Rocket UniVerse

External Database Access (EDA)

Version 11.3.1

October 2016
UNV-1131-EDA-1
Corporate information

Rocket Software, Inc. develops enterprise infrastructure products in four key areas: storage, networks, and compliance; database servers and tools; business information and analytics; and application development, integration, and modernization.

Website: www.rocketsoftware.com

Rocket Global Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA

To contact Rocket Software by telephone for any reason, including obtaining pre-sales information and technical support, use one of the following telephone numbers.

<table>
<thead>
<tr>
<th>Country</th>
<th>Toll-free telephone number</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td>1-855-577-4323</td>
</tr>
<tr>
<td>Australia</td>
<td>1-800-823-405</td>
</tr>
<tr>
<td>Belgium</td>
<td>0800-266-65</td>
</tr>
<tr>
<td>Canada</td>
<td>1-855-577-4323</td>
</tr>
<tr>
<td>China</td>
<td>800-720-1170</td>
</tr>
<tr>
<td>France</td>
<td>08-05-08-05-62</td>
</tr>
<tr>
<td>Germany</td>
<td>0800-180-0882</td>
</tr>
<tr>
<td>Italy</td>
<td>800-878-295</td>
</tr>
<tr>
<td>Japan</td>
<td>0800-170-5464</td>
</tr>
<tr>
<td>Netherlands</td>
<td>0-800-022-2961</td>
</tr>
<tr>
<td>New Zealand</td>
<td>0800-003210</td>
</tr>
<tr>
<td>South Africa</td>
<td>0-800-980-818</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>0800-520-0439</td>
</tr>
</tbody>
</table>

Contacting Technical Support

The Rocket Customer Portal is the primary method of obtaining support. If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Customer Portal and report a problem, download an update, or read answers to FAQs. To log in to the Rocket Customer Portal or to request a Rocket Customer Portal account, go to www.rocketsoftware.com/support.

In addition to using the Rocket Customer Portal to obtain support, you can use one of the telephone numbers that are listed above or send an email to support@rocketsoftware.com.
Contents

Notices............................................................................................................................................................................2
Corporate information............................................................................................................................................................3
Chapter 1: External Database Access............................................................................................................................7
  First Normal Form (1NF) vs. Non-First Normal Form.................................................................................................7
  Table concepts.................................................................................................................................................................7
  Representing multivalues..............................................................................................................................................7
  Primary and foreign keys..............................................................................................................................................8
  Associations.................................................................................................................................................................8
  Virtual attributes and I-descriptors............................................................................................................................8
  Mapping example.........................................................................................................................................................9
Chapter 2: EDA Schema Manager................................................................................................................................11
  Starting the EDA Schema Manager.............................................................................................................................11
  Creating a new U2 server connection.........................................................................................................................12
  Connecting to the U2 server....................................................................................................................................14
  Managing connections................................................................................................................................................14
  About data sources..................................................................................................................................................14
    Connecting to SQL server, Oracle, or IBM DB2.....................................................................................................14
    Connecting to Microsoft SQL server using the Native Client...............................................................................16
  Defining a data source..............................................................................................................................................16
  Creating EDA schemas..............................................................................................................................................18
    Defining attribute details...................................................................................................................................21
    Defining options....................................................................................................................................................23
    Viewing EDA server details................................................................................................................................24
    Viewing U2 server details..................................................................................................................................25
    EDA schema examples........................................................................................................................................26
      Scalar function..................................................................................................................................................26
      TRANS function...............................................................................................................................................27
      Table function example................................................................................................................................29
  Verifying EDA schemas........................................................................................................................................32
    Verification example...........................................................................................................................................34
    Viewing the EDA schema....................................................................................................................................35
    Viewing the DDL scripts.......................................................................................................................................35
    Converting data from U2 to an external database...............................................................................................36
    Viewing EDA files................................................................................................................................................37
  Accessing data in an external database....................................................................................................................38
    Listing data using RetriVe................................................................................................................................38
    Listing data using UniData/UniVerse SQL...........................................................................................................39
Chapter 3: External database supplied drivers............................................................................................................40
  EDA Oracle driver.......................................................................................................................................................40
    Set up the EDA environment................................................................................................................................40
    Set up the Oracle connection file..........................................................................................................................40
    Set up dynamic-loading library..............................................................................................................................41
    Create ORACLEPATH environment variable......................................................................................................41
    Create the EDA data source..................................................................................................................................41
    Oracle data type mapping....................................................................................................................................41
  EDA DB2 driver..........................................................................................................................................................42
    Set up the EDA environment................................................................................................................................42
    Install DB2 or the DB2 client................................................................................................................................42
    Set up connection to the DB2 database..................................................................................................................42
    Create the EDA data source................................................................................................................................42
Chapter 4: External database access driver API........................................................................................................47
EDA driver API.............................................................................................................................................47
Registering an EDA driver........................................................................................................................................47
EDA driver functions...............................................................................................................................................48
EDADRV_LoadSymbols........................................................................................................................................48
EDADRV_Connect..................................................................................................................................................49
EDADRV_Disconnect...........................................................................................................................................50
EDADRV_EndTransaction..................................................................................................................................51
EDADRV_PrepareStmt.........................................................................................................................................51
EDADRV_ExecuteStmt........................................................................................................................................51
EDADRV_CloseStmt................................................................................................................................................53
EDADRV_DropStmt...............................................................................................................................................54
EDADRV_FetchStmt..............................................................................................................................................55
EDADRV_Perform..................................................................................................................................................56
EDADRV_GetEDAAttr........................................................................................................................................57
EDADRV_GetErrmsg.............................................................................................................................................58
EDADRV_Cleanup................................................................................................................................................58
EDADRV_FreeResult...........................................................................................................................................58
EDADRV_GetDBInfo.............................................................................................................................................59
EDADRV_GetSpecialInfo..................................................................................................................................60
ECL command......................................................................................................................................................61
EDA driver log files...............................................................................................................................................62
The EDA driver header file..................................................................................................................................62

Chapter 5: EDA TCL commands........................................................................................................................................70
EDA.CONNECT......................................................................................................................................................70
EDA.CONVERT.....................................................................................................................................................70
EDA.DISCONNECT...............................................................................................................................................71
EDA.EXCEPTION..................................................................................................................................................71
EDA.LOG.............................................................................................................................................................72
EDA.VERSION......................................................................................................................................................72
LIST.EDAMAP....................................................................................................................................................72
SAVE.EDAMAP...................................................................................................................................................73
SELECT.EDA.NONCONFORMING.....................................................................................................................74
VERIFY.EDAMAP.................................................................................................................................................74

Chapter 6: EDA exception handling.............................................................................................................................76
The EDA_EXCEPTION file.........................................................................................................................................76
INMAT..................................................................................................................................................................78

Chapter 7: EDA replication..................................................................................................................................................79
Setting up a server....................................................................................................................................................79
Defining a data source.............................................................................................................................................79
Defining EDA replication parameters....................................................................................................................80
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuring replication parameters</td>
<td>81</td>
</tr>
<tr>
<td>Configuring the replication system</td>
<td>83</td>
</tr>
<tr>
<td>Choosing files to replicate</td>
<td>86</td>
</tr>
<tr>
<td>Synchronizing replication files</td>
<td>88</td>
</tr>
<tr>
<td>Creating EDA schemas for replicated files</td>
<td>89</td>
</tr>
<tr>
<td>Creating default EDA schemas</td>
<td>89</td>
</tr>
<tr>
<td>Creating EDA schemas</td>
<td>90</td>
</tr>
<tr>
<td>Converting replicated files to EDA files</td>
<td>91</td>
</tr>
<tr>
<td>Chapter 8: EDA best practices</td>
<td>93</td>
</tr>
<tr>
<td>Map selected fields</td>
<td>93</td>
</tr>
<tr>
<td>Avoid multiple multivalued associations</td>
<td>93</td>
</tr>
<tr>
<td>Avoid restrictive data types</td>
<td>93</td>
</tr>
<tr>
<td>Data types for record IDs</td>
<td>93</td>
</tr>
<tr>
<td>RECORD_BLOB</td>
<td>94</td>
</tr>
<tr>
<td>Updating an EDA file from the external database</td>
<td>94</td>
</tr>
</tbody>
</table>
Chapter 1: External Database Access

External Database Access (EDA) enables you to convert data stored in the Rocket U2 database to a 1NF database, such as Microsoft SQL Server, then access that data using existing UniVerse BASIC programs, Retrieve, or UniData/UniVerse SQL.

**Note:** EDA was not designed to access data that already resides in a 1NF database. To access this type of data, use the UniVerse BASIC SQL Client Interface (BCI).

You must create an EDA Map Schema to define the translation between U2 and the external database model, which may differ from the U2 model. Additionally, the U2 dictionary record does not fully describe the data it defines. For example, the U2 dictionary record does not define the data type.

In order to use EDA, you must have the external database client installed on the machine where you are running U2. In addition, you must be able to access the external server where you want the data to reside using that client.

**Note:** EDA does not support a multifile file architecture. This includes U2 hashed files and DIR files that were created using the MULTIFILE or MULTIDIR keywords.

First Normal Form (1NF) vs. Non-First Normal Form

Many relational databases, including DB2, SQL Server, and Oracle follow the First Normal Form (1NF) data model. In this model, the relation is considered to be 1NF if and only if each attribute of the relation is atomic, meaning that each column must contain only a single value, and each row must contain the same columns.

U2 follows the nested relational, or Non-First Normal Form model, referred to as NF2. This model enables you to store data in a variety of attributes: singlevalued, multivalued, and multi-subvalued, avoiding data redundancy.

Table concepts

This section is provided to help you understand how the EDA Schema Manager generates tables on an external database, such as SQL Server, so you can plan your mapping strategy. The EDA Schema Manager imposes rules on creating, modifying, and dropping tables.

Representing multivalues

To represent the two nested levels of data within UniVerse files or UniVerse SQL tables (singlevalued, and multivalued), the EDA Schema Manager creates two types of tables, one for each nested level:

- Singlevalued attributes (S) – a table that represents all singlevalued attributes. In this document this table is also called the primary table.

For each association:

- Multivalued attributes (MV) – a table containing multivalued attributes of the association. These tables are “linked” through primary and foreign keys.
Chapter 1: External Database Access

**Note:** Each nonassociated multivalued attribute is mapped to a single external database table linked to the primary table through the primary and foreign keys.

**Primary and foreign keys**

The primary and foreign keys establish the same data relationship between tables as associations do in U2 files or UniVerse/UniData SQL tables.

The purpose of a primary key is to specify one or more attributes whose data values uniquely identify each row of a table.

The purpose of a foreign key is to represent a hierarchical, or parent/child, relationship between two tables. For example, a table containing multivalued attributes is the child of the primary table. The foreign key to this table points to the primary key of the parent, or primary, table.

In order to ensure the uniqueness of the primary key values of the external table containing multivalued attributes, the EDA Schema Manager adds an additional column to that table. Together with the record ID, this column uniquely identifies each row of the multivalued attributes table. This column also contains generated values so that each value of the multivalued attribute is indexed according to its location within the original U2 attribute. This not only ensures the uniqueness of each row in the external table, but preserves the order of values in the multivalued attribute. This also applies to the table containing multi-subvalued attributes: its primary key consists of three columns, the record ID, the additional key column of the multivalued attributes table, and the third column that ensures the uniqueness of the key and the correct order of multi-subvalued attributes. The record ID column of the multivalued attributes table is the foreign key pointing to the primary key of the primary, singlevalued attributes table. The combination of the record ID and the second column of the multi-subvalued attributes table is the foreign key pointing to the primary key of the multivalued attributes table.

**Associations**

The “association” is the mechanism that U2 uses to establish a relationship among attributes. Within an association, multivalued attributes and multi-subvalued attributes are related to, or associated with, each other.

Following is an example of related information that would be stored in a U2 database as an association: The customers of a business each placing orders that contain a product ID, a description, a serial number, the date the order was purchased, the date the order was paid, the list price, the actual price, the discount, the date service starts, the date service ends, the price for the service, and the date the service was paid. You do not want the price for one product getting mistaken for that of another, and you want the correct product names related to the correct product IDs.

For each association, the EDA Schema Manager creates one multivalued attributes table. If the U2 file contains more than one association, the EDA Schema Manager creates separate multivalued (MV) table for each association. One singlevalued attribute (S) table can be the parent of many multivalued attribute (MV) tables.

**Virtual attributes and I-descriptors**

You can map virtual attributes or V-type attributes (UniData) or I-descriptors (UniVerse) to an external database.
There are three types of mapping:

- **Simple** – a simple virtual attribute/l-descriptor, such as A + B. These are virtual attribute formulas that are translated to expressions and SCALAR functions.

- **TRANS** – a virtual attribute/l-descriptor that performs a TRANS operation. These are virtual attributes or l-descriptors that are mapped using TRANS or TABLE function type of mapping. TABLE function mapping is used for multiple TRANS operations.

- **Materialized Virtual/l-descriptor** – a virtual attribute/l-descriptor that is evaluated in U2, with the result stored in DB2. If you are mapping this type of virtual attribute/l-descriptor, select DATA as the type of mapping.

**Note:** You cannot update U2 virtual attributes/l-descriptors. Virtual attributes/l-descriptors are evaluated by the database engine according to the formula you specify in dictionary record. Likewise, you cannot update virtual attributes/l-descriptors that you map to the external database. This applies to all types of virtual attributes/l-descriptors, including materialized ones. Do not attempt to update their values using external database tools, or you risk compromising the consistency of your data and U2 applications.

For more information about these types of attributes, see Defining attribute details, on page 21.

## Mapping example

Consider the following dictionary from the UniVerse demo database CUSTOMER file:

<table>
<thead>
<tr>
<th>Type &amp; Field</th>
<th>Field</th>
<th>Conversion</th>
<th>Column</th>
<th>Output Depth &amp; Heading</th>
<th>Format Assoc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTID</td>
<td>D</td>
<td>0</td>
<td>P(0N)</td>
<td>Customer ID</td>
<td>10R</td>
</tr>
<tr>
<td>@ID</td>
<td>D</td>
<td>0</td>
<td></td>
<td>CUSTOMER</td>
<td>10L</td>
</tr>
<tr>
<td>SAL</td>
<td>D</td>
<td>1</td>
<td></td>
<td>Salutation</td>
<td>5T</td>
</tr>
<tr>
<td>FNAME</td>
<td>D</td>
<td>2</td>
<td></td>
<td>First Name</td>
<td>12T</td>
</tr>
<tr>
<td>LNAME</td>
<td>D</td>
<td>3</td>
<td></td>
<td>Last Name</td>
<td>16T</td>
</tr>
<tr>
<td>COMPANY</td>
<td>D</td>
<td>4</td>
<td></td>
<td>Company Name</td>
<td>20T</td>
</tr>
<tr>
<td>ADDR1</td>
<td>D</td>
<td>5</td>
<td></td>
<td>Address line 1</td>
<td>30T</td>
</tr>
<tr>
<td>ADDR2</td>
<td>D</td>
<td>6</td>
<td></td>
<td>Address line 2</td>
<td>30T</td>
</tr>
<tr>
<td>CITY</td>
<td>D</td>
<td>7</td>
<td></td>
<td>City</td>
<td>12T</td>
</tr>
<tr>
<td>STATE</td>
<td>D</td>
<td>8</td>
<td>P(2A)</td>
<td>State</td>
<td>2L</td>
</tr>
<tr>
<td>ZIP</td>
<td>D</td>
<td>9</td>
<td>P(5N)</td>
<td>Zip</td>
<td>5L</td>
</tr>
<tr>
<td>PHONE</td>
<td>D</td>
<td>10</td>
<td>P(&quot;3N&quot;)&quot;3N</td>
<td>Telephone</td>
<td>13R</td>
</tr>
<tr>
<td>PRODID</td>
<td>D</td>
<td>11</td>
<td>P(1A4N)</td>
<td>Product</td>
<td>5L</td>
</tr>
<tr>
<td>SER_NUM</td>
<td>D</td>
<td>12</td>
<td>P(6N)</td>
<td>Serial#</td>
<td>6L</td>
</tr>
<tr>
<td>PRICE</td>
<td>D</td>
<td>13</td>
<td>MD0,$</td>
<td>Price</td>
<td>7R</td>
</tr>
<tr>
<td>BUY_DATE</td>
<td>D</td>
<td>14</td>
<td>D2/</td>
<td>Date Purchased</td>
<td>8R</td>
</tr>
<tr>
<td>PAID_DATE</td>
<td>D</td>
<td>15</td>
<td>D2/</td>
<td>Date paid</td>
<td>8R</td>
</tr>
<tr>
<td>SVC_PRICE</td>
<td>D</td>
<td>16</td>
<td>MD0,$</td>
<td>Service</td>
<td>7R</td>
</tr>
<tr>
<td>SVC_START</td>
<td>D</td>
<td>17</td>
<td>D2/</td>
<td>Service</td>
<td>8R</td>
</tr>
<tr>
<td>SVC_END</td>
<td>D</td>
<td>18</td>
<td>D2/</td>
<td>start date</td>
<td>8R</td>
</tr>
</tbody>
</table>
The following figure illustrates how the EDA Schema Manager creates tables for all attributes defined in this dictionary. One primary table is the parent of a multivalued table and the multivalued attribute table is the parent of a multi-subvalued table.

The figure shows table names assigned by The EDA Schema Manager on the external database.

- **CUSTOMER** – singlevalued attributes (S) table. Also called the primary table in this document.
- **CUSTOMER_ORDERS_MV** – a multivalued attributes (MV) table based on the association ORDERS.
Chapter 2: EDA Schema Manager

Use the EDA Schema Manager to convert U2 files to an external database format.

To convert U2 files to an external database format, EDA Schema Manager performs the following steps:

1. The EDA Schema processor receives information from the dictionary file for the data file you are converting and other user input.
2. From this information, the EDA Schema Processor creates an EDA schema. This EDA schema is a record in the &EDAMAP& file.
3. Optionally, you can verify the EDA schema.
4. The conversion process uses the EDA schema record and the U2 physical file to create tables and views in the external database.
5. The U2 physical file is replaced by an EDA file in the U2 account. The original data file is saved as filename.edasave. It does not contain data since all data has been transferred to the corresponding external tables.

Starting the EDA Schema Manager

Use the EDA Schema Manager to create a mapping file, called EDA Schema, for a U2 file you are converting to the external database. Then use the mapping file to convert the U2 data to the external database format.

From the Windows Start menu, click Programs → Rocket U2 → EDA Schema Manager.
Creating a new U2 server connection

To create a new U2 server connection, perform the following steps.

Procedure

1. Right-click Servers and select New → U2 Server.
2. In the Create a New U2 Server wizard, from the Name field, enter a unique identifier for the new server.
3. In the Host field, enter the network name of the host computer where the U2 database resides, or the IP address.
4. From the U2 database options, select UniData or UniVerse.
5. If you want to define the protocol type, RPC port number, RPC service name, or the login account, click Advanced.
   A dialog box similar to the following example appears:
Creating a new U2 server connection

a. In the **Protocol Type** field, select the type of communication you are using for the server. You can select **Default**, TCP/IP, or **Lan Manager**. The default is TCP/IP.

b. In the **RPC Port #** field, enter the port number of the UniRPC server running on the host. The default port number is 31438.

c. In the **RPC Service Name** field, enter the name of the RPC service on your system. For UniData, this is normally *udcs* For UniVerse, this is normally *uvcs*.

d. In the **Login Account** field, enter the name of the account to which you want to log on when accessing the U2 database.

e. In the **Commands to Execute** field, click **Add** to enter commands that you want to execute when you log on to the server. Enter commands in the dialog box that appears, then click **OK** when finished.

f. In the **Specify the session to run/debug your BASIC programs on server side** area, specify the type of connection that you want to make to the server. You can specify **Telnet** or **SSH**.

g. In the **Port Number** field, enter the port number you want to use if you do not want to use the default port number of 23.

h. Select the **Use Device License** check box if you want to enable device licensing when connecting to the server.
i. In the **Max # of Sessions** field, enter a number.
6. Click **Finish**.
   The new server appears in the U2 Resource view.

### Connecting to the U2 server

To connect to the U2 server, perform the following steps.

1. Right-click the server name, then click **Connect**.
   When you connect to the server, the Connect to U2 Server dialog box appears.
2. In the **User ID** field, enter the User ID for the machine where U2 is running.
3. Enter the corresponding password in the **Password** field.
   To store the password for future connections, select the Remember me check box. With this check box selected, Microsoft Windows stores the encrypted password on the client computer.
4. If you are using a proxy server, select the **Use Proxy Server** check box.
   a. In the **Proxy Host** field, enter the name or IP address of the computer on which the proxy server is running.
   b. In the **Proxy Port** field, enter the number of the port on which the proxy server listens for communication from UniVerse.
5. Click **Finish**.
   The accounts and existing data sources definitions appear in the U2 Resource view.

### Managing connections

You must define a data source pointing to the external database to which you want to connect.

### About data sources

You must define a data source pointing to the external database client residing on the machine where U2 is installed. The following external databases are supported:

- IBM DB2
- Microsoft SQL Server
- Oracle Database

**Note:** For more information regarding the external database that you are accessing, see [External database supplied drivers, on page 40](#).

The external database server can be on the same machine as the U2 server or on a different machine. However, the external database client must be on the same machine where the U2 server is installed.

### Connecting to SQL server, Oracle, or IBM DB2

The U2 server can reside on UNIX, Linux, or Windows. After the SQL server, Oracle, or DB2 database server is installed, the appropriate ODBC driver must be installed on the U2 server machine. The drivers for access to the databases are:

- SQL Server - Open source or third-party ODBC library for UNIX
- SQL Server - Native client for Windows
- Oracle - Oracle Client Library (OCI)
- DB2 - DB2 Client Library (CLI)

The installation program automatically places the EDA driver library for SQL server (libcomdrv), the EDA driver library for Oracle (liboradrv), and the EDA driver library for DB2 (libdb2drv) in the $UDTBIN (UniData) or $UVBIN (UniVerse) directory.

**Note:** Microsoft provides SQL server ODBC driver on RedHat and SUSE Linux platforms.
Connecting to Microsoft SQL server using the Native Client

To use the native Microsoft SQL server client, the database must reside on a Windows platform. After the SQL server database is installed, the appropriate SQL server client library (native client) must be installed on the U2 server machine. The U2 installation automatically places the EDA driver library for SQL server (libsqldrv) in the $UDTBIN (UniData) or $UVBIN (UniVerse) directory.

The following example shows how U2 connects to SQL server from a Windows platform:

---

Defining a data source

You must define a data source pointing to the external database to which you want to connect.

**Prerequisites**

- Connecting to the U2 server, on page 14

**Procedure**

1. From the U2 Resource view, expand the server you want to use, right-click **EDA Data Sources**, then click **New → EDA Data Source**.
2. In the Create a New EDA Data Source wizard, enter a unique name for the external data source in the **Data Source Name** field, then click **Finish**.

A data source information tab appears in the right pane of the EDA Schema Manager window, as shown in the following example:

![EDA Data Source](image1.png)

3. In the **External DSN** field, enter the name of the external database client that provides the connection to the desired external database instance. For Microsoft SQL Server, it is the name of the ODBC Data Source you have defined in ODBC Data Source Administration. For DB2, it is the database name specified in the `CATALOG DATABASE` command. For Oracle, it is the connection name defined in the `tnsnames.ora` file.

4. In the **Driver** field, enter the type of driver and enter any details about it in the **Description** field.

5. Click **Add**. In the EDA Data Source Connection dialog box that appears, enter the following information:
   a. In the **Login User ID** field, enter the user ID on the external server.
   b. In the **Password** field, enter the password corresponding to the User ID. Enter the password again in the **Re-enter Password** field.
   c. If you want to maintain the connection to the external server after a transaction commits, select **YES** from the **Hold Flag** drop-down menu. If you want to disconnect from the external server after the transaction commits, select **NO**.
   
   **Note:** If you do not use UniVerse BASIC transactions, each U2 database operation, such as a **READ** or **WRITE**, corresponds to an external transaction.
   
   d. In the **Qualified Users** field, enter the U2 user IDs of users who can access the external server from the U2 account using the external Login User ID you specify. Separate the users by a “|” symbol. If all U2 users can access the external account, enter an asterisk (“*”).
   e. Click **Finish**.

**Results**

The following example shows a completed EDA Data Source:
To test the connection to the external instance, click Test. A success message appears if the connection is successful; otherwise a detailed error message appears.

From the File menu, click Save to save your data source definition, or click the Save icon.

Creating EDA schemas

Prerequisites

- Connecting to the U2 server, on page 14
- Defining a data source, on page 16

Procedure

1. From the U2 Resource view, expand Accounts and then expand the account where the files you want to convert reside.
2. Right-click the EDA Schema Files, and select New → EDA Map Schema.
3. In the Create New EDA Map Schema wizard that appears, enter a unique name for the EDA schema in the EDA Schema Name field.
5. On the Source U2 file page, click the file for which you are creating a schema.
6. From the **Using Data Source** drop-down menu, select the data source you are using. Click **Next**. The U2 Dictionary Attributes page appears, as shown in the following example:

![U2 Dictionary Attributes](image1)

The U2 Dictionary Attributes page lists the D-type dictionary attribute for the file you specified. Select each dictionary attribute to map to the external database. To select all D-type dictionary attributes, click **Select All**. To clear all dictionary attributes, click **Deselect All**. For selective mapping or virtual field mapping, you must click **Deselect All**.

If you want to selectively map U2 attributes to an external database, only select those attributes you want to map. If you do not select any dictionary attributes, the @ID attribute is automatically mapped. In the following example, only @ID and ADDRESS have been selected:

![U2 Dictionary Attributes](image2)

7. Click **Finish** when you have selected all the dictionary attributes for which you want to create schemas.
Results

The EDA schema you created appears in a tab on the right pane of the EDA Schema Manager window and the Outline view on the bottom left populates, as shown in the following example:

The following example illustrates the appearance of the window after the @ID and ADDRESS attributes have been selected. Notice that a red arrow appears next to the attribute in the U2 File Dictionary portion of the window, indicating the attribute has been mapped. Click the mapped attribute you want to display in the EDA Map Schema view.
Note: The EDA Schema Manager allows 30-character column names for Oracle and DB2 and 60-character column names for SQL Server. If the dictionary ID length is longer, it will be truncated in the EDA Map Schema portion of the window.

Next step

- Defining attribute details, on page 21
- Defining options, on page 23

Defining attribute details

In the Attribute Details area of the EDA Schema Manager, define the mapping details for the attribute you selected.

Prerequisites

- Creating EDA schemas, on page 18

About this task

You can change the name, type, data type, formatting, database name, namespace, and data source. Namespace refers to the external schema name where the conversion process will create the corresponding external tables and views.

The following example shows the Attribute Details (right) for the @ID mapped attribute.

Procedure

1. Select an Occurrence: Once or Once or More.
2. In the Name field, enter the name of the column in the resulting external table.
3. In the Type field, select the type of attribute.
   Valid types are:
Chapter 2: EDA Schema Manager

- **DATA**: used to store attribute values allowed by the data type you specify. This option creates a column in the external database. If this is a D-type attribute, its values are stored in this column. If it is defined as a virtual attribute or V-type(UniData) or I-descriptor (UniVerse), the virtual attribute/I-descriptor is evaluated in U2, and the result is stored in this column in the external database.

- **EXPRESSION**: used for virtual attributes (UniData) or I-descriptors (UniVerse) only. Enter the SQL expression for the virtual/I-descriptor attribute in the **Expr Body** field, such as FNAME CONCAT ' ' CONCAT LNAME.

- **ID DATA**: the primary key in the external table

- **NOT NULL DATA**: specifies that the external database column cannot contain the null value.

- **SCALAR FUNCTION**: used to define a scalar function to execute an equivalent virtual attribute/I-descriptor on the external database. For information about creating a scalar function, see [Scalar function, on page 26](#).

- **TABLE FUNCTION**: used to define a table function For information about creating a table function, see [Table function example, on page 29](#).

- **TRANS**: used for virtual attributes/ I-descriptors containing a TRANS clause only. If you specify TRANS, you must also specify Reference and Parameters. For more information, see [TRANS function, on page 27](#).

- **UNIQUE DATA**: specifies that values in the external database column must be unique

4. In the **Data Type** field, enter the data type for the mapped attribute. In this case, the data type is VARCHAR.

   The EDA Schema Manager automatically converts the data type based on the dictionary record. See [External database supplied drivers, on page 40](#) for descriptions of the driver conversions. If an application attempts to insert or update an external attribute with a value that does not match the data type you define, the EDA system rejects the operation.

5. In UniData, the **Reference** field is used for V-type attributes that contain a TRANS clause. In UniVerse, this field is used for I-type attributes that contain a TRANS clause. Enter the name of the external table TRANS clause reference in this field. You can alternatively drag and drop the applicable attribute from the U2 File Dictionary area to the Attribute Details area.

6. If the mapped attribute is an expression, enter the SQL expression in the **Expr Body** field.

   This field applies to the virtual attribute/I-descriptor that contains a user-defined function or expression. Use the **Expr Body** field to enter the equivalent SQL statement for the expression or function.

7. Specify the attribute or expression in the TRANS function that returns the record ID in the table you are referencing in the external database. Click the plus sign (+) in the **Parameters** portion of the window.

   In the dialog box that appears, enter a **Parameter**. Click **Finish**.

8. In the **Formatting** field, select the appropriate format for the attribute.

   **Note**: The **Index** field is no longer supported.

---

**Results**

If you select all dictionary attributes, U2 maps only D-type attributes. You must map virtual attributes/ I–descriptors manually.

You cannot map unassociated multivalued or I-descriptors. You also cannot map associations that contain only I-descriptors.
Defining options

You can view the options from a mapped EDA Schema.

Prerequisites

▪ Creating EDA schemas, on page 18

Procedure

1. Click Options in the EDA Map Schema area of the U2 EDA Schema Manager. The Options area (right) appears in the EDA Map Schema area.

2. From the Whole Record drop-down menu, select Yes or No to specify whether to store the entire U2 record in the RECORD_BLOB column on the EDA server at the same time the individually mapped U2 fields are written to their mapped columns.

Note: This option can improve the performance of READ operations, especially when mapping multivalued and multi-subvalued attributes since you avoid complex outer-joins.

If the value of Whole Record is Yes, the entire U2 record will be stored in the RECORD_BLOB on the EDA server.

If the value is No, only unmapped fields are stored in the RECORD_BLOB. The default value is No.

If you select Yes, you should not be updating the data from the external database; only update it through UniVerse BASIC or UniData/UniVerse SQL. A failure to comply with this rule can result in inconsistent data.

3. In the Unmapped Field Block (KB) field, specify the size of the character large object (RECORD_BLOB) in kilobytes. The default value is 16.

The RECORD_BLOB serves several purposes:

▪ To hold all attributes that are not explicitly mapped
▪ To hold the entire U2 record when the WHOLE_RECORD flag is set to “Yes”
▪ To hold nonconforming records
4. From the **Nonconforming Record** drop-down menu, select **Yes** or **No** to specify whether to return validation or truncation errors generated by the external database to your application so the behavior of your application does not change.

A nonconforming record is a U2 record that generates an exception error when it is written to the EDA server database, but does not cause an error in the U2 database.

For example, U2 allows you to store a text string in an attribute defined as having a numeric conversion, such as MD2, Date or Time, but this will generate an error in the external database.

If the value of **Nonconforming Record** is set to **Yes**, a NONCONFORMING_FLAG column is created in the EDA file. When a U2 record is determined to be a NONCONFORMING record, the ID of that record is inserted in the primary key column of the external table, the NONCONFORMING column is set to 1, and the entire U2 record is written to the RECORD_BLOB column. In this case, no error is returned to the U2 application. If U2 attempts to write the nonconforming data to a RECORD_BLOB that is not large enough to contain the data, the write fails and U2 writes the record to the EDA_EXCEPTION file on the U2 database and an error is returned to the U2 application.

If the value of **Nonconforming Record** is **No**, the nonconforming data is only written to the EDA_EXCEPTION file on the U2 database and an error is returned to the U2 application.

For more information about retrieving nonconforming data, see **SELECT.EDA.NONCONFORMING**, on page 74.

5. From the **Table Space (KB)** drop-down menu, select the default page size. The maximum page size is 256 KB.

A table space is the basic storage structure on an external database. By default, U2 creates all tables in USERSPACE 1, the default user table space. This table space has a 4 KB page size, so the length of a row of a table is limited to less than 4 KB. If the row length is exceeded, U2 generates an error during the conversion process.

When you select a table space size, for example, 8 KB, the EDA Schema Manager creates the table space EDATBSPC8K if it does not already exist, then creates the EDA tables in this table space.

### Viewing EDA server details

To view information about the EDA server, click the external Schema Name/Table Name in the **EDA Map Schema** pane of the U2 EDA Schema Manager. Information about the EDA server is displayed, as shown in the following example:
The U2 EDA Schema Manager displays the following details about the EDA server:

- The **DBInstance** field displays the name of the instance on the EDA server.
- The **DBMSName** field displays the name and version of the database on the EDA server.
- The **DBMSFamily** field displays the database family to which the DBMS name belongs.
- The **DBModel** field displays the type of database. U2 only supports 1NF databases.
- The **Name Space (Schema)** field displays the name of the schema on the EDA server. You can change the name of the schema.
- The **Root Name** field displays the name of the table on the EDA server. You can change the name of the table.
- The **Data Source** field displays the name of the data source on the EDA server.

**Viewing U2 server details**

To view information about the U2 server, click the U2 file name in the **EDA Map Schema** area of the EDA Schema Manager. Information about the U2 server appears, as shown in the following example:

The EDA Schema Manager displays the following details about the U2 server:

- The **U2 Host** field displays the name of the U2 host server where the file resides.
- The **U2 System** field displays the type of database on the server where the file resides.
- The **U2 Version** field displays the version of the database on the U2 server where the file resides.
- The **U2 Account** field displays the full path to the account on the U2 server where the file resides.
- The **File Name** field displays the name of the file on the U2 server.
EDA schema examples

Use the following sections to see examples of different types of EDA schemas:

- **Scalar function**
  This example illustrates a scalar function, used to execute an equivalent virtual attribute/I-descriptor on the external database.

- **TRANS function**
  This example illustrates the TRANS function, used for virtual attributes/I-descriptors containing a TRANS clause.

- **Table function example**
  U2 allows you to use the DB2 table function concept to evaluate multiple virtual attributes/I-descriptors at the same time. In some cases, you are able to map more than one virtual attribute/I-descriptor with one DB2 user-defined table function.

**Scalar function**

This example illustrates a scalar function, used to execute an equivalent virtual attribute/I-descriptor on the external database.

Assume you have the following I-descriptor attribute defined for the CUSTOMER file in your UniVerse database:

```plaintext
:AE DICT CUSTOMER UPCASE.LNAME
Top of "UPCASE.LNAME" in "DICT STUDENT", 6 lines, 25 characters.
001: I
002: UPCASE(LNAME)
003: 
004: 
005: 30L
006: S
```

This I-descriptor converts the customer's last name to uppercase.

1. To convert this I-descriptor to a scalar function using the EDA Tool, drag UPCASE.LNAME from the U2 File Dictionary pane under CUSTOMER in the EDA Map Schema pane.
2. In the **Attribute Details** pane, change the **Type** to **SCALAR FUNCTION**.
3. In the **Data Type** field, define the data type for the output.
4. In the **Reference** field, enter the external database function and data type for the value you are passing to the function.

   In this example, the DB2 system function that corresponds to the UniVerse **UPCASE** function is **UCASE**, which resides in the SYSFUN Schema in the DB2 database.

   Enter the following formula in the **Reference** field:

   ```plaintext
   SYSFUN.UCASE(VARCHAR(30))
   ```
5. In the **Parameters** field, click the plus sign (+), and enter the field to pass to the scalar function. The Attribute Details should now look like the following example:

The following example contains the output from U2 when you run this scalar function:

```
>LIST CUSTOMER UPCASE.LNAME
LIST CUSTOMER UPCASE.LNAME 03:48:02pm 09 Jun 2010 PAGE 1
CUSTOMER..........................
2 MORRIS
4 KAHN
6 BURKE
3 ARGONNE
5 WILLIAMS
7 GILL
10 MCCAI
8 HOLLAND
12 PATRY
1 SMITH
9 ORLANDO
11 LEWIS
12 records listed.
```

**Parent topic:** EDA schema examples

**TRANS function**

This example illustrates the TRANS function, used for virtual attributes/l-descriptors containing a TRANS clause.

Assume you have the following U2 virtual attribute/l-descriptor defined in the dictionary of the CUSTOMER file:

```
001: I Product description
002: TRANS(PRODUCTS,PRODID,DESCRIPTION,"C")
003:
004: Product Description
```
1. To convert this virtual attribute to a trans function using the EDA Tool, drag TEACHER from the U2 File Dictionary pane under the COURSES node of STUDENT in the EDA Map Schema pane, as shown in the following example:

![Diagram showing the conversion of TEACHER to a trans function]

This i-descriptor executes a translate from the CUSTOMER file to the PRODUCTS file and returns DESCRIPTION.

2. To convert this i-descriptor to a trans function using the EDA Tool, drag DESCRIPTION from the U2 File Dictionary pane under the ORDERS_MV node of CUSTOMER in the EDA Map Schema pane, as shown in the following example:

![Diagram showing the conversion of DESCRIPTION to a trans function]

3. In the Attribute Details pane, change the Type to TRANS.
4. In the Data Type field, define the data type for the output.
5. In the Reference field, enter the name of the external table that contains the DESCRIPTION information.
   In this example, the information resides in PRODUCTS.PRODUCTS/DESCRIPTION.
6. In the **Parameters** field, click the plus sign (+) and enter the field to pass to the **TRANS** function. In this example, **PRODID** is passed to the **TRANS** function. The following example illustrates output from the **DESCRIPTION TRANS** scalar function:

```
LIST CUSTOMER DESCRIPTION SAMPLE 01:14:13pm 11 Jun 2010 PAGE
1
  CUSTOMER. product description.

2  Moderate duty, entry level, color copier
   Heavy duty monochrome copier
   Sorting attachment for M3000/C3000

4  Heavy duty color copier

6  Moderate duty, monochrome copier

5  Low cost, entry level, light duty, monochrome copier
   Low cost, entry level, light duty, monochrome copier

7  Moderate duty, monochrome copier
   Sorting attachment:
```

Parent topic: **EDA schema examples**

---

**Table function example**

U2 allows you to use the DB2 table function concept to evaluate multiple virtual attributes/I-descriptors at the same time. In some cases, you are able to map more than one virtual attribute/I-descriptor with one DB2 user-defined table function.

Assume you create a monthly report containing product descriptions and prices. To create this report from UniVerse, you use the following I-descriptors for product description and list price:

```plaintext
:AE DICT CUSTOMER DESCRIPTION
001: I
002: TRANS(PRODUCTS, PRODID, DESCRIPTION, "C")
003:
004: Product Description
005: 20T
006: M
007: ORDERS

:AE DICT STUDENT LIST_PRICE
001: I
002: TRANS(PRODUCTS, PRODID, LIST,"C")
003:MDO,$
004: List Price
005: 7R
006: M
007: Orders
```

To map these I-descriptors, drag each one from the **U2 File Dictionary** pane to the ORDERS_MV node in the **EDA Map Schema** pane, as shown in the following example:
Although you map DESCRIPTION and LIST_PRICE separately, some external databases allow you to create one function for use with multiple attributes. You have to define the EDA Map Schema for both I-descriptors, but you only have to define the function once.

The following example illustrates how you would use a DB2 table function in order to evaluate both DESCRIPTION and LIST_PRICE on the DB2 database.

First, let’s map the DESCRIPTION I-descriptor:

In the Attribute Details portion of the window, change the **Type** to **TABLE FUNCTION**.

In the **Data Type** field, enter the data type for the output of the attribute you specified in the **Name** field.

In the **Reference** field, enter the external database schema name, the name of user-defined function you are defining in the **Expr Body** field, the data type for the input value, and the external function attribute name, as shown in the following example:

```
CUSTOMER2.GET_PRODUCT(VARCHAR(20)) / DESCRIPTION
```
In this example, the user-defined function will be named GET_PRODUCT and reside in the CUSTOMER2 schema in the external database. The data type of the input parameter is VARCHAR(20), and you are using the output parameter PRODUCT.

In the **Expr Body** field, enter the table function body. In this example, the function is defined as:

```sql
PRODID
F1:BEGIN ATOMIC RETURN SELECT B.DESCRIPTION, B.LIST_PRICE FROM PRODUCTS.PRODUCTS AS B WHERE B.ID=PRODID; END
```

In the **Parameters** field, click the plus sign (+) to define the parameter to pass to the table function and the output parameters you want to return. For output parameters, you specify OUTPUT, the name of the attribute to return, and the data type. In this example, the output parameters are defined as:

```sql
OUTPUT DESCRIPTION VARCHAR(50)
OUTPUT LIST_PRICE VARCHAR(10)
```

The following example shows the DDL scripts UniVerse creates for this table function:

```sql
CREATE FUNCTION CUSTOMER2.GET_PRODUCT(PRODID VARCHAR(5)) RETURNS TABLE(DESCRIPTION VARCHAR(50), LIST_PRICE VARCHAR(10)) F1:BEGIN ATOMIC RETURN SELECT B.DESCRIPTION, B.LIST_PRICE FROM PRODUCTS.PRODUCTS AS B WHERE B.ID=PRODID; END
```

Next, let’s map the LIST_PRICE I-descriptor.

Click the **LIST_PRICE** attribute. The following example illustrates the **Attribute Details** for this attribute:

In the Attribute Details portion of the window, change the **Type** to **TABLE FUNCTION**.

In the **Data Type** field, enter the data type for the output of the attribute you specified in the **Name** field.
In the Reference field, enter the DB2 Schema name, the name of user-defined function you previously defined in the DESCRIPTION table function (GET_PRODUCT), the data type for the input value, and the external database function attribute name, as shown in the following example:

CUSTOMER2.GET_PRODUCT(VARCHAR(5))/LIST_PRICE

In this example, the user-defined function GET_PRODUCT resides in the CUSTOMER2 schema in the DB2 database. The data type of the input parameter is VARCHAR(5), and you are using the output parameter LIST_PRICE.

Since you previously defined the GET_PRODUCT function, you do not need to enter data in the Expr Body.

In the Parameters field, click the plus sign (+) to define the parameter to pass to the table function. You do not need to define the output parameters since they were previously defined in the GET_PRODUCT function.

**Note:** For information about creating external database user-defined functions, see the external database documentation.

**Parent topic:** EDA schema examples

### Verifying EDA schemas

After you have created an EDA Schema, you can verify the EDA Schema.
1. To verify the EDA Schema, click the **Verify** icon on the toolbar. A dialog box similar to the following example appears:

![Verify EDA Map Schema](image)

2. In the Verify the EDA Map Schema dialog box, select one of the following verification types:
   - **Syntax** – Verifies that the syntax of the SQL statements that creates the external tables is correct.
   - **Metadata** – Verifies that all the metadata required to create the external tables exists.
   - **Data** – Verifies that the UniVerse data meets the requirements for the external tables. You can select one of the following options when verifying your data:
     - **All Records** – analyzes each record in the UniVerse data file
     - **Specified Records** – you can enter specific record IDs to analyze. Separate each record ID with a right brace (}).
     - **First n Records** – the system verifies the first \( n \) records you specify.
     - **Every \( n \)-th Records** – the system verifies every \( n \)-th record you specify.
3. To view the EDA schema, click **Show Schema**. The schema appears in the dialog box, as shown in the following example:

![Show Schema dialog box](image)

**Verification example**

The following example shows a listing from the UniVerse CUSTOMER file:

```
<table>
<thead>
<tr>
<th>LIST CUSTOMER</th>
<th>FNAME</th>
<th>LNAME</th>
<th>CITY</th>
<th>STATE</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOMER...</td>
<td>First Name</td>
<td>Last Name</td>
<td>City.......</td>
<td>State</td>
<td>Zip..</td>
</tr>
<tr>
<td>2</td>
<td>Diana</td>
<td>Morris</td>
<td>Waltham</td>
<td>MA</td>
<td>01133</td>
</tr>
<tr>
<td>4</td>
<td>Jill</td>
<td>Kahn</td>
<td>Boston</td>
<td>MA</td>
<td>01103</td>
</tr>
<tr>
<td>6</td>
<td>Betty</td>
<td>Burke</td>
<td>White River</td>
<td>VT</td>
<td>01644</td>
</tr>
<tr>
<td>3</td>
<td>David</td>
<td>Argonne</td>
<td>Bedford</td>
<td>MA</td>
<td>01182</td>
</tr>
<tr>
<td>5</td>
<td>Kenneth</td>
<td>Williams</td>
<td>Providence</td>
<td>RI</td>
<td>03171</td>
</tr>
<tr>
<td>7</td>
<td>Martha</td>
<td>Gill</td>
<td>Derry</td>
<td>NA</td>
<td>04429</td>
</tr>
<tr>
<td>10</td>
<td>Andrew</td>
<td>McCaig</td>
<td>Brattleboro</td>
<td>VT</td>
<td>03356</td>
</tr>
<tr>
<td>8</td>
<td>Steven</td>
<td>Holland</td>
<td>Lowell</td>
<td>MA</td>
<td>01386</td>
</tr>
<tr>
<td>12</td>
<td>Laurie</td>
<td>Patry</td>
<td>Littleton</td>
<td>MA</td>
<td>01442</td>
</tr>
<tr>
<td>1</td>
<td>Samuel</td>
<td>Smith</td>
<td>Concord</td>
<td>NH</td>
<td>02131</td>
</tr>
<tr>
<td>9</td>
<td>Nicole</td>
<td>Orlando</td>
<td>Burlington</td>
<td>MA</td>
<td>01173</td>
</tr>
<tr>
<td>11</td>
<td>Skip</td>
<td>Lewis</td>
<td>Plymouth</td>
<td>MA</td>
<td>01382</td>
</tr>
</tbody>
</table>
```

Notice that the City attribute for Record ID 6 exceeds the specified length of 12 characters. When you verify the data for the EDA Map Schema, the following error message appears:

```
In C:\u2\uv\sys\CTLG\e\EDAMAPSUB at line 2056 EDA_write_tuple error, id = "521814564"
In C:\u2\uv\sys\CTLG\e\EDAMAPSUB at line 2056 EDA DB2 Driver: [IBM] [CLI Driver] CLI0109E String data right truncation, SQLSTATE=22001
5 records passed data verification.
1 records failed on data verification.
```

Fix the incorrect data in the records before converting the file to an EDA file. If you do not correct the data, the record will not be converted, and will appear in the EDA_EXCEPTION file. For more information and the EDA_EXCEPTION file, see **EDA exception handling, on page 76**.
If the verification succeeds, “Successful” appears in the dialog box. If an error occurs, the error appears in the dialog box.

**Viewing the EDA schema**

If you want to view the EDA schema, click *Show Schema*. The schema appears in the dialog box, as shown in the following example:

![EDA Schema Example](image)

**Viewing the DDL scripts**

To view the DDL script that UniVerse will use to generate the data on the external table, click the *DDL Scripts* icon on the toolbar.

Get DDL Scripts appear in the window, as shown in the following example:
Converting data from U2 to an external database

You can convert data from U2 to the external database.

1. Click the Convert Data icon.
2. In the Convert the U2 File to EDA File dialog box, select the type of conversion to use:
   - Click Force to drop existing tables on the external database before creating new ones. You must select this option if you are reconverting data.
   - Click Verbose to display detailed messages and DDL scripts during the conversion process.
3. Click EDA Convert. If the conversion is successful, a message reports the total number of converted records. If the conversion is not successful, error messages describe the problems.
4. To see which files were converted from U2 to the external database, from the **EDA Schema Manager**, click the plus sign next to expand EDA Schema files.

**Viewing EDA files**

To see which files were converted from U2 to the external database, from the **EDA Schema Manager**, click the plus sign next to expand EDA Schema files.
Accessing data in an external database

Use Retrieve, UniData/UniVerse SQL, and UniVerse BASIC to access the data in the external database.

Listing data using Retrieve

You can use the Retrieve LIST command to view the converted data on the external database, as shown in the following example:

```
$<56>LIST CUSTOMER NAME 1NAME CITY STATE 01:04:25pm 17 Jun 2010 PAGE
1
CUSTOMER.  First Name...  Last Name......  City...............  State
2  Diane  Morris............................  Waltham  NA
4  Jill  Kahn..............................  MAson  NA
6  Betty  Burke............................  White River Jun  VT
3  David  Argonne...........................  Bedford  NA
5  Kenneth  Williams.......................  Providence  RI
7  Martha  Gill..............................  Derry  NH
10  Andrew  McCaig.........................  Brattleboro  VT
8  Steven  Holland..........................  Lowell  MA
12  Laurie  Patry............................  Littleton  NA
1  Samuel  Smith............................  Concord  NH
9  Nicole  Orlando..........................  Burlington  MA
11  Skip  Lewis..............................  Plymouth  MA
12 records listed.
```
Listing data using UniData/UniVerse SQL

You can use the SQL SELECT command to view the converted data on the external database, as shown in the following example:

```sql
>SELECT FNAME, LNAME, CITY, STATE FROM CUSTOMER,
First Name.. Last Name...... City......... State
Morris Waltham MA
Kahn Boston MA
Burke White River Jun VT
Agronne Bedford MA
Williams Providence RI
Gill Derry NH
McCaig Brattleboro VT
Holland Lowell MA
Patry Littleton MA
Smith Concord NH
Orlando Burlington MA
Lewis Plymouth MA
```
12 records listed.
Chapter 3: External database supplied drivers

This section describes the allowed drivers, how to set them up with U2, and how EDA maps U2 data to the external data types.

The following supplied drivers are explained:

- **EDA Oracle driver**
  The EDA Oracle driver is a dynamic-loading library which the EDA engine uses to exchange data with an Oracle database.

- **EDA DB2 driver**
  The EDA DB2 driver is a dynamic-loading library which the EDA engine uses to exchange data with a DB2 database.

- **EDA SQL server driver**
  The EDA SQL server driver is a dynamic-loading library which the EDA engine uses to exchange data with a SQL server database.

- **EDA SQL server driver for UNIX and Linux**
  U2 provides an EDA SQL server driver for UNIX and Linux. The EDA SQL server driver for UNIX and Linux is used to provide access for U2 installations running on UNIX or Linux to Microsoft SQL server running on Windows platforms.

### EDA Oracle driver

The EDA Oracle driver is a dynamic-loading library which the EDA engine uses to exchange data with an Oracle database.

The EDA Oracle driver supports Oracle Version 11g.

Parent topic: External database supplied drivers

### Set up the EDA environment

On UNIX platforms, execute the operating system-level command `edasetup.sh` to set up the EDA environment. This command prompts you for information and generates the `edaconfig` file in the `$UVHOME` account. The `edaconfig` file contains the following information:

```
DRIVER=ORACLE
ORACLEPATH=/test1/oracle/instantclient_11_1
LOGLEVEL=0
```

You can change the LOGLEVEL to 1 or 2. The higher the log level, the more information is captured.

On Windows platforms, manually edit the `edaconfig` file to add LOGLEVEL.

### Set up the Oracle connection file

Set up the `tnsnames.ora` file, used to connect to the Oracle database. The following example illustrates the `tnsnames.ora` file:

```
ORDEVDDB=
(DESCRIPTION =
```
Set up dynamic-loading library

To load the Oracle OCI libraries, you must set up a dynamic-loading library path. The following table specifies where to add the Oracle library path based on the platform you are using:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Oracle path location</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>Add Oracle library path to LIBPATH</td>
</tr>
<tr>
<td>HP</td>
<td>Add Oracle library path to SHLIB_PATH</td>
</tr>
<tr>
<td>Windows</td>
<td>Add Oracle library path to PATH</td>
</tr>
<tr>
<td>Other</td>
<td>Add Oracle library path LD_LIBRARY_PATH</td>
</tr>
</tbody>
</table>

If you do not set this environment variable correctly, you will not be able to connect to Oracle. After you add the Oracle library path, restart unirpcd.

Create ORACLEPATH environment variable

UniVerse 11.2.0 through 11.2.2 only. On UNIX platforms, you need to set the ORACLEPATH environment variable specifying the directory containing the Oracle libraries.

Create the EDA data source

When you create the EDA data source in the EDA Schema Manager, the external DB Name should be the connection name you specified in the tnsnames.ora file, such as ORDEVDB.

Oracle data type mapping

The next table describes how EDA maps U2 data to Oracle data types:

<table>
<thead>
<tr>
<th>U2 data</th>
<th>Oracle data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characters</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>Date</td>
<td>DATE</td>
</tr>
<tr>
<td>Time</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>Number (integer)</td>
<td>NUMBER(38)</td>
</tr>
<tr>
<td>Number (noninteger)</td>
<td>NUMBER</td>
</tr>
<tr>
<td>U2 data</td>
<td>Oracle data type</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Unmapped fields</td>
<td>CLOB</td>
</tr>
</tbody>
</table>

**Note:** When mapping U2 time data to SQL Server DATETIME, the date portion is filled with 01/01/2000.

## EDA DB2 driver

The EDA DB2 driver is a dynamic-loading library which the EDA engine uses to exchange data with a DB2 database.

The EDA DB2 driver supports DB2 8.0 and greater.

The DB2 driver is available on AIX and HP Itanium platforms.

**Parent topic:** [External database supplied drivers](#)

## Set up the EDA environment

On UNIX platforms, execute the operating system-level command `edasetup.sh` to set up the EDA environment. This command prompts you for information and generates the `edaconfig` file in the `$UVHOME` account. The `edaconfig` file contains the following information:

```
DRIVER=DB2
DB2PATH=/home/db2inst1/sqllib
LOGLEVEL=0
```

You can change the LOGLEVEL to 1 or 2. The higher the log level, the more information is captured.

On Windows platforms, manually edit the `edaconfig` file to add LOGLEVEL.

## Install DB2 or the DB2 client

Install the DB2 client on the machine where U2 is installed. Create a database on the DB2 server to which U2 can connect. If you install the DB2 client after you install U2, you must restart unirpcd.

## Set up connection to the DB2 database

After you install the DB2 client, create a cataloged database on the DB2 client to use to connect to the database on the DB2 server.

## Create the EDA data source

Create an EDA data source in the EDA Schema Manager.

Make sure you use the cataloged database name as the External DB Name if you installed the DB2 client. Otherwise, use the database name directly as the External DB Name.
DB2 data type mapping

The following table describes how EDA maps U2 data to DB2 data types:

<table>
<thead>
<tr>
<th>U2 data</th>
<th>DB2 data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characters</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>Date</td>
<td>DATE</td>
</tr>
<tr>
<td>Time</td>
<td>TIME</td>
</tr>
<tr>
<td>Number (integer)</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Number (noninteger)</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Unmapped fields</td>
<td>CLOB</td>
</tr>
</tbody>
</table>

EDA SQL server driver

The EDA SQL server driver is a dynamic-loading library which the EDA engine uses to exchange data with a SQL server database.

The EDA SQL server driver supports SQL server 2005 and greater. When setting up an EDA ODBC DSN with the SQL server client, only the SQL server native client x.x driver is supported, not the default SQL server driver.

**Note:** The EDA SQL server driver is available on Windows platforms only.

Parent topic: External database supplied drivers

Install SQL server and create ODBC data source

Install SQL server, or at least the SQL Server Native Client, on the same machine where U2 is installed.

Create a database on SQL Server to which U2 can connect.

Create an ODBC data source to use to connect to the SQL Server database. Use Control Panel -> Administrative Tools -> Data Sources (ODBC) to open the ODBC Data Source Administrator dialog box. Add a system DSN and select SQL Native Client as the driver.

Create the EDA data source

Create an EDA Data Source in the EDA Schema Manager. Make sure you use the ODBC data source created in the previous step as the External DB Name.

Set up the EDA configuration file

Create an edaconfig file in $UDTHOME for the EDA SQL Server driver log. Add the following information to the edaconfig file:

LOGLEVEL=0
You can change the LOGLEVEL to 1 or 2. The higher the log level, the more information is captured.

**SQL server data types**

The following table describes how EDA maps U2 data to SQL server data types.

<table>
<thead>
<tr>
<th>U2 data</th>
<th>SQL server data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characters</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>Date</td>
<td>DATE (default) or DATETIME</td>
</tr>
<tr>
<td>Time</td>
<td>TIME (default) or DATETIME</td>
</tr>
<tr>
<td>Number (integer)</td>
<td>INT</td>
</tr>
<tr>
<td>Number (noninteger)</td>
<td>REAL</td>
</tr>
<tr>
<td>Unmapped fields</td>
<td>VARCHAR(MAX)</td>
</tr>
</tbody>
</table>

**Note:**

Use of the DATE and TIME SQL server data type is supported on UniVerse versions 11.3.1 and later. When mapping U2 date data to SQL server DATETIME, the time portion is filled with 00:00:00. When mapping U2 time data to SQL server DATETIME, the date portion is filled with 01/01/1753.

**EDA SQL server driver for UNIX and Linux**

U2 provides an EDA SQL server driver for UNIX and Linux. The EDA SQL server driver for UNIX and Linux is used to provide access for U2 installations running on UNIX or Linux to Microsoft SQL server running on Windows platforms.

The EDA SQL server driver for UNIX and Linux is a dynamic-loading library. It uses unixODBC as the ODBC driver, such as the Easysoft ODBC driver, to communicate with Microsoft SQL Server. The EDA SQL server driver for UNIX and Linux is available on AIX, HP Itanium, Solaris SPARC, and Linux platforms.

**Parent topic:** [External database supplied drivers](#)

**Set up the EDA environment**

Execute the operating system-level command `edasetup.sh` to set up the EDA environment. This command prompts you for information and generates the `edaconfig` file in the `$UVHOME` account. The `edaconfig` file contains the following information:

```
DRIVER=ODBC
ORACLEPATH=/usr/local/easysoft
LOGLEVEL=0
```

You can change the LOGLEVEL to 1 or 2. The higher the log level, the more information is captured.
Install unixODBC and third-party ODBC driver

Install the ODBC data source manager unixODBC and the third-party ODBC driver on the machine on which U2 is installed.

If the third-party ODBC driver, such as Easysoft SQL Server driver includes unixODBC, you just have to install the ODBC driver.

Set up connection to external database

To connect to the external database, you must set up a connection using unixODBC. To do this, you must add a data source. You can add a system data source which is available to anyone who logs on to this UNIX machine, or a user data source which is only available to the users who are currently logged on to this UNIX machine.

U2 adds the system data source to /etc/odbc.ini and the user data source to $HOME/.odbc.ini, as shown in the following example:

```
[SQL2005]
Driver = Easysoft ODBC-SQL Server
Description = SQL Server DSN created during installation
Server = jyaosql
Port =
User = jyao
Password = 1234
Language = English
Database = DEVDB
Logging = 0
LogFile =
QuotedId = Yes
AnsiNPW = Yes
Mars_Connection = Yes
```

U2 adds the ODBC driver to /etc/odbcinst.ini for the system data source or $HOME/.odbcinst.ini for the user data source, as shown in the following example:

```
[Easysoft ODBC-SQL Server]
Driver =
/usr/local/easysoft/sqlserver/lib/libessqlsrv.a
Setup =
/usr/local/easysoft/sqlserver/lib/libessqlsrvS.a
Trace = yes
TraceFile = /tmp/easysoft_odbc
MARS_Connection = Yes
Threading = 0
FileUsage = 1
DontDLClose = 1
UsageCount = 2
```

Set up the ODBC dynamic loading library path

To load the ODBC driver libraries, set up the ODBC dynamic loading path.

For AIX platforms, add the ODBC driver library path to LIBPATH, as shown in the following example:
“setenv LIBPATH /usr/lib:/usr/local/lib:/usr/local/easysoft/sqlserver/lib:/usr/local/easysoft/lib:/usr/local/easysoft/unixODBC/lib”

For the HP Itanium platform, add the ODBC driver library path to SHLIB_PATH.
For Linux platforms, add the ODBC driver library path to the LD_LIBRARY_PATH.
You also need to set up the ODBCPATH environment variable, as shown in the following example:
“setenv ODBCPATH /usr/local/easysoft/unixODBC”
After you set the library path, restart the unirpcd daemon.

Create the EDA data source

Create an EDA data source in the EDA Schema Manager. Make sure you use the ODBC data source created in the previous step as the External DB Name.

Automatic data type mapping

Automatic data type mapping depends on which external data base to which you connect.
If you connect to Oracle, the data types are described in Oracle data type mapping, on page 41.
If you connect to SQL Server, the data types are described in SQL server data types, on page 44.
If you connect to DB2, the data types are described in DB2 data type mapping, on page 43.
Chapter 4: External database access driver API

EDA driver API

EDA enables you to convert data stored in the U2 database to a 1NF database, such as DB2, Oracle, and SQL Server, then access that data using existing UniVerse BASIC programs, RetrieVe, or UniVerse/UniData SQL.

**Note:** EDA was not designed to access data that already resides in a 1NF database. To access this type of data, use the UniVerse BASIC SQL Client Interface (BCI).

The EDA driver API enables you to write your own driver to access data in any relational database, such as Informix Dynamic Server. The EDA Driver API is a set of sixteen functions which EDA calls to communicate with an external database.

Registering an EDA driver

EDA drivers definitions reside as records in the EDA_DRIVER file, located in $UVHOME. The following table describes each attribute of the EDA_DRIVER record:

<table>
<thead>
<tr>
<th>Attribute number</th>
<th>Attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>@ID</td>
<td>The record ID for the EDA driver.</td>
</tr>
<tr>
<td>1</td>
<td>Data Model</td>
<td>The type of database to which the driver is connecting. At this release, the only valid value is 1NF.</td>
</tr>
<tr>
<td>2</td>
<td>DBMS Family</td>
<td>The name of the database management system family. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DB2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• U2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ORACLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Microsoft SQL Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Other</td>
</tr>
<tr>
<td>3</td>
<td>DBMS Name</td>
<td>The user-defined name and version of the database management system.</td>
</tr>
<tr>
<td>4</td>
<td>Description</td>
<td>A user-defined description of the EDA driver.</td>
</tr>
<tr>
<td>5</td>
<td>Driver Name</td>
<td>The name of the driver dll.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> Do not include the .dll extension when defining the name of the driver.</td>
</tr>
<tr>
<td>6</td>
<td>Driver Version</td>
<td>The version of the EDA driver.</td>
</tr>
<tr>
<td>7</td>
<td>Driver Supplier Name</td>
<td>The name of the supplier of the EDA driver.</td>
</tr>
<tr>
<td>8</td>
<td>Driver Creation Date</td>
<td>The date the EDA driver was created.</td>
</tr>
</tbody>
</table>
EDA driver functions

The following table lists the EDA driver functions.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDADRV_LoadSymbols</td>
<td>Loads other functions to the EDA Driver symbol array.</td>
</tr>
<tr>
<td>EDADRV_Connect</td>
<td>Connects to an external database.</td>
</tr>
<tr>
<td>EDADRV_Disconnect</td>
<td>Disconnects from an external database.</td>
</tr>
<tr>
<td>EDADRV_EndTransaction</td>
<td>Ends a transaction on an external database.</td>
</tr>
<tr>
<td>EDADRV_PrepareStmt</td>
<td>Prepares a statement.</td>
</tr>
<tr>
<td>EDADRV_ExecuteStmt</td>
<td>Executes an SQL statement that has been prepared with the EDADRV_PrepareStmt function.</td>
</tr>
<tr>
<td>EDADRV_CloseStmt</td>
<td>Closes a statement.</td>
</tr>
<tr>
<td>EDADRV_DropStmt</td>
<td>Closes a statement and makes it unavailable.</td>
</tr>
<tr>
<td>EDADRV_FetchStmt</td>
<td>Retrieves rows from an open cursor.</td>
</tr>
<tr>
<td>EDADRV_Perform</td>
<td>Executes a statement directly on the external database.</td>
</tr>
<tr>
<td>EDADRV_GetEDAAttr</td>
<td>Communicates an EDA attribute value to the driver.</td>
</tr>
<tr>
<td>EDADRV_GetErrMsg</td>
<td>Retrieves the last error message.</td>
</tr>
<tr>
<td>EDADRV_Cleanup</td>
<td>Releases memory, external handles, and the environment.</td>
</tr>
<tr>
<td>EDADRV_FreeResult</td>
<td>Frees the buffer allocated for the result set.</td>
</tr>
<tr>
<td>EDADRV_GetDBInfo</td>
<td>Retrieves information about the database.</td>
</tr>
<tr>
<td>EDADRV_GetSpecialInfo</td>
<td>Retrieves information about the database to which the application is connected, such as rename table, rename index, and BLOB data type equivalents.</td>
</tr>
</tbody>
</table>

**EDADRV_LoadSymbols**

The EDADRV_LoadSymbols function loads other functions to the EDA Driver Symbol array. This is the first function EDA calls.

The EDA Driver Symbol array is an array of structures of the type EDA_T_SYMBOL. It contains pointers to all other driver functions as array elements.

**Syntax**

```
RETCODE EDADRV_LoadSymbols(numsymbols, symbols)
```

**Output variables**

The following table describes the output variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *</td>
<td>numsymbols</td>
<td>The number of symbols returned.</td>
</tr>
<tr>
<td>EDA_T_SYMBOL *</td>
<td>symbols</td>
<td>The Driver Symbol Array pointer.</td>
</tr>
</tbody>
</table>

**Array template**

EDADRV_LoadSymbols allocates memory for the Driver Symbol array, fills in the array with either pointers to other driver functions or constants according to the following template.
EDADRV_Connect

EDA calls the EDADRV_Connect function to connect to an external database.
**Syntax**

```c
RETCODE EDADRV_Connect(connhdl, datasource, login_name, password)
```

**Input variables**

The following table describes the input variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char *</td>
<td>datasource</td>
<td>The name of the data source.</td>
</tr>
<tr>
<td>char *</td>
<td>login_name</td>
<td>The user login name.</td>
</tr>
<tr>
<td>char *</td>
<td>password</td>
<td>The password corresponding to the login name.</td>
</tr>
</tbody>
</table>

**Output variables**

The following table describes the output variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_CONN_HDL *</td>
<td>connhdl</td>
<td>The connection handle.</td>
</tr>
</tbody>
</table>

**Return codes**

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful.</td>
</tr>
<tr>
<td>1</td>
<td>EDADRV_ERR_SYSTEM</td>
<td>External system error.</td>
</tr>
<tr>
<td>101</td>
<td>EDADRV_ERR_MEMORY</td>
<td>Internal memory allocation error.</td>
</tr>
</tbody>
</table>

**EDADRV_Disconnect**

EDA calls the EDADRV_Disconnect function to disconnect from an external database and release the connection handle.

**Syntax**

```c
RETCODE EDADRV_Disconnect(connhdl)
```

**Input variables**

The following table describes the input variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_CONN_HDL</td>
<td>connhdl</td>
<td>The connection handle.</td>
</tr>
</tbody>
</table>

**Return codes**

The following table describes the return codes.
### EDADRV_EndTransaction

EDA calls the EDADRV_EndTransaction function to end a transaction on an external database. The EDADRV_EndTransaction function commits or rolls back a transaction on the external database, depending on the value of `trans_flag`.

#### Syntax

```plaintext
RETCODE  EDADRV_EndTransaction(connhdl, trans_flag)
```

#### Input variables

The following table describes the input variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_CONN_HDL</td>
<td><code>connhdl</code></td>
<td>The connection handle.</td>
</tr>
<tr>
<td>int</td>
<td><code>trans_flag</code></td>
<td>The action to take if the transaction ends. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 – EDA_COMMIT. Commit the transaction.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 – EDA_ROLLBACK. Rollback the transaction.</td>
</tr>
</tbody>
</table>

#### Return codes

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful.</td>
</tr>
<tr>
<td>1</td>
<td>EDADRV_ERR_SYSTEM</td>
<td>External system error.</td>
</tr>
<tr>
<td>102</td>
<td>EDADRV_INVALID_CONN_ID</td>
<td>Invalid connection handle.</td>
</tr>
<tr>
<td>114</td>
<td>EDADRV_INVALID_TRANS_FLAG</td>
<td>Invalid END TRANSACTION flag.</td>
</tr>
</tbody>
</table>

### EDADRV_PrepareStmt

EDA calls the EDADRV_PrepareStmt function to prepare a statement.

The EDADRV_PrepareStmt function prepares a statement passed to it by EDA. If the driver already has a statement handle that it can reuse, it may choose to return this preallocated handle in the `stmthdl` output variable, otherwise, it allocates a new statement handle and returns it in `stmthdl`.

If the statement is a DDL statement (statement type EDASTMT_DDL) or a DML statement, such as INSERT, UPDATE or DELETE (statement type EDASTMT_DML), or a SELECT statement (statement type
EDASTMT_QUERY), the statement may contain input parameters. These parameters are designated by parameter markers ("?"). In this case, EDA supplies as many parameter descriptions as there are parameter markers.

If the statement is a stored procedure call (statement type EDASTMT_PROCEDURE), the statement may contain input, output, and input/output parameters (parameter types of EDAPARAM_IN, EDAPARAM_OUT, and EDAPARAM_INOUT). In this case, EDA supplies as many parameter descriptions as there are input and input/output parameters of a stored procedure.

The EDARDV_PrepareStmt function allocates an array of EDA_T_PTYPE structures, converts EDA data type into the corresponding external database data type, and associates this array with the statement handle for its later use in EDARDV_ExecuteStmt.

Syntax

RETCODE EDARDV_PrepareStmt(connhdl, stmthdl, stmt)

Input variables

The following table describes the input variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_CONN_HDL</td>
<td>connhdl</td>
<td>The connection handle.</td>
</tr>
<tr>
<td>EDA_T_STMT</td>
<td>stmt</td>
<td>The statement content.</td>
</tr>
</tbody>
</table>

Output variables

The following table describes the output variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_STMT_HDL</td>
<td>stmthdl</td>
<td>The statement handle.</td>
</tr>
</tbody>
</table>

Return codes

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful.</td>
</tr>
<tr>
<td>1</td>
<td>EDARDV_ERR_SYSTEM</td>
<td>External system error.</td>
</tr>
<tr>
<td>101</td>
<td>EDARDV_ERR_MEMORY</td>
<td>Internal memory allocation error.</td>
</tr>
<tr>
<td>102</td>
<td>EDARDV_INVALID_CONN_ID</td>
<td>Invalid connection handle.</td>
</tr>
<tr>
<td>112</td>
<td>EDARDV_INVALID_DATATYPE</td>
<td>Invalid data type</td>
</tr>
</tbody>
</table>

EDARDV_ExecuteStmt

EDA calls the EDARDV_ExecuteStmt function to execute an SQL statement that has been prepared with EDA_PrepareStmt.

EDA provides parameter values for each input and input/output parameter. Parameter values are supplied in a string format, separated by field marks. The value of the field mark is determined by calling the EDARDV_GetADDAttr function. The EDARDV_ExecuteStmt function binds each parameter and executes a statement that has already been prepared with the EDARDV_PrepareStmt function.
If the statement is INSERT, UPDATE, or DELETE (statement type EDASTMT_DML), the output variable `rowcount` contains the number of rows affected by the operation. If the statement is a query (statement type EDASTMT_QUERY), `rowcount` contains the number of columns of the result set. If the statement is a stored procedure (statement type EDASTMT_PROCEDURE), `rowcount` is not set.

**Syntax**

```c
RETCODE EDADRV_ExecuteStmt(stmthdl, parameters, rowcount)
```

**Input variables**

The following table describes the input variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_STMT_HDL</td>
<td>stmthdl</td>
<td>The statement handle.</td>
</tr>
<tr>
<td>EDA_T_STRING</td>
<td>parameters</td>
<td>The statement parameters.</td>
</tr>
</tbody>
</table>

**Output variables**

The following table describes the output variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *</td>
<td>rowcount</td>
<td>The number of the rows affected by the INSERT, UPDATE, or DELETE statement, or the number of columns in the result set, or the values of output and input/output parameters of a stored procedure.</td>
</tr>
</tbody>
</table>

**Return codes**

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful</td>
</tr>
<tr>
<td>1</td>
<td>EDADRV_ERR_SYSTEM</td>
<td>External system error</td>
</tr>
<tr>
<td>2</td>
<td>EDADRV_SYSERR_OBJ_EXIST</td>
<td>Object already exists</td>
</tr>
<tr>
<td>101</td>
<td>EDADRV_ERR_MEMORY</td>
<td>Internal memory allocation error</td>
</tr>
<tr>
<td>103</td>
<td>EDADRV_INVALID_STATEMENT_ID</td>
<td>Invalid statement handle</td>
</tr>
<tr>
<td>111</td>
<td>EDADRV_INVALID_PARAM_TYPE</td>
<td>Invalid parameter type</td>
</tr>
<tr>
<td>112</td>
<td>EDADRV_INVALID_DATATYPE</td>
<td>Invalid data type</td>
</tr>
<tr>
<td>113</td>
<td>EDADRV_TOO_MANY_OUT_PARAM</td>
<td>Too many output parameters</td>
</tr>
</tbody>
</table>

**EDADRV_CloseStmt**

EDA calls the EDADRV_CloseStmt function to close a statement.

Once a cursor is opened by the execution of the `EDADRV_ExecuteStmt` function, it remains open even after all the rows have been fetched. The EDADRV_CloseStatement closes any open cursors associated with the statement handle.
**Warning:** The result buffer allocated by the EDADRV_FetchStmt function should not be freed by this function, it can only be freed by the EDADRV_FreeResult function.

**Syntax**

RETCODE EDADRV_CloseStmt(stmthdl)

**Input variables**

The following table describes the input variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_STMT_HDL</td>
<td>stmthdl</td>
<td>Statement handle.</td>
</tr>
</tbody>
</table>

**Return codes**

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful.</td>
</tr>
<tr>
<td>1</td>
<td>EDADRV_ERR_SYSTEM</td>
<td>External system error.</td>
</tr>
<tr>
<td>103</td>
<td>EDADRV_INVALID_STMT_ID</td>
<td>Invalid statement handle.</td>
</tr>
</tbody>
</table>

**EDADRV_DropStmt**

EDA calls the EDADRV_DropStmt function to close a statement and make it unavailable.

The EDADRV_DropStmt function closes any open cursors associated with the statement handle and makes the SQL statement unavailable for any future use.

**Warning:** The result buffer allocated by the EDADRV_FetchStmt function should not be freed by this function, it can only be freed by the EDADRV_FreeResult function.

**Syntax**

RETCODE EDADRV_DropStmt(stmthdl)

**Input variables**

The following table describes the input variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_STMT_HDL</td>
<td>stmthdl</td>
<td>Statement handle.</td>
</tr>
</tbody>
</table>

**Return codes**

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful.</td>
</tr>
</tbody>
</table>
EDADRV_FetchStmt

EDA calls the EDADRV_FetchStmt function to fetch rows from an open cursor.

The EDADRV_FetchStmt function fetches `numrows` rows from an open cursor. Currently, EDA only uses forward scrolling. EDA expects the result set to be returned in a string format. The rows of the result are separated with record marks (EDADRV_ATTR_RM) and column values within each row separated with the NULL character ("\0").

In order to hold the result set, the EDADRV_FetchStmt function allocates a buffer. This buffer can only be freed by the EDADRV_FreeResult function.

Syntax

```c
RETCODE EDADRV_FetchStmt(stmthdl, direction, numrows, rowsfetched, result)
```

Input variables

The following table describes the input variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_STMT_HDL</td>
<td>stmthdl</td>
<td>The statement handle.</td>
</tr>
<tr>
<td>int</td>
<td>direction</td>
<td>The fetch direction. Valid values are: 0 – Fetch Forward</td>
</tr>
<tr>
<td>int</td>
<td>numrows</td>
<td>The number of rows to fetch.</td>
</tr>
</tbody>
</table>

Output variables

The following table describes the output variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *</td>
<td>rowsfetched</td>
<td>The number of the row retrieved.</td>
</tr>
<tr>
<td>EDA_T_RESULT *</td>
<td>result</td>
<td>The result string.</td>
</tr>
</tbody>
</table>

Return codes

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful.</td>
</tr>
<tr>
<td>1</td>
<td>EDADRV_ERR_SYSTEM</td>
<td>External system error.</td>
</tr>
<tr>
<td>101</td>
<td>EDADRV_ERR_MEMORY</td>
<td>Internal memory allocation error.</td>
</tr>
<tr>
<td>103</td>
<td>EDADRV_INVALID_STMT_ID</td>
<td>Invalid cursor handle.</td>
</tr>
</tbody>
</table>
## EDADRVR_Perform

EDA calls the EDADRVR_Perform function to execute a statement directly on the external database.

The EDADRVR_Perform function combines the processing of the EDADRVR_PrepareStmt and the EDADRVR_Execute functions, and if the statement is a query, it also fetches the entire result set as in EDADRVR_FetchStmt. The driver designer may choose to either Prepare and Execute or Execute Direct on the external database side.

### Syntax

```
RETCODE EDADRVR_Perform(connhdl, stmt, numrows, result)
```

### Input variables

The following table describes the input variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_CONN_HDL</td>
<td>connhdl</td>
<td>The connection handle.</td>
</tr>
<tr>
<td>EDA_T_STMT</td>
<td>stmt</td>
<td>The statement content.</td>
</tr>
</tbody>
</table>

### Output variables

The following table describes the output variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *</td>
<td>numrows</td>
<td>The number of rows retrieved or affected.</td>
</tr>
<tr>
<td>EDA_T_RESULT *</td>
<td>result</td>
<td>The result string.</td>
</tr>
</tbody>
</table>

### Return codes

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful.</td>
</tr>
<tr>
<td>1</td>
<td>EDADRVR_ERR_SYSTEM</td>
<td>External system error.</td>
</tr>
<tr>
<td>2</td>
<td>EDADRVR_SYSERR_OBJ_EXIST</td>
<td>Object already exists.</td>
</tr>
<tr>
<td>101</td>
<td>EDADRVR_ERR_MEMORY</td>
<td>Internal memory allocation error.</td>
</tr>
<tr>
<td>102</td>
<td>EDADRVR_INVALID_CONN_ID</td>
<td>Invalid connection handle.</td>
</tr>
<tr>
<td>106</td>
<td>EDADRVR_TOO_MANY_DATA</td>
<td>Too much data fetched.</td>
</tr>
<tr>
<td>112</td>
<td>EDADRVR_INVALID_DATATYPE</td>
<td>Invalid data type</td>
</tr>
<tr>
<td>113</td>
<td>EDADRVR TOO MANY_OUT_PARAM</td>
<td>Too many output parameters.</td>
</tr>
</tbody>
</table>
EDA calls the `EDADRV_GetEDAAttr` function to communicate an EDA attribute value to the driver. The `EDADRV_GetEDAAttr` function receives an attribute name – value pair.

**Syntax**

```c
RETCODE EDADRV_GetEDAAttr(attribute_type, value)
```

**Input variables**

The following table describes the input variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td><code>attribute_type</code></td>
<td>The name of the EDA driver attribute.</td>
</tr>
<tr>
<td>EDA_T_SYMBOL</td>
<td><code>value</code></td>
<td>The value of the EDA driver attribute.</td>
</tr>
</tbody>
</table>

Valid values for the EDA driver attribute are:

<table>
<thead>
<tr>
<th>Type</th>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>EDADR_ENV</code></td>
<td><code>EDADR_ATTR_SYSNAME</code></td>
<td>At this release, the only valid values are UniVerse.</td>
</tr>
<tr>
<td><code>EDADR_ENV</code></td>
<td><code>EDADR_ATTR_VERSION</code></td>
<td>Valid values are 7.1 or 7.2.</td>
</tr>
<tr>
<td><code>EDADR_ENV</code></td>
<td><code>EDADR_ATTR_RM</code></td>
<td>The ASCII character representing a record mark (RM), such as ^255.</td>
</tr>
<tr>
<td><code>EDADR_ENV</code></td>
<td><code>EDADR_ATTR_FMR</code></td>
<td>The ASCII character representing a field mark (FM), such as ^254.</td>
</tr>
<tr>
<td><code>EDADR_ENV</code></td>
<td><code>EDADR_ATTR_VM</code></td>
<td>The ASCII character representing a value mark (VM), such as ^253.</td>
</tr>
<tr>
<td><code>EDADR_ENV</code></td>
<td><code>EDADR_ATTR_SM</code></td>
<td>The ASCII character representing a subvalue mark (SM), such as ^252.</td>
</tr>
<tr>
<td><code>EDADR_ENV</code></td>
<td><code>EDADR_ATTR_TM</code></td>
<td>The ASCII character representing a text mark (TM), such as ^251.</td>
</tr>
<tr>
<td><code>EDADR_ENV</code></td>
<td><code>EDADR_NULLVAL</code></td>
<td>The ASCII character representing the null value.</td>
</tr>
</tbody>
</table>

**Return codes**

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful.</td>
</tr>
<tr>
<td>105</td>
<td><code>EDADR_INVALID_DRV_ATTR</code></td>
<td>Invalid driver attribute.</td>
</tr>
</tbody>
</table>
EDADRV_GetErrMsg

EDA calls the EDADRV_GetErrMsg function to retrieve the last error message.

If the last driver function returned an error code, EDA calls this function to retrieve the corresponding error message string. If the error is returned from the external database, the driver returns this external database error. Otherwise, the driver should generate its own error message.

**Syntax**

```c
RETCODE EDADRV_GetErrMsg(errmsg)
```

**Output variables**

The following table describes the output variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_RESULT *</td>
<td>errmsg</td>
<td>The error message string.</td>
</tr>
</tbody>
</table>

**Return codes**

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful.</td>
</tr>
<tr>
<td>1</td>
<td>EDADRV_ERR_SYSTEM</td>
<td>External system error.</td>
</tr>
<tr>
<td>101</td>
<td>EDADRV_ERR_MEMORY</td>
<td>Internal memory allocation error.</td>
</tr>
</tbody>
</table>

EDADRV_Cleanup

EDA calls the EDADRV_Cleanup function to release memory, external handles, and the environment.

This is the last function that EDA calls. This function frees all allocated memory and all handles to the external database, closes all files, and frees the driver environment.

**Syntax**

```c
RETCODE EDADRV_Cleanup
```

**Return codes**

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful.</td>
</tr>
</tbody>
</table>

EDADRV_FreeResult

EDA calls the EDADRV_FreeResult function to free the buffer allocated for the result set.
The buffer allocated by the EDADRV_FetchStmt or the EDADRV_Perform function is not freed until EDA calls the EDADRV_FreeResult function. The EDADRV_FreeResult function frees the result set buffer.

**Syntax**

```plaintext
RETCODE EDADRV_FreeResult(buf)
```

**Input variables**

The following table describes the input variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_RESULT</td>
<td>buf</td>
<td>The result buffer</td>
</tr>
</tbody>
</table>

**Return codes**

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful.</td>
</tr>
</tbody>
</table>

**EDADRV_GetDBInfo**

EDA calls the EDADRV_GetDBInfo function to retrieve information about the database to which the application is connected.

**Syntax**

```plaintext
RETCODE EDADRV_GetDBInfo(connhdl, infotype, dbinfo)
```

**Input variables**

The following table describes the input variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_CONN_HDL</td>
<td>connhdl</td>
<td>The connection handle.</td>
</tr>
<tr>
<td>int</td>
<td>infotype</td>
<td>The information type. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- EDADRV_DBMS_NAME – The name of the database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- EDADRV_DBMS_VERSION – The database version.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- EDADRV_SERVER_NAME – The name of the instance on the external database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- EDADRV_DATABASE_NAME – The name of the database.</td>
</tr>
</tbody>
</table>

**Output variables**

The following table describes the output variables.
### Chapter 4: External database access driver API

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_RESULT *</td>
<td>dbinfo</td>
<td>The database information.</td>
</tr>
</tbody>
</table>

#### Return codes

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful.</td>
</tr>
<tr>
<td>1</td>
<td>EDADRV_ERR_SYSTEM</td>
<td>External system error.</td>
</tr>
<tr>
<td>101</td>
<td>EDADRV_ERR_MEMORY</td>
<td>Internal memory allocation error.</td>
</tr>
<tr>
<td>102</td>
<td>EDADRV_INVALID_CONN_ID</td>
<td>Invalid connection handle.</td>
</tr>
<tr>
<td>115</td>
<td>EDADRV_INVALID_INFOTYPE</td>
<td>Invalid information type.</td>
</tr>
</tbody>
</table>

#### EDADRV_GetSpecialInfo

EDA calls the EDADRV_GetSpecialInfo function to retrieve special information about the database to which it is currently connected, such as rename table, rename index, and BLOB data type equivalents.

EDA calls the EDADRV_GetSpecialInfo function to retrieve special information about the database to which it is currently connected, such as rename table, rename index, and data types of the external database. It also retrieves information for this driver.

When you specify EDADRV_BLOB_FIELD as the infotype, do not specify the parameter input variable. The output parameter returns the data type for the BLOB field on the external database, followed by the separator '\0' and '0' or '1' for precision. For example, DB2 returns 'CLOB\01' and SQL Server returns 'VARCHAR(MAX)\00.'

When you specify EDADRV_RENAME_TABLE or EDADRV_RENAME_INDEX as the infotype, you must also specify both the source table or index name and the target table or index name, separated by "\0." The output variable returns the rename table or rename index statement from the external database.

When you specify EDADRV_DRIVER_INFO as the infotype, do not specify the parameter variable. The output parameter returns EDA driver information, including the EDA driver version, the supplier of the EDA driver, the date the EDA driver was created, the EDA driver target database name, and the EDA driver target database version.

**Syntax**

```
RETCODE EDADRV_GetSpecialInfo(infotype, parameters, dbinfo)
```

**Input variables**

The following table describes the input variables.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connhdl</td>
<td>The connection handle.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| **infotype** | The information type. Valid values are:  
  - EDADRV_DATA_TYPE – The data type for the external database. EDADRV_DATA_TYPE returns each data type followed by precision.  
  - EDADRV_RENAME_TABLE – The rename table statement.  
  - EDADRV_RENAME_INDEX – The rename index statement.  
  - EDADRV_DRIVER_INFO – returns the driver information, including the EDA driver API version, the EDA driver version, the EDA driver supplier name, the EDA driver creation date, the EDA driver target database name, and the EDA driver target database version. Each value is separated by “\0.” |

**parameters** | The parameter array. |

### Output variables

The following table describes the output variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_T_RESULT *</td>
<td>dbinfo</td>
<td>The database information.</td>
</tr>
</tbody>
</table>

### Return codes

The following table describes the return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Successful.</td>
</tr>
<tr>
<td>101</td>
<td>EDADRV_ERR_MEMORY</td>
<td>Internal memory allocation error.</td>
</tr>
<tr>
<td>115</td>
<td>EDADRV_INVALID_INFOTYPE</td>
<td>Invalid information type.</td>
</tr>
</tbody>
</table>

### ECL command

Use the **EDA.VERSION** command to retrieve information about the EDA Driver.

**Syntax**

EDA.VERSION <datasource>

where **datasource** is the name of the external data source.

The **EDA.VERSION** command returns the following information:

- The driver target database name
- The driver target database version
- The supplier of the driver
- The version of the driver
- The data the driver was created
EDA driver log files

The logging level for the EDA Common driver, the EDA Oracle driver, the EDA SQL Server driver, and the EDA DB2 driver can be set to values between 0 and 8.

The definition for each log level is described in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No log</td>
</tr>
<tr>
<td>1</td>
<td>Only report errors with timestamps</td>
</tr>
<tr>
<td>2</td>
<td>Log EDA driver functions that only show SQL statements, not read/write data</td>
</tr>
<tr>
<td>3</td>
<td>Log EDA driver functions and calling external database APIs that only show SQL statements, not read/write data</td>
</tr>
<tr>
<td>4</td>
<td>Log EDA driver functions that show SQL statements and update data, not read data</td>
</tr>
<tr>
<td>5</td>
<td>Log EDA driver functions and calling external database APIs that show SQL statements and update data, not read data</td>
</tr>
<tr>
<td>6</td>
<td>Log all EDA driver functions, show SQL statements, and read/write data</td>
</tr>
<tr>
<td>7</td>
<td>Log all EDA driver functions and calling external database APIs, show SQL statements, and read/write data, but for looping get data, only show once</td>
</tr>
<tr>
<td>8</td>
<td>Log all EDA driver functions and calling all external database APIs</td>
</tr>
</tbody>
</table>

The $UVHOME/logs/eda.errlog file is also available to collect information about server-side EDA errors.

The EDA driver header file

```c
int len;/* length of data content */
int buflen;/* length of buffer */
char * buf;/* buffer pointer */
} EDA_T_STRING;/* for difndef edadrv_public_H_DEFINED
#define edadrv_public_H_DEFINED
/*
* edadriv_public.h 7.2 Jun. 19, 2007
* *
* Version: 1.0
*/
/*
* The EDA driver symbols and prototypes:
* *
* EDA Drivers are DLLs to deal with external databases. All drivers must
* have a list symbols defined. Symbols are either data symbol or function
* symbols. Data symbols represent global variables and function symbols
* represent the C functions of the driver. All symbols are defined in
* Driver Symbol Codes. It’s allowed that some drivers do not support some
* specific symbols. In this case, Any access to these symbols will
* generate a error to Upper layer.
* All symbols will be loaded into Driver Dynamic Symbol Array after
* driver DLL is loaded by Driver Manager through EDADRIV_LOAD_SYMBOLS.
* EDADRIV_LOAD_SYMBOLS is a function defined in driver. It is responsible
* to allocate the Driver Dynamic Symbol Array and load the symbols in
* the order defined in Driver Symbol Codes.
* *
* This header file defines the driver symbols and prototypes. All data
```
* types of function symbols are also defined, as well as error codes it
* returned.
*/

**************************************************************************
***
* Section One: Definition of Datatypes
**************************************************************************

/**
typedef int (* T_FUNC)(); /* Datatype of function pointer */
/**************************** Datatype of Driver Symbols *********************/
typedef struct EDA_T_SYMBOL {
  int symtype; /* Symbol type, defined below */
  union {
    T_FUNC func; /* Pointer to driver functions */
    char * strparam; /* Pointer to driver string parameters */
    int intparam; /* Pointer to driver integer parameters */
  } sym; /* The symbol */
  char * description; /* Description of the symbol */
} EDA_T_SYMBOL;
/******************** Symbol Type Values *******************/
#define EDAT_FUNC0
#define EDAT_INT1
#define EDAT_STRING2
/******************** Datatype of EDA String *******************/
typedef struct EDA_T_STRING {
  ata transferring between driver API */
typedef EDA_T_STRING EDA_T_RESULT;
/******************** Datatype of EDA Statement *******************/
typedef struct EDA_T_PTYPE {
  short inout; /* IN/OUT */
  short datatype; /* parameter datatype */
} EDA_T_PTYPE; /* define a parameter type */
/******************** Parameter IN/OUT *******************/
#define EDAPARAM_IN0 /* Input parameter */
#define EDAPARAM_OUT1 /* Output parameter */
#define EDAPARAM_INOUT2 /* In/Out Parameter */
/******************** Parameter Types *******************/
#define EDAPARAM_CHAR1
#define EDAPARAM_VARCHAR2
#define EDAPARAM_INTEGER3
#define EDAPARAM_LONG4
#define EDAPARAM_SHORT5
#define EDAPARAM_FLOAT6
#define EDAPARAM_DOUBLE7
#define EDAPARAM_DATE8
#define EDAPARAM_TIME9
#define EDAPARAM_TIMESTAMP10
#define EDAPARAM_CLOB11
#define EDAPARAM_BLOB12
#define EDAPARAM_NUMERIC13
#define EDAPARAM_DECIMAL14
typedef struct EDA_T_STMT {
  short type; /* statement type, defined below. */
  short flags; /* statement flags, defined below. */
  char * stmt; /* statement buffer, stmt is null ended */
  int numparam; /* number of question marks in stmt */
  EDA_T_PTYPE * paramdef; /* array of parameter defs */
} EDA_T_STMT; /* EDA statement definition */
/******************** Statement Types *******************/
#define EDASTMT_SQL_DDL1 /* SQL DDL statement */
#define EDASTMT_SQL_DML2 /* SQL DML statement */
#define EDASTMT_SQL_QUERY3 /* SQL QUERY statement */
Chapter 4: External database access driver API

#define EDASTMT_PROCEDURE 4 /* Stored Procedure */
#define EDASTMT_XQUERY 5 /* XQUERY statement */

/************************** Definition of Driver Connection Handle **************/
typedef long EDA_T_CONN_HDL;
#define INVALID_CONN_HDL(hdl) ((hdl)<0)
#define CLEAR_CONN_HDL(hdl) ((hdl)=-1)/* Clear a handle */

/************************** Definition of Driver Statement Handle **************/
typedef long EDA_T_STMT_HDL;/* Positive integer */
#define INVALID_STMT_HDL(hdl) ((hdl)<0)/* validation of handle */
#define CLEAR_STMT_HDL(hdl) ((hdl)=-1)/* Clear handle */

/******************************************************************************/
/* Section 2: Definition of Driver Loading API and Dynamic Symbol Array */
/******************************************************************************/

#define EDADRV_LoadSymbols "EDA_driver_load_symbols" /* Load driver symbol */

extern int EDA_driver_load_symbols();

/* FUNCTION: EDADRV_LoadSymbols:Load EDA Driver Symbol Array. */
/* DESCRIPTION: This is the function called after a driver DLL is loaded.
 * The driver should allocate the Driver Symbol Array and 
 * load the driver symbols into the array according to Driver Symbol 
 * Code. It can invoke driver initialization if needed. 
 * CALLS BY : Driver Manager after loading the driver DLL. 
 * Parameters 
 * int * numsymbols; OUT: The number of symbols returned. 
 * EDA_T_SYMBOL *symbols;OUT:The Driver Symbol Array pointer. 
 * RETURN : int 
 * 0: Succeeded; 
 * errcode if failed. 
 * GLOBAL : 
 */
#define EDA_SYMBOL_NUM 17

/************************** Driver Symbol Codes **************************/
#define EDA_DBTYPE0 /* Integer: Driver Data Type code */
#define EDA_DBFAMILY1 /* Integer: Driver DBMS Family code */
#define EDA_VERSION2 /* String: the version string */
#define EDA_CONNECT3 /* Function: Make Connection */
#define EDA_DISCONNECT4 /* Function: Disconnect */
#define EDA_END_TRANSACTION5 /* Function: End Transaction */
#define EDA_PREPARE_STMT6 /* Function: Prepare Statement */
#define EDA_DROP_STMT7 /* Function: Drop Statement */
#define EDA_EXECUTE_STMT8 /* Function: Execute Statement */
#define EDA_CLOSE_STMT9 /* Function: Close Statment */
#define EDA_FETCH_STMT10 /* Function: Fetch Statement */
#define EDA_PERFORM11 /* Function: Perform a command */
#define EDA_GET_EDA_ATTR12 /* Function: Get EDA attribute */
#define EDA_GET_ERRMSG13 /* Function: Get last error message */
#define EDA_CLEANUP14 /* Function: Cleanup the driver */
#define EDA_GET_DBINFO15 /* Function: Get external DB info */
#define EDA_FREE_RESULT16 /* Function: Free result buffer */
#define EDA_GET_SPECIALINFO17 /* Function: Get special info */

/************************** EDASYM_DBTYPE Code: Driver Data Type *************/
#define EDA_UNKNOW 0
#define EDAoftware 1 /* Relational Database */
#define EDA_NF2 2 /* Multi-valued Database */
#define EDA_XML 3 /* XML Database */
#define EDA_OBJ 4 /* Object Database */

/************************** EDASYM_DBFAMILY Code: Driver DBMS Family *************/
#define DBMS_OTHERS0/* Other DBMS */
#define DBMS_DB21/* DB2 family */
#define DBMS_ORACLE2/* Oracle family */
#define DBMS_MSSQL3/* Microsoft SQL Server family */
#define DBMS_U24/* IBM U2 family */

/*********************************************************
* Section 3: Definition of EDA API Functions
**********************************************************/

**FUNCTION SYMBOL: EDASYM_CONNECT**
**FUNCTION : EDADRV_Connect: Make connection to external database.**
**DESCRIPTION:**
**CALLS BY :** Connection Manager per request.
**PARAMETERS :**
EDA_T_CONN_HDL * connhdl;OUT: The connection handle.
char * datasource;IN: The datasource name.
char * loginname;IN: login user name.
char * password;IN: login password.
**RETURN :**
0: Succeeded;
errcode if failed.
**GLOBAL :**

**FUNCTION SYMBOL: EDASYM_DISCONNECT**
**FUNCTION : EDADRV_Disconnect: Disconnect a connection.**
**DESCRIPTION:**
**CALLS BY :** Connection Manager per request.
**PARAMETERS :**
EDA_T_CONN_HDL connhdl;IN: the connection handle.
**RETURN :**
0: Succeeded;
errcode if failed.
**GLOBAL :**

**FUNCTION SYMBOL: EDASYM_END_TRANSACTION**
**FUNCTION : EDADRV_EndTransaction, Ending a transaction on a connection.**
**DESCRIPTION:** Either Commit or Rollback a transaction.
**CALLS BY :** Transaction Control Module.
**PARAMETERS :**
EDA_T_CONN_HDL connhdl;IN: the connection handle
int trans_flag;IN: Commit or Rollback.
**RETURN :**
0: Succeeded;
errcode if failed.
**GLOBAL :**

********** Transaction flag for EDADRV_EndTransaction **********/
#define EDA_COMMIT0
#define EDA_ROLLBACK1

**FUNCTION SYMBOL: EDASYM_PREPARE_STMT**
**FUNCTION : EDADRV_PrepareStmt: Prepare a statement.**
**DESCRIPTION:** Driver create a statement by given connection handle
and a statement string. Statement string may include Question Marks
for later parameter input. A list of parameter types is specified for
later execution. The number of arguments must match the number of
Question Marks, otherwise it will cause errors in the execution.
If the prepared statement is a stored procedure, the values of input
parameters will be passed in by EDASYM_ExecuteStmt, and the results
of output parameters will be passed out by EDASYM_FetchStmt.
* NOTE: Driver can use a pre-allocated handle to prepare the
* statement; or allocate a new handle every time according to its
* implementation.
* CALLS BY : Transaction Control Module.
* PARAMETERS :
* EDA_T_CONN_HDL connhdl;IN: connection handle.
* EDA_T_STMT_HDL *stmthdl;OUT: the statement handle.
* EDA_T_STMT stmt;IN: the statement content.
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*********************************************************************/

FUNCTION SYMBOL: EDASYM_EXECUTE_STMT
FUNCTION : EDADRV_ExecuteStmt:
DESCRIPTION: EDADRV_ExecuteStmt open the execution of a statement
by given parameters. The statement must have been prepared and the
number and types of parameters given here must match those in
preparing. Parameters are separated by FM. If the statement handle is
still open, EDADRV_ExecuteStmt will close the old one and re-open it
with the given parameters.
* CALLS BY : Transaction Control Module
* PARAMETERS :
* EDA_T_STMT_HDL stmthdl;IN: Statement handle
* EDA_T_STRING parameters;IN: Statement parameters
* int*rowcount;OUT: number of row affected
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*********************************************************************/

FUNCTION SYMBOL: EDASYM_CLOSE_STMT
FUNCTION: EDADRV_CloseStmt: Close the execution of a statement.
DESCRIPTION: A statement is opened after calling EDADRV_ExecuteStmt
and will keep opening even though all rows has been fetched by
EDADRV_FetchStmt. EDADRV_CloseStmt close the execution of the
statement.
* CALLS BY : Transaction Control Module
* PARAMETERS :
* EDA_T_STMT_HDL stmthdl; IN: the statement handler
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*********************************************************************/

FUNCTION SYMBOL: EDASYM_DROP_STMT
FUNCTION : EDADRV_DropStmt: Drop a statement.
DESCRIPTION: EDADRV_DropStmt drop a statement and make it unavailable.
If a statement is opened EDADRV_DropStmt will close it before
dropping it.
* CALLS BY : Transaction Control Module
* PARAMETERS :
* EDA_T_STMT_HDL stmthdl; IN: the statement handler
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
The EDA driver header file

******************************************************************************
* FUNCTION SYMBOL: EDASYM_FETCH_STMT
* FUNCTION: EDADRV_FetchStmt: Fetch an open statement
* DESCRIPTION: EDA always treat driver statement as scrollable cursor.
* The format of result differs on the driver's data type:
* EDA_1NF: Result is rows returned by SQL statement. Columns
* are separated by '\0' and rows are separated by RM.
* EDA_XML: Result is XML document.
* EDA_NF2: Result is multi-valued records separated by RM.
* EDA_OBJ: Result is self-defined.
* CALLS BY: Transaction Control Module
* PARAMETERS:
* EDA_T_STMT_HDL stmthdl;IN: statement handle.
* int direction;IN: fetch direction.
* int numrows;IN: number of rows to fetch.
* int *rowsfetched;OUT: number of rows fetched.
* EDA_T_RESULT *result;OUT: the result string.
* RETURN:
* 0: Succeeded;
* errcode if failed.
* GLOBAL:
******************************************************************************
/******************** Fetch Direction ********************/
#define EDA_FETCH_FORWARD0
#define EDA_FETCH_BACKWARD1
#define EDA_FETCH_FIRST2
#define EDA_FETCH_LAST3
******************************************************************************
* FUNCTION SYMBOL: EDASYM_PERFORM
* FUNCTION: EDADRV_Perform: Perform a statement directly onto
* external DBMS.
* DESCRIPTION: The statement could be a DDL, DML or SQL query without
* question marks. If the statement is a stored procedure, the question
* marks in the statements must represent output parameters.
* If the statement has result returned, the format of returned
* result is same as those defined in EDA_FetchStmt.
* CALLS BY: Transaction Control Module
* PARAMETERS:
* EDA_T_CONN_HDL connhdl;IN: connection handle.
* EDA_T_STMT stmt;IN: statement content.
* int *numrows;OUT: number of rows fetched or affected.
* EDA_T_RESULT *result;OUT: the result string.
* RETURN:
* 0: Succeeded;
* errcode if failed.
* GLOBAL:
******************************************************************************
/******************** Driver Attribute Names ********************/
#define EDADRV_ATTR_SYSNAME1/* EDADRV_T_STR: UniData or UniVerse*/
#define EDADRV_ATTR_VERSION2/* EDADRV_T_STR: U2 Version string */
#define EDADRV_ATTR_RM3/* EDADRV_T_INT: U_RM */
#define EDADRV_ATTR_FM4/* EDADRV_T_INT: U_FM */
#define EDADRV_ATTR_VM5/* EDADRV_T_INT: U_VM */
#define EDADRV_ATTR_SM6/* EDADRV_T_INT: U_SM */
#define EDADRV_ATTR_TM7/* EDADRV_T_INT: U_TM */
#define EDADRV_ATTR_NULLVAL8/* EDADRV_T_INT: U_NULLVAL */

/*============================================================================
* FUNCTION SYMBOL: EDASYM_GET_ERRMSG
* FUNCTION : EDADRV_GetErrmsg: Get last error message string.
* DESCRIPTION: if error code returned from a driver API, the calling
* module can call this function to get the detailed error message.
* If the error reported from external DBMS, the driver will return the
* system error message; Otherwise, driver should generate it's own error
* message.
* PARAMETERS :
* EDA_T_RESULT * errmsg;OUT: Error message string.
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
/*============================================================================

/*============================================================================
* FUNCTION SYMBOL: EDASYM_CLEANUP
* FUNCTION : EDADRV_Cleanup: Cleanup the driver.
* DESCRIPTION: Driver should free all allocated memory segement and all
* handlers to external DBMS client; close all opened files and
* cleanup driver environment.
* PARAMETERS :
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
/*============================================================================

/*============================================================================
* FUNCTION SYMBOL: EDASYM_FREE_RESULT
* FUNCTION : EDADRV_FreeResult: Free memory space allocated by driver.
* DESCRIPTION: Driver should free memory allocated by itself.
* PARAMETERS :
* EDA_T_RESULT buf;IN: the result buffer
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
/*============================================================================

/*============================================================================
* FUNCTION SYMBOL: EDASYM_GET_DBINFO
* FUNCTION : EDADRV_GetDBInfo: Get database information.
* DESCRIPTION: Get general information about the DBMS that the
* application is currently connected to.
* CALLED BY: Driver Manager
* PARAMETERS :
* EDA_T_CONN_HDL connhdl;IN: connection handle.
* intinfotype;IN: information type
* EDA_T_RESULT *dbinfo;OUT: the DB info.
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
/*============================================================================

/*============================================================================
* Infotypes in EDADRV_GetDBInfo */
The EDA header file

#define EDADRV_DBMS_NAME1 /* DBMS name */
#define EDADRV_DBMS_VERSION2 /* DBMS version */
#define EDADRV_SERVER_NAME3 /* DB2 instance */
#define EDADRV_DATABASE_NAME4 /* Database name */

/***************************************************************************/
* FUNCTION SYMBOL: EDASYM_GET_SPECIALINFO
* FUNCTION : EDADRv_GetSpecialInfo: Get special information
* for external database or driver.
* DESCRIPTION : Get special information about the DBMS that the
* application is currently connected to, like rename
* table, rename index, blob data type. Also get the
* information for this driver.
* CALLED BY : Driver Manager
* PARAMETERS :
* int infotype; IN: information type
* EDA_T_STRING parameters; IN: parameter array
* EDA_T_RESULT *spinfo; OUT: the special info.
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
***************************************************************************/

/******************************* Infotypes in EDADRv_GetSpecialInfo *******************/
#define EDADRv_BLOB.Field1 /* data type for BLOB field */
#define EDADRv_RENAME_TABLE2 /* get rename table statement */
#define EDADRv_RENAME_INDEX3 /* get rename index statement */
#define EDADRv_DRIVER_INFO4 /* EDA driver info. */
/* More to be defined */

/******************************* Driver Error Codes ***************************/
#define EDADRv_ERR_SYSTEM1 /* General External system error */
#define EDADRv_SYSERR_OBJ_EXIST2 /* Object existing */
#define EDADRv_ERR_MEMORY101 /* Internal memory allocation error */
#define EDADRv_INVALID_CONN_ID102 /* Invalid connection handle */
#define EDADRv_INVALID_STMT_ID103 /* Invalid statement handle */
#define EDADRv_INVALID_FETCH_DIR 104 /* Invalid fetch direction */
#define EDADRv_INVALID_EDA_ATTR105 /* Invalid EDA attribute */
#define EDADRv_TOO_MANY_DATA106 /* Too many data fetched */
#define EDADRv_GET_UDDHOME_ERROR 107 /* Get UDDHOME error */
#define EDADRv_CONFIG_NOT_EXIST108 /* edaconfig file not exist */
#define EDADRv_OPEN_CONFIG_ERROR 109 /* Open edaconfig error */
#define EDADRv_DB2_INSTANCE_NOT_SET110 /* DB2INSTANCE not set */
#define EDADRv_INVALID_PARAM_TYPE111 /* Invalid parameter type */
#define EDADRv_INVALID_DATATYPE112 /* Invalid data type */
#define EDADRv_TOO_MANY_OUT_PARAM113 /* Too many output parameters */
#define EDADRv_INVALID_TRANS_FLAG114 /* Invalid end transaction flag */
#define EDADRv_INVALID_INFO_TYPE115 /* Invalid information type */
/* More to be defined */
#endif
Chapter 5: EDA TCL commands

This chapter describes TCL commands you can use to connect to external databases and to convert U2 data to an external database, to verify your EDA Schema, to list your EDA Schema, to save your EDA Schema, and view nonconforming U2 records.

EDA.CONNECT

Use the EDA.CONNECT command to connect your EDA system to the external data source. You may want to use this command if you want to connect using a log on ID and password different from the default.

If you issue the EDA.CONNECT command, U2 maintains the connection until you issue the EDA.DISCONNECT command.

Syntax

EDA.CONNECT datasource [WITH logon_name[, password]]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datasource</td>
<td>The name of the data source to which you are connecting. The data source must exist in the EDA_DATASOURCE file.</td>
</tr>
<tr>
<td>WITH logon_name, password</td>
<td>The logon name on the external data source. If you do not specify logon_name, U2 searches the EDA_DATASOURCE file for a qualified user. If you specify logon_name without password, U2 searches the Connection Password file and connects with logon_name and that password. If you specify both logon_name and password, U2 uses both the make the connection.</td>
</tr>
</tbody>
</table>

EDA.CONVERT

Use the EDA.CONVERT command to convert U2 data to the external database based on an EDA Schema. The conversion results in an EDA Object Set on the external database. An EDA file replaces the original U2 file in the U2 database.

If the U2 file you are converting is an EDA file, the conversion process removes the file and creates the new EDA file. If the file exists but is not an EDA file, the conversion process renames the file as filename.edasave and creates the new EDA file.

The conversion process copies data, trigger, and index information to the new EDA file.

**Note:** The EDA.CONVERT command is not allowed on a replicated file when U2 Data Replication is running. Before using EDA.CONVERT, you must suspend U2 Data Replication. When the EDA.CONVERT command has completed, restart and sync U2 Data Replication using the sync operation.
EDA.DISCONNECT

Use the EDA.DISCONNECT command to disconnect from the external data source.

Syntax

EDA.DISCONNECT datasource

where datasource is the external data source from which you want to disconnect.

EDA.EXCEPTION

Use the EDA.EXCEPTION command to utilize the EDA_EXCEPTION file.

For more information about this file, see The EDA_EXCEPTION file, on page 76.

Syntax

EDA.EXCEPTION [ON | OFF]

By default, EDA.EXCEPTION is set to ON.
Chapter 5: EDA TCL commands

**EDA.LOG**

Use the EDA.LOG command to write a text log file to C:\U2\uv113\uvtemp. The format of the log file is EDA_EDATEST_nnnn where EDATEST is the account name and nnnn is the pid.

**Syntax**

EDA.LOG [ON {1|2|3} | OFF]

A value of 1 logs for EDA connection and mapping functions.

A value of 2 logs level 1 and EDA data access functions.

A value of 3 logs level 2 and EDA engine calling EDA driver functions.

**EDA.VERSION**

Use the EDA.VERSION command to retrieve information about the EDA Driver.

**Syntax**

EDA.VERSION datasource

where datasource is the name of the external data source.

The EDA.VERSION command returns the following information:

- The driver target database name
- The driver target database version
- The supplier of the driver
- The version of the driver
- The data the driver was created

**LIST.EDAMAP**

The LIST.EDAMAP command displays the EDA Schema you specify.

**Syntax**

LIST.EDAMAP {[XMAP] eda_schema | EDA.FILE [DICT] eda_file | DEFAULT.MAP} [DATASOURCE data_source] [OBJECT.SET [name_space.]primary_table] [FILE.NAME target_file] [XMAP | OBJECT.TREE | DLL]

**Parameters**

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eda_schema</td>
<td>Specifies the name of the EDA schema to display.</td>
</tr>
<tr>
<td>eda_file</td>
<td>Specifies the name of the EDA file whose schema is to be extracted and displayed. If you specify FILE.NAME target_file, target_name replaces the U2 file name in the schema U2 displays.</td>
</tr>
</tbody>
</table>
SAVE.EDAMAP

The SAVE.EDAMAP command saves the EDA schema in a schema file in either the EDAMAP or EDAXMAP format.

Syntax

```
SAVE.EDAMAP{[XMAP] eda_schema | EDA.FILE [DICT] eda_file | DEFAULT.MAP}
[DATASOURCE data_source] [OBJECT.SET [name_space.]primary_table]
[FILENAME target_file] TO [XMAP] <schema_name>
```

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eda_schema</td>
<td>Specifies the name of the EDA schema to save.</td>
</tr>
<tr>
<td>eda_file</td>
<td>Specifies the name of the EDA file whose schema is to be saved.</td>
</tr>
<tr>
<td></td>
<td>If you specify FILE.NAME target_file, target_name replaces the</td>
</tr>
<tr>
<td></td>
<td>U2 file name in the schema U2 displays.</td>
</tr>
<tr>
<td>DEFAULT.MAP</td>
<td>Specifies to only save the primary key (@ID) mapping, irrespective of the</td>
</tr>
<tr>
<td></td>
<td>attributes actually mapped of the schema you specify.</td>
</tr>
<tr>
<td>data_source</td>
<td>Specifies the data source name to use when saving the schema.</td>
</tr>
<tr>
<td>primary_table</td>
<td>Specifies the name of the primary table, containing only singlevalued</td>
</tr>
<tr>
<td></td>
<td>attributes to use when saving the schema.</td>
</tr>
<tr>
<td></td>
<td>If you also specify name_space, U2 uses it for Name Space (external</td>
</tr>
<tr>
<td></td>
<td>Schema Name).</td>
</tr>
<tr>
<td>target_file</td>
<td>Specifies the name of the U2 file to use when saving the schema.</td>
</tr>
<tr>
<td>TO</td>
<td>Defines where to store the Map Schema, and the format in which to save it.</td>
</tr>
<tr>
<td></td>
<td>If you specify XMAP, U2 saves the Map Schema in the EDAXMAP format.</td>
</tr>
<tr>
<td></td>
<td>If you do not specify this parameter, U2 saves the map schema in the</td>
</tr>
<tr>
<td></td>
<td>EDAMAP format.</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>schema_name</code></td>
<td>The record ID of the EDA Schema.</td>
</tr>
</tbody>
</table>

## SELECT.EDA.NONCONFORMING

The `SELECT.EDA.NONCONFORMING` command creates a select list of all nonconforming U2 record IDs. After creating the select list, you can execute ECL commands to access the records.

### Syntax

`SELECT.EDA.NONCONFORMING filename`

where `filename` is the name of the U2 file for which you want to view nonconforming records.

### Example

The following example illustrates creating a select list of all nonconforming records in the CUSTOMER file:

```ecl
:SELECT.EDA.NONCONFORMING CUSTOMER
2 records selected to list #0.
>SAVE.LIST CUSTOMER.NONCONFORMING
2 record(s) SAVEd to SELECT list "CUSTOMER.NONCONFORMING".
:GET.LIST CUSTOMER.NONCONFORMING
2 records retrieved to list 0.
$<50>LIST CUSTOMERS FNAME LNAME ADDR1 CITY STATE 11:07:56AM 21 JUN 2010
PAGE
1

CUSTOMER........10
First Name......Andrew
Last Name.......McCaig
Address line 1..999 Hill Road
City............Brattleboro
State...........VT

CUSTOMER........6
First Name......Betty
Last Name.......Burke
Address line 1. 400 Technology Path
City............White River Jun
State...........VT

2 records listed.
```

In this example, the CITY field exceeds the specified data type of VARCHAR(10).

## VERIFY.EDAMAP

The `VERIFY.EDAMAP` command verifies the EDA schema.
Syntax

VERIFY.EDAMAP\{[XMAP] \textit{eda_schema} | \texttt{EDA.FILE} [\texttt{DICT}] \textit{eda_file}\}
DEFAULT.MAP} [\texttt{DATA.SOURCE} \textit{data_source}] [\texttt{OBJECT.SET}
[name\_space.]\textit{primary_table}] [\texttt{FILE.NAME} \textit{target_file} [\texttt{METADATA}]

Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{eda_schema}</td>
<td>Specifies the name of the EDA schema to verify.</td>
</tr>
<tr>
<td>\textit{eda_file}</td>
<td>Specifies the name of the EDA file whose schema is to be extracted and verified. If you specify \texttt{FILE.NAME target_file}, \textit{target_name} replaces the U2 file name in the schema U2 verifies.</td>
</tr>
<tr>
<td>DEFAULT.MAP</td>
<td>Specifies to only verify the primary key (@ID) mapping, irrespective of the attributes actually mapped of the schema you specify.</td>
</tr>
<tr>
<td>\textit{data_source}</td>
<td>Specifies the data source name to use when verifying the schema.</td>
</tr>
<tr>
<td>\textit{primary_table}</td>
<td>Specifies the name of the primary table, containing only singlevalued attributes, to use when verifying the schema. If you also specify \texttt{name_space}, U2 uses it as the external schema name.</td>
</tr>
<tr>
<td>\textit{target_file}</td>
<td>Specifies the name of the U2 file to use when verifying the schema.</td>
</tr>
<tr>
<td>METADATA</td>
<td>Connects to the external database and verifies the metadata on that database.</td>
</tr>
</tbody>
</table>
Chapter 6: EDA exception handling

Exceptions can occur when an INSERT, DELETE, or UPDATE operation fails on the external database tables, or when U2 cannot convert a record to the external database tables.

Note: We recommend that you verify data and correct any exceptions prior to converting your data to an external database. For more information about data verification, see Verifying EDA schemas, on page 32.

You may choose to avoid certain external database errors being returned to your application by using the NONCONFORMING RECORD flag. For more information about this flag, see Defining options, on page 23.

Following are the EDA exceptions that can occur:

▪ Inserting or updating a record with the field length longer than the length defined in the EDA table.
▪ Inserting or updating a record with an incorrect data type value that the system cannot automatically convert to the data type defined in the EDA table.
▪ Inserting or updating a multivalued or multi-subvalued attribute to a singlevalued attribute.
▪ Inserting or updating a record containing an unmapped field when UNMAPFIELD has been disabled in the EDA Map Schema.
▪ The operation violates defined constraints.

When U2 detects an exception, the following events occur:

▪ If you are converting data, the conversion process loads data to the EDA file after it generates the metadata.
▪ U2 inserts or updates data to the EDA file at runtime.
▪ U2 deletes the EDA file record at runtime.

You can turn exception processing on and off with EDA.EXCEPTION command.

The EDA_EXCEPTION file

The EDA_EXCEPTION file is a multilevel file, with each subfile relating to one EDA data source. The name of the subfile is EDA_dataSource. The EDA_EXCEPTION file resides in /uvhome/sys on UniVerse for UNIX and \uvhome\sys on UniVerse for Windows platforms. Enter EDA.EXCEPTION ON to utilize the EDA_EXCEPTION file.

U2 records exceptions occurring during the conversion process or an INSERT, UPDATE, or DELETE operation in the EDA_EXCEPTION file.
Note: The command, EDA.EXCEPTION uses a period, while the file uses an underscore: EDA_EXCEPTION. The following example shows this difference.

```plaintext
>CT VOC EDA.EXCEPTION
   EDA.EXCEPTION
0001 Verb - Turn on/off EDA exception file
0002 EDA.EXCEPTION
0003 I
0004 G

>CT VOC EDA_EXCEPTION
   EDA_EXCEPTION
0001 File - EDA Exception File.
0002 C:\U2\uv113\EDA_EXCEPTION
0003 C:\U2\uv113\D_EDA_EXCEPTION
>
```

The following table describes each attribute of the EDA_EXCEPTION file.

<table>
<thead>
<tr>
<th>Location</th>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>@ID</td>
<td>The ID of the exception record. The ID concatenates the process ID, timestamp, and a sequential number.</td>
</tr>
<tr>
<td>1</td>
<td>ACCOUNT</td>
<td>The full path to the account where the data record resides.</td>
</tr>
<tr>
<td>2</td>
<td>FILE_NAME</td>
<td>The name of the EDA file where the exception happened.</td>
</tr>
<tr>
<td>3</td>
<td>FULL_PATH</td>
<td>The full path of the EDA file.</td>
</tr>
<tr>
<td>4</td>
<td>UID</td>
<td>The user ID of the user generating the exception.</td>
</tr>
<tr>
<td>5</td>
<td>DATE</td>
<td>The date the exception occurred.</td>
</tr>
<tr>
<td>6</td>
<td>TIME</td>
<td>The time the exception occurred.</td>
</tr>
<tr>
<td>7</td>
<td>ERROR_MSG</td>
<td>The error message returned from the external database.</td>
</tr>
<tr>
<td>8</td>
<td>OPERATION</td>
<td>The operation causing the exception. Valid values are EDA.CONVERT, UPDATE, INSERT, or DELETE.</td>
</tr>
<tr>
<td>9-13</td>
<td>Reserved for future enhancements</td>
<td></td>
</tr>
<tr>
<td>14 - n</td>
<td>REC_START</td>
<td>The data record causing the exception.</td>
</tr>
</tbody>
</table>

Example

In the following example, the CUSTOMER file contains an error from an EDA conversion (EDA.CONVERT). Line 0007 displays the error message, and line 0028 shows the non-conforming data record "8410DATE".

```plaintext
01 CT EDA_EXCEPTION,EDA_SQLS 4480-1470758635-1
4480-1470758635-1
0001 C:\U2\UV\HS.SALES
0002 H
0003 C:\U2\UV\HS.SALES\H
0004 DEN-VM-Tl4\Administrator
0005 17754
0006 36235
0007 EDA SQL Server Driver: [Microsoft][SQL Server Native Client 11.0][SQL Server] Conversion failed when converting date and/or time from character string.
0008 CONVERT
0009
0010
```
INMAT

The INMAT function returns the following values after UniVerse BASIC WRITE is executed from a UniVerse BASIC program or subroutine:

- **0** – The WRITE completed successfully
- **2** – The WRITE completed, but data type enforcement checking failed. If you specify the LOG DTE.OPT, the errors appear in the file_level log. If you specify the IGNORE option, use the SET.DTELOG command to set up your log file.
Chapter 7: EDA replication

EDA Replication is useful if you want to maintain an account from which you can create reports. This increases your options for keeping your applications available. Now you can replicate your data to an SQL database in addition to keeping your data safely stored in U2. When you store data in U2, it is simultaneously replicated to Oracle, IBM DB2, or Microsoft SQL Server. Use the replicated database for data-mining or reporting while you use U2 as your production workhorse.

To manage EDA Replication, use the EDA Replication Config Tool. This tool enables you to edit EDA map schemas, edit data source definitions, and convert U2 files to EDA files.

Setting up a server

To access the EDA Replication Config Tool, from the Start menu, select Programs, then select Rocket U2, then select EDA Replication Config Tool. The EDA Replication Tool appears, as shown in the following example:

To start using the EDA Replication Config Tool, you must first create a U2 server and then connect to it, as described in the following sections:

- Creating a new U2 server connection, on page 12
- Connecting to the U2 server, on page 14

Defining a data source

You must define a data source pointing to the external database to which you want to connect.
To define a new data source, connect to your U2 server, right-click **Data Sources**, then click **New EDA Data Source**. The Create a New EDA Data Source dialog box appears.

In the **Data Source Name** box, enter a unique name for the external data source, then click **Finish**. A data source information dialog box appears in the right pane of the EDA Schema Manager window.

In the **DSN/Net Service/DB Alias** box, enter the name of the external database to which you are connecting. The name of the external database must be the data source name defined in the ODBC Data Source Administrator.

From the **Driver** list, select the type of driver.

To define an EDA data source connection, click **Add**. The EDA Data Source Connection dialog box appears.

In the **Login User ID** box, enter the user ID on the external server.

In the **Password** box, enter the password corresponding to the User ID.

In the **Re-enter Password** box, type the password again to verify it.

To maintain the connection on the external server after a transaction commits, select **YES** from the **Hold Flag** list. Otherwise, if you want to disconnect from the external server after the transaction commits, select **NO**.

**Note:** If you do not use UniVerse BASIC transactions, each U2 database operation, such as a READ or WRITE, corresponds to an external transaction.

In the **Qualified Users** box, enter the U2 user IDs of users who can access the external server from the U2 account using the external Login User ID you specify. Separate the users by a “|” symbol. If all U2 users can access the external account, enter an asterisk (*).

To test the connection to the external instance, click **Test**. If the connection is successful, a message appears stating such.

**Defining EDA replication parameters**

In the EDA Replication Config Tool, right-click the account to which you want to replicate your EDA data, then select **EDA replication config tool**. The EDA Replication Config Tool editor is displayed, as shown in the following example:
Configuring replication parameters

Click **Configure Replication Parameters**. The **Configure Replication Parameters** panel appears, as shown in the following example:
To change the value of a configuration parameter, click the **New Value** column of the parameter you want to change, then enter the new value for the parameter.

**MAX_LRF_FILESIZE**

The maximum Log Reserve File Size, in bytes. The default value is 1,073,741,824 (1 GB). The maximum value is 2,147,483,136.

**MAX_REP_SHMSZ**

The maximum shared memory buffer segment size. The default value is 67,108,864 (64 MB).

**N_REP_OPEN_FILE**

The maximum number of open replication log files for a udt or tm process. The default value is 8.
REP_FLAG (replication flag)

The REP_FLAG parameter turns U2 Data Replication on or off. If you choose the install U2 with the U2 Data Replication feature, U2 sets the REP_FLAG to 1. The following table describes the REP_FLAG options:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (zero)</td>
<td>The U2 Data Replication System is off.</td>
</tr>
<tr>
<td>Any positive integer</td>
<td>The U2 Data Replication System is on.</td>
</tr>
</tbody>
</table>

TCA_SIZE

The maximum number of entries in the transaction control area (TCA). TCA is only used when there is more than one replication group configured, and there are transactions across replication groups. The default value is 2048.

If you are not using transaction processing, this parameter is irrelevant. If you are using transaction processing, set the value of TCA_SIZE to at least the number of users on the system.

Configuring the replication system

To define the system to which you want to replicate EDA data, perform the following steps.

1. Click **Configure Replication System**.
   The Configure Replication System dialog box appears, as shown in the following example:
2. From the Replication Systems list, select the system to which you want to replicate data, and click OK.
This system should be the same system on which the EDA account resides.
The Configure Replication System dialog box appears, as shown in the following example:

3. In the System ID field, enter a name for the replication system.
This name should be unique on a system. System ID can contain a combination of alphabetic characters, number, and any of the following character: ~ ! @ $ % ^ & * - + . / \.

4. In the Host Name field, enter the host name of the replication system location. A system can have only one host name.

5. In the Version field, select the version of U2 on the system location. The version number must be 112 or higher.

6. Select the DHCP check box if you want to specify that the local system has a dynamic IP address.
If you define a local system as a DHCP system, U2 Data Replication automatically sends the current IP address in the SYNC request to the server.

7. Next to Auto Resume, select Yes or No.
Auto Resume indicates whether replication from the system you specify is synchronized and resumes automatically when U2 starts, or after a reconfiguration. Select Yes if you want to automatically synchronize or No if you do not want to automatically synchronize.

8. Define the sync interval.
Sync Interval defines the time interval, in minutes, in which the replication system automatically synchronizes replication.
U2 Data Replication automatically synchronizes subscribing systems with their publisher every period defined by sync_interval. A sync_interval of 0 indicates a manual synchronization system, where the system does not automatically synchronize the systems.
Sync Interval applies only to those subscribing groups that have deferred replication. It does not apply to publishing groups.
9. To verify the subscribing system, select the Connect Authorization check box. U2 Data Replication performs an authorization check when a SYNC request is received from the subscribing system.

For remote systems with a static IP address, the publishing system can always trust the subscribing system because the IP address is defined in the repsys file. However, if the remote system is a DHCP system, the publishing system cannot verify the IP address.

10. Define the timeout.

**Timeout** defines the number of seconds to wait if no packets are received from the system before suspending replication.

The publishing system sends a packet to the subscribing system approximately every 4 seconds when replication is idle. The subscribing system then sends a packet back to the publishing system. If the subscribing system location has timeout defined, the publistener process counts the time that has elapsed between packets being received. If the amount exceeds the value defined by timeout, replication is suspended.

In the value of **Timeout** is 0, no timeout occurs.

We recommend that you not set the value of the TIMEOUT phrase to less than 2 minutes.

11. Next to Exception Action, click Browse to define the full path to the exception action.

The replication exception action is a shell script on UNIX platforms, or a batch program on Windows platforms. For example, if you define a replication exception action as UDRepExceptionAction.sh in the /usr/uv112 directory, browse to that directory.

12. The account definition is automatically populated with the account you previously defined. To define a different account to which to replica EDA data, click Add in the Account Definitions area.

The Replication System Definition dialog box appears, as shown in the following example:

![Replication System Definition dialog box](image)

a. In the Account Name field, enter the name of the account.
b. In the Account Path field, select the path to the account.
c. Click Finish to save the definitions.
13. To save your settings, click **Save Changes**.

**Choosing files to replicate**

To choose the files to replicate, perform the following steps.

1. Click **Choose Files to Replicate**.
   
   If you have not previously defined a group, the following message appears:

2. Click **Create** to define a replication group.
   
   The Choose Files to Replicate dialog box appears, as shown in the following example:
a. In the **Group ID** field, enter a unique name for the subscribing group.

b. In the **Level** field, select the level of replication. You can only select FILE for EDA Replication.

c. In the **Files** area, click **Add** to select the files you want to publish.

   A dialog box similar to the following example appears:

   ![Replication Distribution Details](image)

   By default, both the data portion and the dictionary portion of the file are selected. If you do not want to publish the data portion of the file, clear the **Data** check box. If you do not want to publish the dictionary portion of the file, clear the **Dict** check box. To enable the ability to update the file on the subscribing system, select the **Sub Writable** check box.

   Select the files you want to publish, then click **Finish**.

d. In the **Distributions** area, click **Add** to define replication distribution details.

   The Replication Distribution Details dialog box appears.

   Select the local system. Select the Replication mode you want to use. For information about types of U2 Data Replication, see the *U2 Data Replication User Guide.*
Click **Finish**.

e. Set the configuration parameters.
Set any of the configuration parameters necessary for your environment in the **Configuration** area of the Publishing Group Details dialog box, as shown in the following example:

For information about these parameters, see the *U2 Data Replication User Guide*.

3. To save your settings, click **Save Changes**.

**Synchronizing replication files**

To synchronize replication files, perform the following steps.

1. Click **Synchronize replication files**.

   The Synchronize replication files dialog box appears, as shown in the following example:
2. Select the files from the source account that you want to synchronize with the target account, then click **Start File Synchronization**.

If the file you are trying to synchronize already exists on the target file, U2 displays the following message box:

![Synchronize replication files dialog box](image)

If you want to overwrite the existing file, select **Overwrite existing files in the target account**.

3. Click **OK**.

This process may take several minutes or longer, depending on the size of the files.

**Note:** Make sure no users are updating the database when you synchronize the files.

While U2 is synchronizing the files, it pauses the database.

When the synchronization is complete, U2 displays a message indicating that the synchronization was successful.

**Creating EDA schemas for replicated files**

You can create a default EDA schema for the files you selected, which maps each D-type attributes, or select the attributes you want to map.

**Creating default EDA schemas**

To create a default EDA schema, click **Create Default EDA Schemas**. The following dialog box appears:
Select the data source for which you want to create schemas, or click **New Data Source** to create a new data source. Click **OK**.

U2 displays informational messages when creating the schemas, as shown in the following example:

```
Generating schema for 'COURSES' successful.
Generating schema for 'CUSTOMER' successful.
Generating schema for 'INVENTORY'
```

**Creating EDA schemas**

To select the dictionary attributes you want to map, click **Create EDA Schemas**.

Select the data source for which you want to create schemas, or click **New Data Source** to create a new data source. Click **OK**.

Select the dictionary attributes for which you want to create a schema. If you are creating schemas for multiple files, click the arrow next to the current file name to proceed to the next file.

The following example illustrates the U2 Dictionary Attributes dialog box:
When you have finished selecting the dictionary attributes for which you want to create schemas, click **Finish**.

U2 creates the schema files and generates informational messages.

**Converting replicated files to EDA files**

To convert replicated files to EDA files, click **Convert the Replicated Files to EDA Files**. A dialog box similar to the following example appears:
Select the files you want to convert to EDA. Make sure you have synchronized the files before converting them to EDA. Click **EDA Convert**. The following dialog box appears:

![EDA Replication Config Tool](image)

Select the type of conversion you want to use. Valid options are:

- **Reconvert** – If the file has already been converted to EDA and you want to convert the file again, select **Re-convert**
- **Force** – Drops existing tables before creating new ones
- **Verbose** – Show detailed messages during the conversion process

Click **Finish**. U2 suspends replication during the conversion process and converts the replicated files you selected to EDA files and displays informational messages.
Chapter 8: EDA best practices

UniVerse provides many options when mapping data from the UniVerse database to the EDA database. You can choose multiple multivalued fields and I-descriptors, but be aware that certain mapping choices can result in nonoptimal performance data and conformance issues. This chapter describes guidelines to achieve better EDA performance and fewer errors.

Map selected fields

Only map the fields that need to be viewed on the EDA server, or the fields that are frequently listed on the U2 server. The more fields you select to map, the greater the chance for invalid data type errors, and the worse the performance.

Single-valued fields have much less performance overhead than multivalued or multi-subvalued fields.

Avoid multiple multivalued associations

Explicitly mapping multivalued or multi-subvalued attributes from multiple associations to the EDA server causes expensive outer-joins when returning records to U2, causing in turn performance degradation when using a RetriVe or READ statement. The large result set may also encounter external database limitations and result in external errors.

If you need to map multiple associations, use the WHOLE RECORD option to increase performance and avoid external database errors. See Defining options, on page 23 for more information.

Avoid restrictive data types

Mapping U2 data using an incorrect data type can cause insert operations to fail, and may cause unexpected results when executing a query against the mapped data.

For example, consider the following example if you map a field containing ABC to the EDA server as a CHAR(10) datatype:

```
SELECT * FROM TEST WHERE field1 = “ABC”;
```

U2 will not return any results because ABC actually appears as “ABC    “ on the EDA server. If you choose VARCHAR(10) as the data type, U2 will return results.

Data types for record IDs

When mapping the record ID, we strongly suggest using a VARCHAR() data type with the length at least as long as the longest record ID in the U2 file. If the insert operation of the record ID fails, U2 does not write any of the record to the EDA server.

Use care when mapping record IDs that have a data type of INTEGER, DATE, or TIME. Because U2 converts these types of data, using OCONV to map the data to the external database and ICONV to use the data in U2, different results may occur. For example, if you have two records with the record IDs of “1” and “1.0” in the U2 database, and you use the TIME data type when mapping to the external database, only one record will be mapped, since each of these record IDs converts to 00:00:10.
When you specify NONCONFORMING or WHOLE_RECORD, U2 stores unmapped fields or the whole record in a record blob on the external database. Set the size of the record blob to the largest record size in the U2 file to avoid insert failures.

The following table describes the default sizes for RECORD_BLOB.

<table>
<thead>
<tr>
<th>Server type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td>The CLOB data type is used and set to 2GB by default.</td>
</tr>
<tr>
<td>SQL Server 2008</td>
<td>The varchar(max) data type is used and set to $2^{31} - 1$ bytes by default.</td>
</tr>
<tr>
<td>SQL Server 2012</td>
<td>The varchar(max) data type is used and set to 2GB by default.</td>
</tr>
<tr>
<td>DB2</td>
<td>Maximum length of CLOB is 2,147,483,647 bytes (2GB - 1 byte)</td>
</tr>
</tbody>
</table>

### Updating an EDA file from the external database

You can update external tables that comprise an EDA file using external database applications and tools, and these updates will be seen by U2 applications. You must follow a few rules to ensure the integrity of the data.

The following types of external tables must not be updated outside of U2 EDA:

- When the WHOLE_RECORD flag is set to true.
- When an I-descriptor index is mapped to a column of the external table, for example, when the mapping type is set to DATA for a Idescriptor in the EDA conversion process.

Additionally, you must pay special attention to the NONCONFORMING flag, and follow these rules:

- When inserting a new row, always set NONCONFORMING_FLAG to 0 for that row.
- When updating an existing row, if the NONCONFORMING_FLAG is 0, you may proceed with the update, but do not update a row which has the NONCONFORMING_FLAG set to 1. If you need to update a row with the NONCONFORMING_FLAG set to 1, perform a clean up of the values in this record before attempting an update. See Defining options, on page 23 for more information.