apt/auth.conf.d/edb.conf'
```

3. Add support to your system for secure APT repositories:

```
apt-get install apt-transport-https
```

4. Add the EDB signing key:

```
wget -q -O - https://<username>:<password>@apt.enterprisedb.com/edb-deb.gpg.key |
apt-key add -
```

5. Update the repository metadata:

```
apt-get update
```

6. Install DEB package:

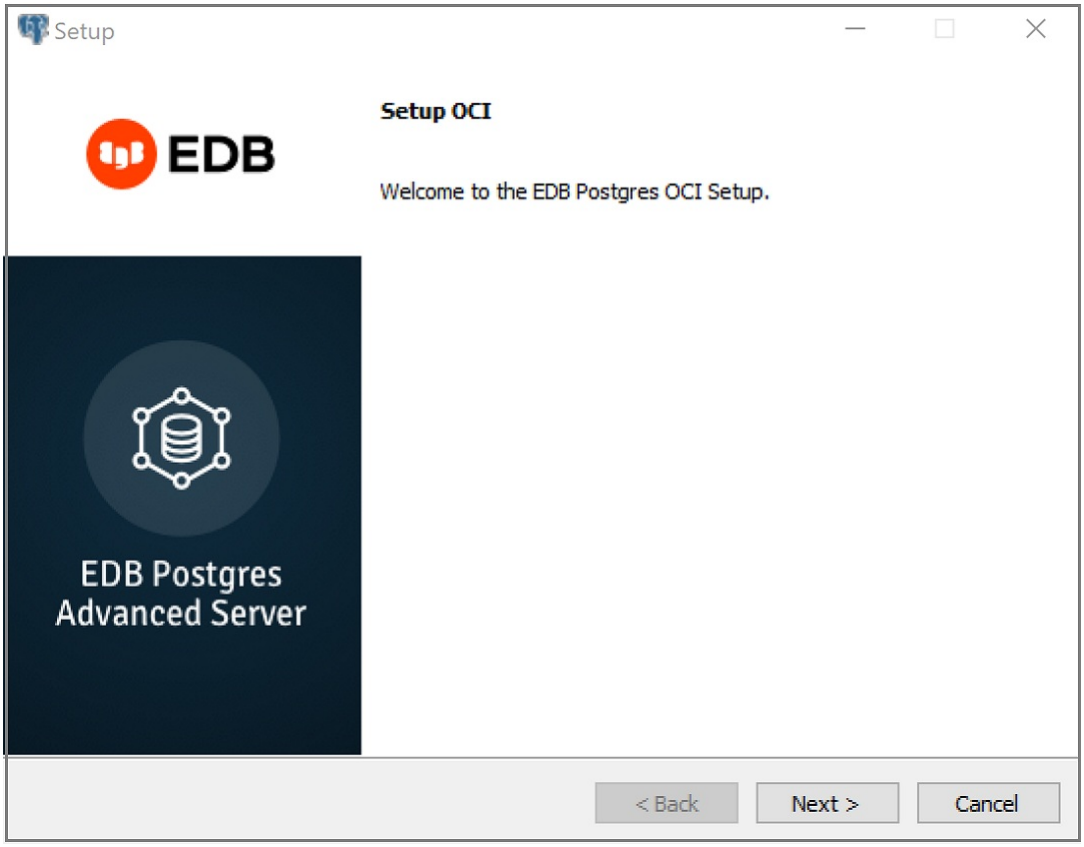
```
apt-get install edb-oci
apt-get install edb-oci-dev
```

Using the Graphical Installer to Install the Connector

You can use the EDB Connectors Installation wizard to add the EDB OCL connector to your system; the wizard is available at the EDB website.

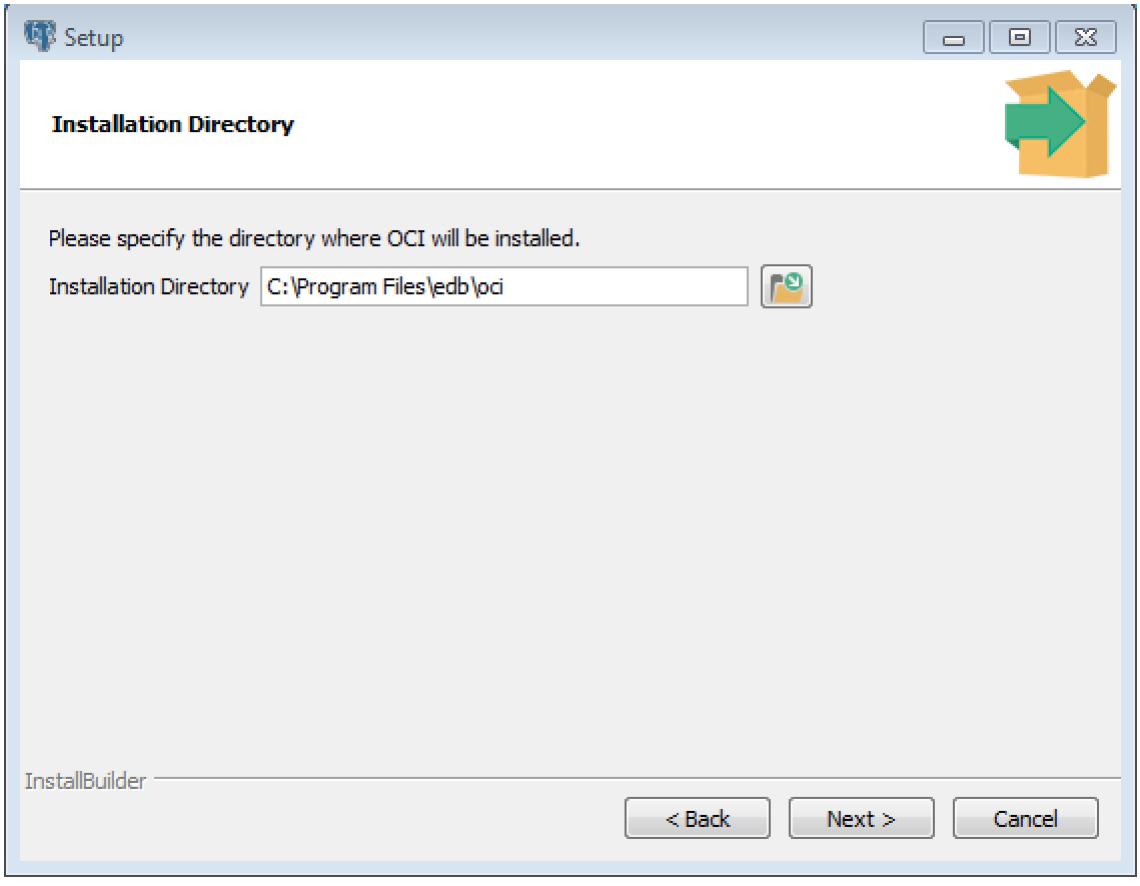
This section demonstrates using the Installation Wizard to install the Connectors on a Windows system. (Download the installer, and then, right-click on the installer icon, and select `Run As Administrator` from the context menu.)

When the `Language Selection` popup opens, select an installation language and click `OK` to continue to the `Setup` window.



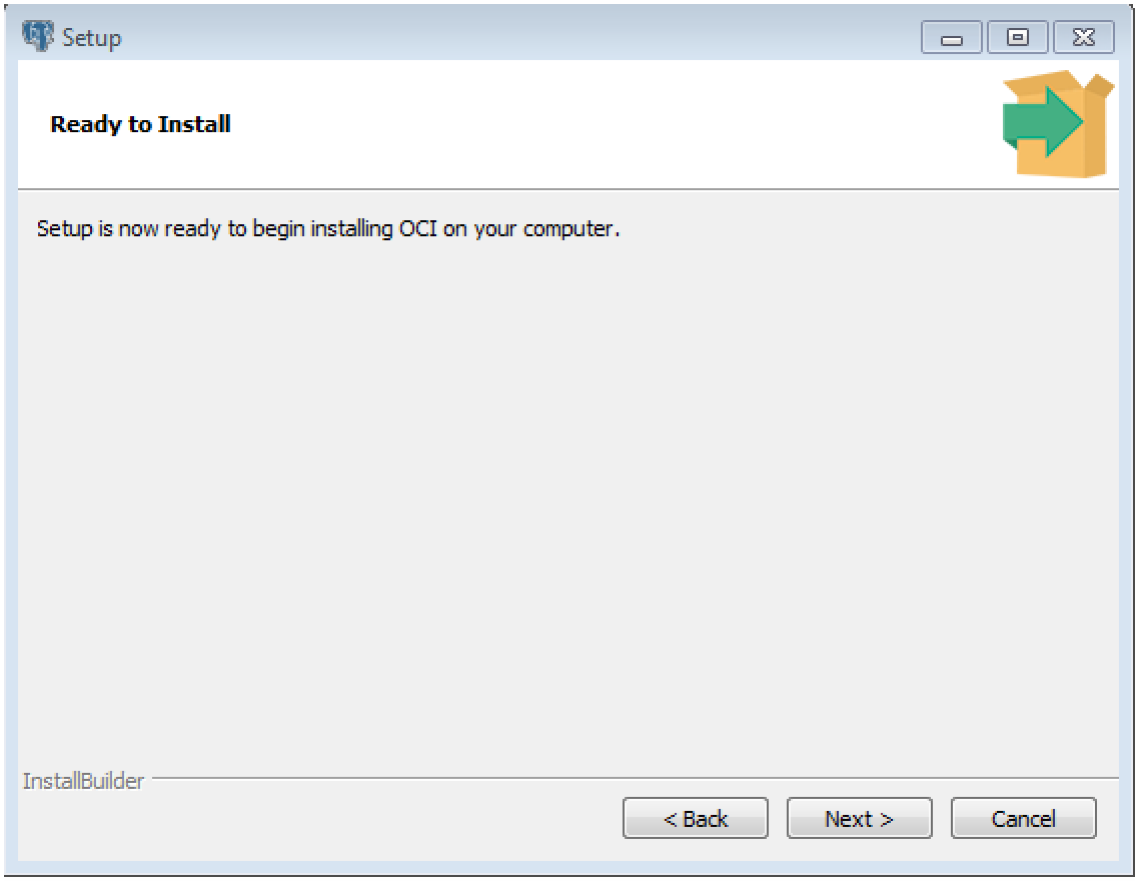
The OCL Connector Installation wizard

Click `Next` to continue.



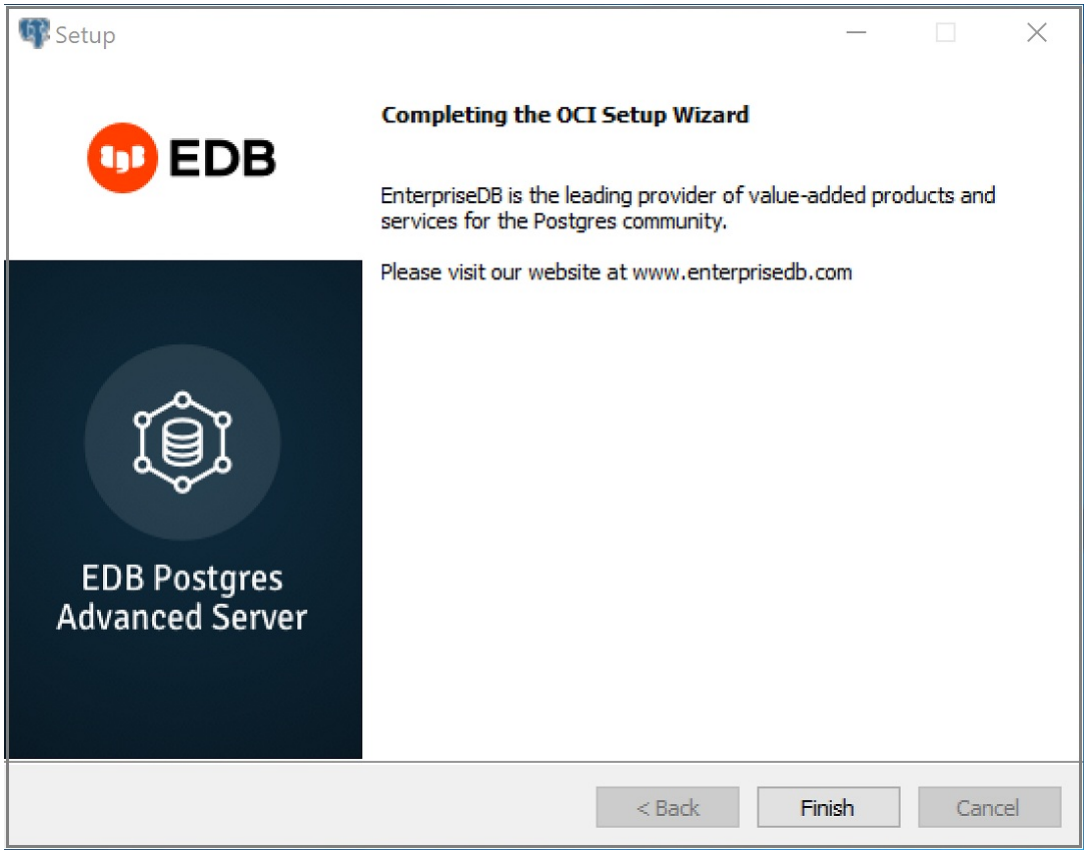
The Installation dialog

Use the `Installation Directory` dialog to specify the directory in which the connector will be installed, and click `Next` to continue.



The Ready to Install dialog

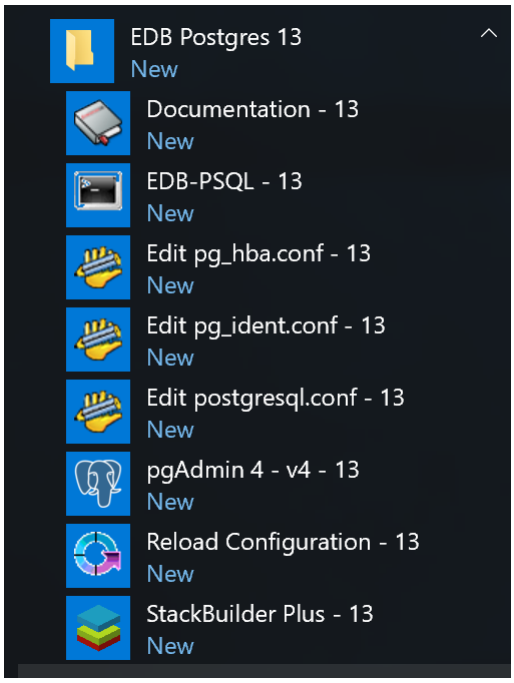
Click `Next` on the `Ready to Install` dialog to start the installation; popup dialogs confirm the progress of the installation wizard.



The installation is complete

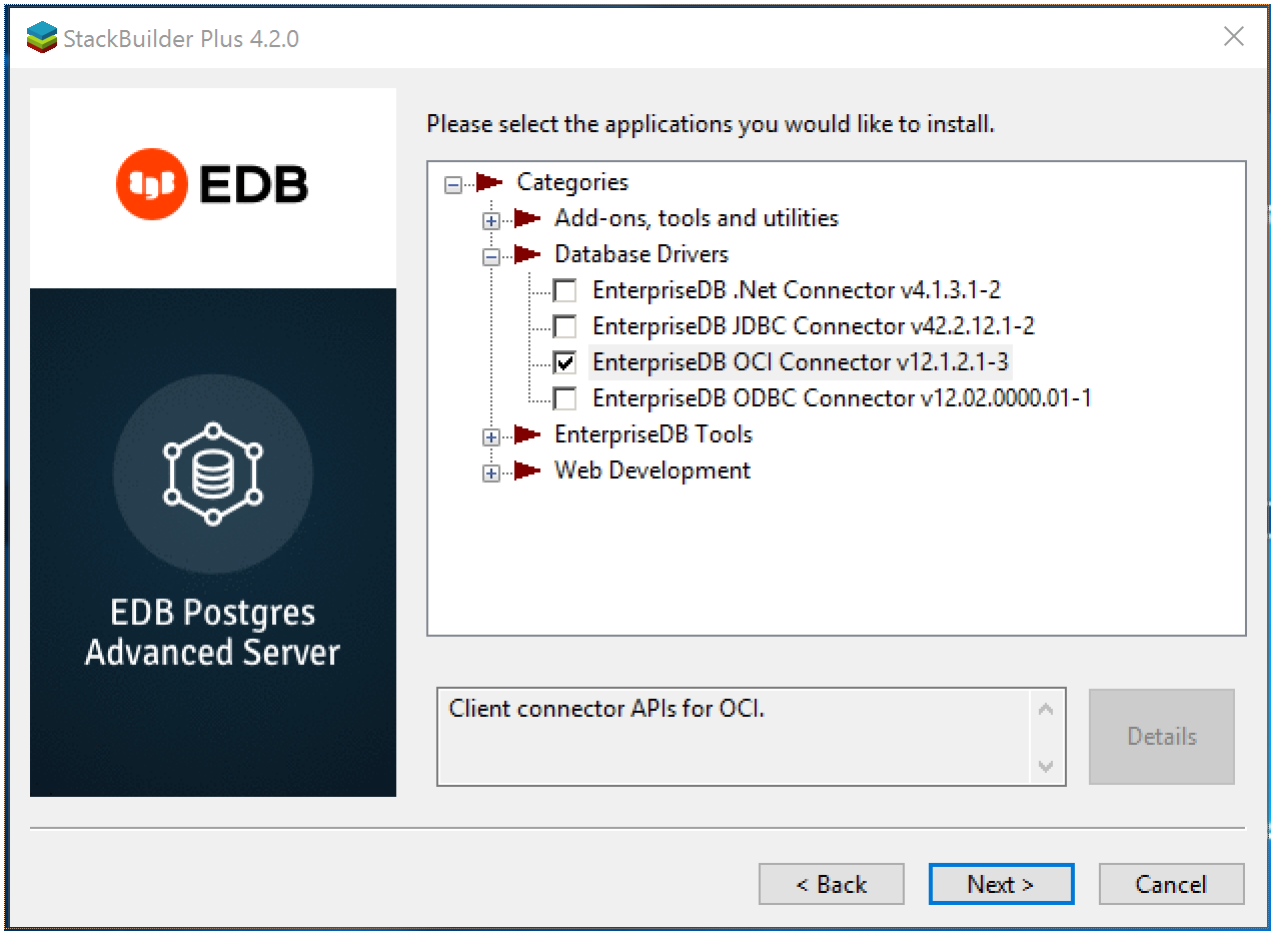
When the wizard informs you that it has completed the setup, click the `Finish` button to exit the dialog.

You can also use StackBuilder Plus to add or update the connector on an existing Advanced Server installation; to open StackBuilder Plus, select StackBuilder Plus from the Windows `Apps` menu.



Starting StackBuilder Plus

When StackBuilder Plus opens, follow the onscreen instructions. Select the `EnterpriseDB OCI Connector` option from the `Database Drivers` node of the tree control.



Selecting the Connectors installer

Follow the directions of the onscreen wizard to add or update an installation of the EDB Connectors.

4.2 Forming a Connection String

The EDB OCL connector accepts both Oracle-style and Postgres-style connection URI's. A connection string may take the following Oracle-style form:

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

or 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 are connecting; to specify an IPV6 address, enclose the address in square brackets.

`port` is the port number to which you are connecting.

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

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

For example, each of the following connection strings establish 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, please see the PostgreSQL core documentation, available at the EDB website.

4.3 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

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

1. Set the `ORACLE_HOME` and `EDB_HOME` environment variables.
2. 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
```

3. Set `EDB_HOME` to the complete pathname of the home directory.

For example:

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

4. 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
```

5. Set `LD_LIBRARY_PATH` to include the Advanced Server Open Client library. By default, `libiconv.so.2` is located in `$EDB_HOME/oci/lib`.

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

6. Then, compile and link the OCL API program.

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

```
make
```

4.4 Ref Cursor Support

The 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.

The following example demonstrates how to invoke 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`) has been created on the database server, and that the required handles have been 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 is 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))
```

4.5 OCL Function Reference

The following tables list the functions supported by the EDB OCL connector. Note that any and all header files must be supplied by the user. Advanced Server does not supply any such files.

Connect, Authorize and Initialize Functions

Function	Description
OCIBreak	Aborts the specified OCL function.
OCIEnvCreate	Creates an OCL environment.
OCIEnvInit	Initializes an OCL environment handle.
OCIInitialize	Initializes the OCL environment.
OCILogoff	Releases a session.
OCILogon	Creates a logon connection.
OCILogon2	Creates a logon session in various modes.
OCIReset	Resets the current operation/protocol.
OCIServerAttach	Establishes an access path to a data source.
OCIServerDetach	Removes access to a data source.
OCISessionBegin	Creates a user session.
OCISessionEnd	Ends a user session.
OCISessionGet	Gets session from session pool.
OCISessionRelease	Releases a session.

Function	Description
OCITerminate	Detaches from shared memory subsystem.

Using the tnsnames.ora File

The `OCIServerAttach` and `OCILogon` method uses `NET_SERVICE_NAME` as 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 the user's home directory for a file named `.tnsnames.ora`; if OCL doesn't find the `.tnsnames.ora` file in the user's home directory, it searches the `tnsnames.ora` on path specified in `TNS_ADMIN` environment variable.

Multiple descriptors `(NET_SERVICE_NAME)` can be specified in `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 the `localhost` on port `5444`.

A C program call to `OCIServerAttach` that uses the `tnsnames.ora` file will look 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. 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. 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.

Function	Description
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, Advanced Server will treat 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

Advanced Server supports statements that execute as `WITH HOLD` cursors. The `EDB_ATTR_HOLDABLE` attribute specifies which statements execute as `WITH HOLD` cursors. The `EDB_ATTR_HOLDABLE` attribute can be set to any of the following three 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 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`, 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` – do not 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 is not 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 rollback. 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 may continue to fetch from that cursor. However, the database server will not persist open cursors when you roll back a transaction. If you try to fetch from a cursor after a `ROLLBACK`, the database server will report 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( ... );
```

It is important to understand that using `EDB_COMMIT_AFTER_CURSOR` will commit any pending changes.

`EDB_CURSOR_WITHOUT_XACT_BLK`

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

It is important to understand that you may still experience errors if the cursor declaration is not the first statement within a transaction – if you execute some other statement before declaring the cursor, the `WITH HOLD` cursor will be created in a transaction block and may be rolled back if an error occurs (or if your application calls `OCITransRollback()`).

Please note that 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 will `ROLLBACK` the current transaction whenever the server reports an error. If you choose, you can override the automatic `ROLLBACK` with the `edb_stmt_level_tx` parameter, which preserves modifications within a transaction, even if one (or several) statements raise an error within 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	Returns OCL environment handle.
xaoSvcCtx	Returns OCL service context.

xaoSvcCtx

In order to use the xaoSvcCtx function, extensions in the `xaoSvcCtx` or `xa_open` connection string format must be provided as follows:

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

Where `required_fields` are the following:

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

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

`SqlNet=dbname` specifies the database name.

`Acc=P/username/password` specifies the database username and password. *password* may be omitted in which case the field is specified 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=
4
```

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	Adds two interval values.
OCIIntervalAssign	Copies one interval value into another interval value.
OCIIntervalCompare	Compares two interval values.
OCIIntervalGetDaySecond	Extracts days, hours, minutes, seconds and fractional seconds from an interval.
OCIIntervalSetDaySecond	Modifies days, hours, minutes, seconds and fractional seconds in an interval.
OCIIntervalGetYearMonth	Extracts year and month values from an interval.
OCIIntervalSetYearMonth	Modifies year and month values in an interval.
OCIIntervalDivide	Implements division of OCIInterval values by OCINumber values.
OCIIntervalMultiply	Implements multiplication of OCIInterval values by OCINumber values.
OCIIntervalSubtract	Subtracts one interval value from another interval value.
OCIIntervalToText	Extrapolates a character string from an interval.
OCIIntervalCheck	Verifies the validity of an interval value.
OCIIntervalToNumber	Converts an OCIInterval value into a OCINumber value.

Function	Description
OCIIntervalFromNumber	Converts a OCINumber value into an OCIInterval value.
OCIDateTimeIntervalAdd	Adds an OCIInterval value to an OCIDatetime value, resulting in an OCIDatetime value.
OCIDateTimeIntervalSub	Subtracts an OCIInterval value from an OCIDatetime value, resulting in an OCIDatetime value.
OCIIntervalFromText	Converts a text string into an interval.
OCIIntervalFromTZ	Converts 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	Increments 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.

Function	Description
OCINumberSetPi	Initialize a NUMBER to Pi.
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

Function	Description
OCILobRead	Returns a LOB value (or a portion of a LOB value).
OCILOBWriteAppend	Adds data to a LOB value.
OCILobGetLength	Returns the length of a LOB value.
OCILobTrim	Trims data from the end of a LOB value.
OCILobOpen	Opens a LOB value for use by other LOB functions.
OCILobClose	Closes 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_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
SQLT_STR	Zero-terminated string
SQLT_TIMESTAMP	Timestamp

Function	Description
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

4.6 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

Error Code	Condition Name	Oracle Error Code
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
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, please see the PostgreSQL core documentation.

4.7 Multithreading Support

OCL is supported in multithreaded environment. You can enable/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.

4.8 OTL Support

OTL (Oracle Template Library) is a C++ library for database access. It consists of a single header file. To know more about OTL, visit:

http://otl.sourceforge.net/

OTL Certification

EDB OCL Connector, version 13.1.4.2 is certified with OTL 4.0. To use OTL supported datatypes and for other OTL specific behaviour, OTL environment variable should be defined on the shell before running OTL based app. The value of OTL is not important, just it should be defined. For example: You can export OTL=TRUE for conditional execution of scenarios which are related to OTL.

EDB OCL Connector is certified with the following OTL features:

- Connect, disconnect, commit and rollback using `otl_connect`.
- Constant SQL Statements (A SQL statement is constant if it does not have any bind variables) using static function `otl_cursor::direct_exec`. 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`.

Examples

Connect and Login

The following code demonstrates how to initialize OCL and connect to a database using `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

The following code demonstrates the use of `otl_cursor::direct_exec` to create a table and then insert a row in this 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

The following code demonstrates the use of 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

The following code demonstrates how to create a stored procedure using `otl_cursor::direct_exec` and then call 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

The following code demonstrates how to create a function using `otl_cursor::direct_exec` and then call it using `otl_stream`:

!!! Note This example is using `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

The following code demonstrates how to create a package with a procedure that returns three REF CURSORs as `OUT` parameters and then calls it:

!!! Note This example is using `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();
```

5 Generating the OCL Trace

The OCL tracing option logs direct communication (queries, updates, etc.) with the backend in specified `OCI_DEBUG_LOG file`. In addition, 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 specific location.

A tracefile 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 below environment variables:

```
export OCI_DEBUG_LEVEL=4
```

```
export OCI_DEBUG_LOG=oci_log_file
```

2. Once you have exported the environment variables, execute the application. The OCL trace files are generated in the specified directory.

6 Using SSL

EDB Postgres Advanced Server provides native support for using SSL connections to encrypt client/server communications for increased security. In OCL, it is 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 `edboci.h` header file available in installation directory.

3. After setting 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, please see:

https://www.postgresql.org/docs/13/libpq-ssl.html#LIBPQ-SSL-SSLMODE-STATEMENTS

---

7 Scram Compatibility

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