Edition

Publication date: April 2014
Book number: UNV-1123-EDA-1
Product version: Rocket UniVerse V11.2.3

Copyright

© Rocket Software, Inc. or its affiliates 1985-2014. All Rights Reserved.

Trademarks

Rocket is a registered trademark of Rocket Software, Inc. For a list of Rocket registered trademarks go to: www.rocketsoftware.com/about/legal. All other products or services mentioned in this document may be covered by the trademarks, service marks, or product names of their respective owners.

Examples

This information might contain examples of data and reports. The examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

License agreement

This software and the associated documentation are proprietary and confidential to Rocket Software, Inc., are furnished under license, and may be used and copied only in accordance with the terms of such license.

Note: This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

Contact information

Website: www.rocketsoftware.com
Rocket Software, Inc. Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA
Tel: +1 781 577 4321
Fax: +1 617 630 7100
Contacting Global Technical Support

If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Customer Portal to report and track a problem, to submit an enhancement request or question, or to find answers in the U2 Knowledgebase. The Rocket Customer Portal is the primary method of obtaining support.

To log in to the Rocket Customer Portal, go to:

www.rocketsoftware.com/support

If you do not already have a Rocket Customer Portal account, you can request one by clicking Need an account? on the Rocket Customer Portal login page.

Alternatively, you can contact Global Technical Support by email or by telephone:

Email: u2support@rocketsoftware.com

Telephone:

<table>
<thead>
<tr>
<th>Region</th>
<th>Phone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>North America</td>
<td>+1 800 729 3553</td>
</tr>
<tr>
<td>United Kingdom/France</td>
<td>+44 (0) 800 773 771 or +44 (0) 20 8867 3691</td>
</tr>
<tr>
<td>Europe/Africa</td>
<td>+44 (0) 20 8867 3692</td>
</tr>
<tr>
<td>Australia</td>
<td>+1 800 707 703 or +61 (0) 29412 5450</td>
</tr>
<tr>
<td>New Zealand</td>
<td>+0800 505 515</td>
</tr>
</tbody>
</table>
# Table of Contents

## Chapter 1: External Database Access (EDA)
- External Database Access ................................................. 1-2
- First normal form (1NF) vs. non-first normal form ............... 1-3
- Table concepts. .............................................................. 1-4
  - Representing multivalues ............................................ 1-4
  - Associations .............................................................. 1-5
  - I-descriptors ............................................................. 1-5
- Mapping example .......................................................... 1-7

## Chapter 2: The EDA Schema Manager
- The EDA Schema Manager steps ...................................... 2-2
- The EDA Schema Manager ............................................... 2-3
  - Create a new UniVerse server connection ....................... 2-4
  - Connect to UniVerse server ......................................... 2-8
- Managing connections .................................................... 2-11
  - About data sources ..................................................... 2-11
  - Defining a data source ................................................ 2-14
- Selecting EDA schema files ............................................ 2-19
  - Selectively mapping attributes .................................... 2-21
  - View EDA server details ............................................. 2-27
  - View UniVerse server details ...................................... 2-28
  - View options ............................................................ 2-29
- Verifying the EDA schema .............................................. 2-44
  - Viewing the EDA schema ............................................. 2-47
- Viewing the DDL scripts ................................................ 2-48
- Converting data .......................................................... 2-49
  - Viewing EDA files ...................................................... 2-51
- Listing and selecting data ............................................. 2-52
  - Listing data using RetriVe .......................................... 2-52
  - Listing data using UniVerse SQL .................................. 2-53
Chapter 3: External database drivers supplied with EDA

EDA Oracle driver ........................................... 3-2
  Set up the EDA environment. ........................... 3-2
  Set up the Oracle connection file ...................... 3-2
  Set up dynamic-loading library ....................... 3-3
  Create ORACLEPATH Environment Variable ............ 3-3
  Create the EDA data source ......................... 3-3
  Oracle data type mapping ........................... 3-4
EDA DB2 driver ............................................. 3-5
  Set up the EDA environment. ........................... 3-5
  Install DB2 or the DB2 client ....................... 3-5
  Set up connection to the DB2 database .............. 3-5
  Create the EDA data source ......................... 3-6
  DB2 data type mapping ........................... 3-6
EDA SQL Server driver .................................... 3-7
  Install SQL Server and create ODBC data source ... 3-7
  Create the EDA data source ......................... 3-7
  Set up the EDA configuration file ................... 3-7
  SQL Server data types ............................. 3-8
EDA SQL Server driver for UNIX and Linux .......... 3-9
  Set up the EDA environment. ........................... 3-9
  Install unixODBC and third-party ODBC driver .... 3-9
  Set up connection to external database ............ 3-10
  Set up the ODBC dynamic loading library path ...... 3-10
  Create the EDA data source ......................... 3-11
  Automatic data type mapping ....................... 3-11

Chapter 4: External Database Access Driver API

External Database Access Driver API .................. 4-2
Registering an EDA driver ............................ 4-3
EDA Driver functions ................................... 4-4
  EDADRV_LoadSymbols .................................. 4-5
  EDADRV_Connect ...................................... 4-8
  EDADRV_Disconnect .................................... 4-9
  EDADRV_EndTransaction .............................. 4-10
  EDADRV_PrepareStmt .................................. 4-11
  EDADRV_ExecuteStmt .................................. 4-13
  EDADRV_CloseStmt .................................... 4-15
  EDADRV_DropStmt .................................... 4-16
Chapter 5  Chapter 5: EDA ECL commands

   Connecting to the external database. 5-2
  Disconnecting from the external database. 5-3
   Converting UniVerse data to an external database 5-3
   VERIFY.EDAMAP 5-4
   LIST.EDAMAP 5-5
   SAVE.EDAMAP 5-7
   SELECT.EDA.NONCONFORMING 5-8
   Retrieving information about the EDA driver 5-9

Chapter 6  Chapter 6: EDA exception handling

Chapter 7  Chapter 7: EDA Replication

   EDA Replication 7-2
   Setting up a server 7-3
   Create a new UniVerse server connection 7-4
   Defining a data source 7-8
   Defining EDA replication parameters 7-13

Chapter 8  Chapter 8: EDA best practices

   Map selected fields 8-1
   Avoid multiple multivalued associations 8-1
   Avoid restrictive data types 8-1
   RECORD_BLOB 8-2
   Updating an EDA tile from the external database 8-3
# Chapter 1: External Database Access (EDA)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Database Access</td>
<td>1-2</td>
</tr>
<tr>
<td>First normal form (1NF) vs. non-first normal form</td>
<td>1-3</td>
</tr>
<tr>
<td>Table concepts</td>
<td>1-4</td>
</tr>
<tr>
<td>Representing multivalues</td>
<td>1-4</td>
</tr>
<tr>
<td>Associations</td>
<td>1-5</td>
</tr>
<tr>
<td>I-descriptors</td>
<td>1-5</td>
</tr>
<tr>
<td>Mapping example</td>
<td>1-7</td>
</tr>
</tbody>
</table>
External Database Access

External Database Access (EDA) enables you to convert data stored in the UniVerse database to a 1NF database, such as Microsoft SQL Server, then access that data using existing UniVerse BASIC programs, RetrieVe, or 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 SQL Client Interface (BCI).

You must create an EDA Map Schema to define the translation between UniVerse and the external database model, which may differ from the UniVerse model. Additionally, the UniVerse dictionary record does not fully describe the data it defines. For example, the UniVerse dictionary record may 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 UniVerse. In addition, you must be able to access the external database server where you want the data to reside using that client.
First normal form (1NF) vs. non-first normal form

Many relational databases, including DB2, Oracle, and SQL Server, 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.

UniVerse follows the nested relational, or Non-First Normal Form model, referred to as NF2. This model enables you to store data in singlevalued or multivalued attributes, 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 DB2, Oracle, or 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.

*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 UniVerse files or UniVerse 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 UniVerse attribute. This not only ensures the uniqueness of each row in the external table, but preserves the order of values in the multivalued attribute. The record ID column of the multivalued attributes table is the foreign key pointing to the primary key of the primary, singlevalued attributes table.

**Associations**

The “association” is the mechanism UniVerse uses to establish a relationship among attributes. Within an association, multivalued attributes are related to, or associated with, each other.

Following is an example of related information that would be stored in a UniVerse 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 UniVerse file contains more than one association, the EDA Schema Manager creates a separate multivalued (MV) table for each association. One singlevalued attribute (S) table can be the parent of many multivalued attribute (MV) tables.

**I-descriptors**

You can map I-descriptors to an external database. There are three types of mapping:
Simple – a simple I-descriptor, such as A + B. These are I-descriptor formulas that are translated to expressions and SCALAR functions. For more information, see View field details in Chapter 2, “Chapter 2: The EDA Schema Manager.”

TRANS – an I-descriptor that performs a TRANS operation. These are I-descriptors that are mapped using TRANS or TABLE function type of mapping. TABLE function mapping is used for multiple TRANS operations. For more information, see View field details in Chapter 2, “Chapter 2: The EDA Schema Manager.”

Materialized Virtual – an I-descriptor that is evaluated in UniVerse, with the result stored in the external database. If you are mapping this type of I-descriptor, select DATA as the type of mapping. For more information, see View field details in Chapter 2, “Chapter 2: The EDA Schema Manager.”

Note: You cannot update UniVerse I-descriptors. I-descriptors are evaluated by the database engine according to the formula you specify in the dictionary record. Likewise, you cannot update I-descriptors you map to an external database. This applies to all types of I-descriptors, including materialized I-descriptors. Do not attempt to update their values using external database tools, or you risk compromising the consistency of your data and UniVerse applications.
## Mapping example

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

```plaintext
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Conversion</th>
<th>Column</th>
<th>Output Depth</th>
<th>Code</th>
<th>Heading</th>
<th>Format</th>
<th>Assoc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTID</td>
<td>D 0</td>
<td>P(0N)</td>
<td>Customer ID</td>
<td>10R</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@ID</td>
<td>D 0</td>
<td></td>
<td>CUSTOMER</td>
<td>10L</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAL</td>
<td>D 1</td>
<td></td>
<td>Salutation</td>
<td>5T</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNAME</td>
<td>D 2</td>
<td></td>
<td>First Name</td>
<td>12T</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LNAME</td>
<td>D 3</td>
<td></td>
<td>Last Name</td>
<td>16T</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPANY</td>
<td>D 4</td>
<td></td>
<td>Company Name</td>
<td>20T</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR1</td>
<td>D 5</td>
<td></td>
<td>Address line 1</td>
<td>30T</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR2</td>
<td>D 6</td>
<td></td>
<td>Address line 2</td>
<td>30T</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CITY</td>
<td>D 7</td>
<td></td>
<td>City</td>
<td>12T</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE</td>
<td>D 8</td>
<td>P(2A)</td>
<td>State</td>
<td>2L</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZIP</td>
<td>D 9</td>
<td>P(5N)</td>
<td>Zip</td>
<td>5L</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHONE</td>
<td>D 10</td>
<td>P(&quot;(&quot;3N&quot;)3N&quot;)</td>
<td>Telephone</td>
<td>13R</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROID</td>
<td>D 11</td>
<td>P(1A4N)</td>
<td>Product</td>
<td>5L</td>
<td>M ORDER S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SER_NUM</td>
<td>D 12</td>
<td>P(6R)</td>
<td>Serial#</td>
<td>6L</td>
<td>M ORDER S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRICE</td>
<td>D 13</td>
<td>MD0,§</td>
<td>Price</td>
<td>7R</td>
<td>M ORDER S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUY_DATE</td>
<td>D 14</td>
<td>D2/</td>
<td>Date Purchased</td>
<td>8R</td>
<td>M ORDER S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAID_DATE</td>
<td>D 15</td>
<td>D2/</td>
<td>Date paid</td>
<td>8R</td>
<td>M ORDER S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVC_PRICE</td>
<td>D 16</td>
<td>MD0,§</td>
<td>Service</td>
<td>7R</td>
<td>M ORDER S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVC_START</td>
<td>D 17</td>
<td>D2/</td>
<td>Service</td>
<td>8R</td>
<td>M ORDER S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVC_END</td>
<td>D 18</td>
<td>D2/</td>
<td>Service</td>
<td>8R</td>
<td>M ORDER S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVC_PAID_DATE</td>
<td>D 19</td>
<td>D2/</td>
<td>end date</td>
<td>8R</td>
<td>M ORDER S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FULLNAME</td>
<td>I</td>
<td>SAL:'</td>
<td>Contact Name</td>
<td>30T</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISCOUNT</td>
<td>I</td>
<td>SUBR(&quot;HS.DIS MD1\n CNT&quot;, LIST_PRI C&quot;, PRICE)</td>
<td>Discount</td>
<td>6R</td>
<td>M ORDER S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATENAME</td>
<td>I</td>
<td>TRANS(STATES, STATE, NAME,&quot;C&quot;)</td>
<td>State name</td>
<td>14T</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>I</td>
<td>TRANS(PRODUCT &amp; PROID, DESCRIPTON,&quot;C&quot;)</td>
<td>Product Description</td>
<td>20T</td>
<td>M ORDER S</td>
<td></td>
<td></td>
<td></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.

singlevalued attribute table

CUSTOMER
(ID, SAL, FNAME, LNAME, COMPANY, ADDR1, ADDR2, CITY, STATE, ZIP, PHONE)

multivalued attribute table

CUSTOMER_ORDERS_MV
(PRODID, SER_NUM, PRICE, BUY_DATE, PAID_DATE, SVC_PRICE, SVC_START, SVC_END, SVC_PAID_DATE, DISCOUNT, DESCRIPTION)

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: The EDA Schema Manager

The EDA Schema Manager steps ........................................ 2-2
The EDA Schema Manager ............................................. 2-3
  Create a new UniVerse server connection .................... 2-4
  Connect to UniVerse server ....................................... 2-8
Managing connections ............................................... 2-11
  About data sources .............................................. 2-11
  Defining a data source ......................................... 2-14
Selecting EDA schema files ....................................... 2-19
  Selectively mapping attributes ............................... 2-21
  View EDA server details ....................................... 2-27
  View UniVerse server details ............................... 2-28
  View options .................................................... 2-29
Verifying the EDA schema ......................................... 2-44
  Viewing the EDA schema ....................................... 2-47
  Viewing the DDL scripts ....................................... 2-48
Converting data ..................................................... 2-49
  Viewing EDA files .............................................. 2-51
Listing and selecting data ...................................... 2-52
  Listing data using Retrieve .................................. 2-52
  Listing data using UniVerse SQL .......................... 2-53
The EDA Schema Manager steps

The conversion of UniVerse files to DB2 tables and views is shown in the following diagram:

The EDA Schema Manager performs the following steps when converting UniVerse files to the external database:

1. The EDA Schema Processor receives information from the UniVerse 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 UniVerse physical file to create tables and views in the external database.
5. The UniVerse physical file is replaced by an EDA file in the UniVerse account. The original data file is saved under filename.edasave. The EDA file contains runtime mapping information. It does not contain data since all data has been transferred to the corresponding external tables.
The EDA Schema Manager

Use the EDA Schema Manager to create a mapping file, called EDA Schema, for a UniVerse file you are converting to an external database. You then convert UniVerse data to the external database using this mapping file.

To access the EDA Schema Manager, from the Start menu, select Programs, then select Rocket U2, then select EDA Schema Manager. The EDA Schema Manager appears, as shown in the following example:
Create a new UniVerse server connection

To create a new UniVerse server connection, right-click U2 Servers, then click New U2 Server. A dialog box similar to the following example appears:

Enter server name

In the Name box, enter a unique identifier for the new server.

Enter host name

In the Host box, enter the network name of the host computer where the UniVerse database resides, or the IP address.
Select the database

Select the type of database where the data resides. Valid choices are UniData or UniVerse.
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:
**Protocol type**

In the Protocol Type box, choose the type of communication you are using the server. You can choose Default, TCP/IP, or Lan Manager. The default is TCP/IP.

**RPC port number**

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

**RPC service name**

In the RPC Service Name box, enter the name of the RPC service on your system. For UniVerse, this is normally uVcs.

**Login account**

In the Login Account box, enter the name of the account to which you want to log on when accessing UniVerse.

**Commands to execute**

Click Add in the Commands to Execute box to enter commands you want to execute when you log on to the server. The following dialog box appears:
Click **Finish** to establish the UniVerse server. The new server appears in the **U2 Servers** area of the **EDA Schema Manager**, as shown in the following example:

![EDA Schema Manager](image)

**Connect to UniVerse server**

To connect to the UniVerse server, right-click the server name, then click **Connect**.
When you connect to the server, the **Connect to U2 Server** dialog box appears, as shown in the following example:

![Connect to U2 Server dialog box](image)

In the **User ID** box, enter the User ID for the machine where UniVerse is running, then enter the corresponding password in the **Password** box, then click Finish.
The Accounts and existing Data Source definitions appear in the **U2 Server** area, as shown in the following example:
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 UniVerse is installed. UniVerse supports the following three types of external databases:

- IBM DB2
- Microsoft SQL Server
- Oracle Database

Note: For more information regarding the external database that you are accessing, we strongly suggest reading Chapter 3, "Chapter 3: External database drivers supplied with EDA," before proceeding.

An external database server may reside on the same or different machine from the UniVerse server machine, but the corresponding client must reside on the same machine where the UniVerse server is installed. The following diagrams describe typical client-server configurations.

Connecting to SQL Server, Oracle, or the IBM database

The UniVerse database server may 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 UniVerse database server machine. The drivers for access to the databases are:

- SQL Server - Open Source or third-party ODBC library for UNIX
- Oracle - Oracle Client Library (OCI)
- DB2 - DB2 Client Library (CLI)
The UniVerse installation automatically places the EDA Driver Library for S2L Server (libcomdrv), the EDA Driver Library for Oracle (liboradrv), and the EDA Driver Library for DB2 (libdb2drv) and the in the $UVBIN directory.

The following example shows how UniVerse connects to an Oracle server:

The next example shows how UniVerse connects to a DB2 server:
The next example shows how UniVerse connects to SQL Server from a UNIX or Linux platform:

**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 UniVerse database server machine. The UniVerse installation automatically places the EDA Driver Library for SQL Server (libsqldrv) in the $UVBIN directory.

The following example shows how UniVerse connects to SQL Server from a Windows platform:
**Defining a data source**

To define a new data source, connect to your UniVerse server, right-click **Data Sources**, then click **New EDA Data Source**. The **Create a New EDA Data Source** dialog box appears, as shown in the following example:

In the **Enter 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, as shown in the following example:

In the **External DB Name** box, 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.

In the **Driver** box, enter the type of driver.
Click **Add**. The **EDA Data Source Connection** dialog box appears, as shown in the following example:

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. Enter the password again in the **Re-enter Password** box.

If you want to maintain the connection to the external server after a transaction commits, select **YES** in the **Hold Flag** box. If you want to disconnect from the server after the transaction commits, select **NO** in this box.

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

In the **Qualified Users** box, enter the UniVerse user IDs of users who can access the external server from the UniVerse account using the external Login User ID you specify. Separate the users by a “|” symbol. If all UniVerse users can access the external account, enter an asterisk (“*”).
The following example shows a completed **EDA Data Source** dialog box:
To test the connection to the external database instance, click **Test**. If the connection is successful, a message similar to the following example appears:

![Test EDA Data Source Connection](image)

Successfully made connection to data source "silver.denver.ibm.com".

From the **File** menu, click **Save** to save your data source definition, or click the Save icon.
Selecting EDA schema files

From the EDA Schema Manager, expand Accounts, expand the UniVerse account where the files you want to convert reside, right-click the EDA Schema Files, then click New EDA Map Schema. The Create New EDA Map Schema dialog box appears, as shown in the following example:
In the **EDA Schema Name** box, enter a unique name for the EDA schema. Select **EDA Schema** for the Map Format, then click **Next**. The **Source U2 file** dialog box appears, as shown in the following example:

In the file list, highlight the UniVerse file for which you are creating a schema. Click **Next**.
The **U2 Dictionary Attributes** dialog box appears, as shown in the following example:

![U2 Dictionary Attributes Dialog Box](image)

The EDA Map Schema displays each D-type dictionary attribute for the file you specified. Select each dictionary attribute you want 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 I-descriptor mapping, you must click **Deselect All**.

### Selectively mapping attributes

If you want to selectively map UniVerse attributes to an external database, only select those attributes you want to map from the **U2 Dictionary Attributes** dialog box when creating the EDA Map Schema. If you do not select any dictionary attributes, UniVerse automatically maps the @ID attribute.
In the following example, only the LNAME and @ID attributes have been selected:

![Diagram of attribute selection](image)

<table>
<thead>
<tr>
<th>@ID</th>
<th>T</th>
<th>LOC</th>
<th>C.</th>
<th>NAME</th>
<th>F...</th>
<th>S.</th>
<th>A...</th>
<th>Correlatives</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CUSTOMER</td>
<td>10L</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Customer</td>
<td>10R</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Salutation</td>
<td>5T</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>First Name</td>
<td>12T</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Last Name</td>
<td>16T</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Company...</td>
<td>20T</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Address li.</td>
<td>30T</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>City</td>
<td>12T</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>State</td>
<td>2L</td>
<td>S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Select All  Deselect All

< Back  Next >  Finish  Cancel
The following example illustrates the appearance of the window after the LNAME attribute has 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.

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.

**Attribute details**

In the Attribute Details portion of the window, define the mapping details for the attribute you selected.

**Note:** 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 Name attribute is the name of the column in the resulting external table.
- In the Type box, select the type of attribute. In this case, the type of attribute is TRANS.
- In the Data Type box, enter the data type for the attribute. In this case, the data type is VARCHAR.
The **Reference** box is used for I-type attributes that contain a TRANS clause. Enter the name of the external table TRANS clause reference in this box.

For example, if you are mapping the DESCRIPTION attribute from the CUSTOMER file, drag the attribute under ORDERS_MV, as shown in the following example:

As you can see in the **Mapped U2 Fields** portion of the window, this attribute contains a TRANS function to the PRODUCTS file. In the **Reference** box, enter the external table and column where the EDA CUSTOMER file was mapped, in this case, PRODUCT.PRODUCT/DESCRIPTION.
In the **Parameters** box, specify the attribute or expression in the TRANS function that returns the record ID in the table you are referencing in the external table. For example, the record ID for PRODUCTS is PRODID. Click the plus sign (“+”) in the **Parameters** portion of the window. Enter PRODID in the **Parameter** box, as shown in the following example:
- If the attribute is an expression, enter the SQL expression in the **Expr Body** box. For example, the `FULLNAME` attribute in the `CUSTOMER` file concatenates the salutation, space, first name, space, and last name attributes. In the expression box, enter the corresponding SQL statement for this attribute, as shown in the following example:

- In the **Formatting** box, select the appropriate format for the attribute.
- The **Index** box is not supported at this release of UniVerse.

**Note:** If you select all dictionary attributes, UniVerse only maps D-type attributes. You must map I-descriptors manually.

In addition, you cannot map unassociated multivalued or I-descriptors. You also cannot map associations that contain only I-descriptors.
View EDA server details

To view information about the EDA Server, click the EDA Schema Name/Table Name in the EDA Map Schema area of the U2 EDA Schema Manager. Information about the EDA server appears, as shown in the following example:

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

- The DBInstance box displays the name of the instance on the EDA server.
- The DBMSName box displays the name and version of the database on the EDA server.
- The DBMSFamily box displays the database family to which the DBMS name belongs.
- The DBModel box displays the type of database. At this release, UniVerse only supports 1NF databases.
- The NameSpace (Schema) box displays the name of the schema on the EDA server. You can change the name of the schema.
- The Root Name box displays the name of the table on the EDA server. You can change the name of the table.
- The Data Source box displays the name of the data source on the EDA server.
View UniVerse server details

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

![UniVerse Server Details](image)

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

- The **Host** box displays the name of the UniVerse host server where the file resides.
- The **U2 System** box displays the type of database on the server where the file resides.
- The **U2 Version** box displays the version of the database on the UniVerse server where the file resides.
- The **U2 Account** box displays the full path to the account on the UniVerse server where the file resides.
- The **File Name** box displays the name of the file on the UniVerse server.
View options

To view information about the mapping options, click Options in the EDA Map Schema area of the U2 EDA Schema Manager. Information about the mapping options server appear, as shown in the following example:

The following mapping options are available:

Whole record

The Whole Record option specifies whether or not to store the entire UniVerse record in the RECORD_BLOB column on the EDA server at the same time the individually mapped UniVerse fields are written to their mapped columns. This option may improve the performance of READ operations, especially when mapping multivalued attributes since you avoid complex outer-joins.

If the value of Whole Record is Yes, the entire UniVerse 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 choose the WHOLE_RECORD option, you should not be updating the data from the external database, and only update it through UniVerse BASIC or UniVerse SQL. A failure to comply with this rule can result in inconsistent data.
Unmapped field block (KB)

The Unmapped Field Block option defines the size of the Character Large Object (CLOB on DB2 databases), expressed in kilobytes. The default value is 16. You can change this value to meet your needs. This Character Large Object, called RECORD_BLOB, serves several purposes:

- To hold all attributes that are not explicitly mapped
- To hold the entire UniVerse record when the WHOLE_RECORD flag is set to “Yes”
- To hold nonconforming records

Nonconforming record

A nonconforming record is a UniVerse record that generates an exception error when it is written to the EDA server database, but does not cause an error in the UniVerse database. The Nonconforming Record option enables you to tell UniVerse not to return validation or truncation errors generated by the external database to your application, so the behavior of your application does not change. For example, UniVerse 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 UniVerse 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 UniVerse record is written to the RECORD_BLOB column. In this case, no error is returned to the UniVerse application. If UniVerse attempts to write the nonconforming data to a RECORD_BLOB that is not large enough to contain the data, the write fails and UniVerse writes the record to the EDA_EXCEPTION file on the UniVerse database and an error is returned to the UniVerse application. If the value of Nonconforming Record is No, the nonconforming data is only written to the EDA_EXCEPTION file on the UniVerse database and an error is returned to the UniVerse application.

For more information about retrieving nonconforming data, see SELECT.EDA.NONCONFORMING
Table space

Table Space is used for DB2 only. A Table Space is the basic storage structure on the DB2 database. By default, UniVerse 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, UniVerse generates an error during the conversion process.

You can change the default page size in the Table Space (KB) box. The maximum page size is 256 KB. 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.

View field details

To view the details of a mapped attribute, click the attribute you want to view from the EDA Map Schema portion of the dialog box. For example, the following window appears if you click the FNAME attribute:

```
The Attribute Details area of the screen displays information about the attribute you selected.

- The Name box displays the name of the mapped attribute.
```
- The **Type** box displays the type of attributes. Valid types at this release of UniVerse are:
  - 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 an I-descriptor, the I-descriptor is evaluated in UniVerse, and the result is stored in this column in the external database.
  - EXPRESSION – used for I-descriptors only. Enter the SQL expression for the I-descriptor in the **Expr Body** box, 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 I-descriptor on the external database. For information about creating a scalar function, see “Scalar function example” on page 33.
  - TABLE FUNCTION – used to define a table function. For information about creating a table function, see “Table function example” on page 38.
  - TRANS – used for I-descriptors containing a TRANS clause only. If you specify TRANS, you must also specify **Reference** and **Parameters**. For more information, see “TRANS example” on page 36.
  - UNIQUE DATA – specifies that values in the external database column must be unique

- The **Data Type** box displays the data type for the mapped attribute. The EDA Schema Manager automatically converts the data type based on the dictionary record. 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.

- The **Reference** box applies to an I-descriptor. This value should be the name of the column the TRANS function references in the external table, or the reference for a scalar or table function.
The **Expr Body** box applies to I-descriptors that contain a user-defined function or expression. Use the **Expr Body** box to enter the equivalent SQL statement for the expression or function.

The **Indexed** box indicates if you want to create an index on the mapped external attribute. If you want to specify the name of the external index, enter a phrase similar to the following example in the **Index** box:

```
REMOTE [DB2_indexname]
```

If you do not specify the external index name, the EDA process generates one.

The **Parameters** box specifies the attribute or expression in the function that returns the record ID in the table you are referencing in the external database.

### Scalar function example

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

```
: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.
To convert this I-descriptor to a scalar function using the EDA Tool, drag UPCODE.LNAME from the U2 File Dictionary pane under CUSTOMER in the EDA Map Schema pane, as shown in the following example:

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

Define the data type for the output in the **Data Type** box.

In the **Reference** box, enter the external function and data type for the value you are passing to the function. In this example, we are using the DB2 system function UCASE which corresponds to the UniVerse UPCASE function, which resides in the SYSFUN Schema in the DB2 database. Enter the following formula in the **Reference** box:

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

![Attribute Details](image)

The following example illustrates the output from UniVerse when you execute 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            MCCALG
   8             HOLLAND
   12            PATRY
   1             SMITH
   9             ORLANDO
   11            LEMIS

12 records listed.
```
**TRANS example**

The next example illustrates a TRANS function. Assume you have the following UniVerse I-descriptor defined in the dictionary of the CUSTOMER file:

```
001: I Product description
002: TRANS(PRODUCTS, PRODID, DESCRIPTION, 'C')
003:
004: Product Description
005: 20T
006: M
007: ORDERS
```

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

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:
Click DESCRIPTION in the EDA Map Schema portion of the window to define the Attribute Details for this function. The following example illustrates the details of the DESCRIPTION function:

In the Attribute Details portion of the window, change the Type to TRANS. Define the data type for the output in the Data Type box.

In the Reference box, enter the name of the external table that contains the DESCRIPTION information. In this example, the information resides in PRODUCTS.PRODUCTS/DESCRIPTION.

In the Parameters box, 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 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
3 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:
```

**Table function example**

Table functions are valid for the DB2 database only. UniVerse allows you to use the external table function concept to evaluate multiple I-descriptors at the same time. In some cases, you may be able to map more than one I-descriptor with one external 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:

:**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: | MD0,$ |
| 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 box, enter the data type for the output of the attribute you specified in the Name box.

In the Reference box, enter the external database schema name, the name of user-defined function you are defining in the Expr Body box, 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** box, enter the table function body. In this example, the function is defined as:

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

In the **Parameters** box, 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:

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

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

```
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:

![Attribute Details](image)

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

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

In the **Reference** box, 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:

```sql
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** box, 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.
Verifying the EDA schema

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

To verify the EDA Schema, click the type of verification to execute. Valid options are:

- **Syntax** – Verifies the syntax of the SQL statements to create 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 parentheses (})
  - First \( n \) records – the system verifies the first \( n \) records you specify.
  - Every \( n\)-th record – the system verifies every \( n\)-th record you specify.

**Verification example**

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

```
LIST CUSTOMER FNAME LNAME CITY STATE ZIP 11:52:59am 14 Jun 2010 PAGE 1
CUSTOMER.. First Name.. Last Name....... City........ State Zip..
2  Diana  Morris  Waltham  MA  01133
4  Jill    Kahn  Boston    MA  01103
6  Betty   Burke  White River VT  01644
3  David   Argonne  Bedford  MA  01182
5  Kenneth Williams  Providence RI  03171
7  Martha  Gill   Derry     NH  04429
10 Andrew  McCaig  Brattleboro VT  0356
8  Steven  Holland  Lowell    MA  01386
12 Laurie  Patry  Littleton  MA  0142
1  Samuel  Smith  Concord   NH  02131
9  Nicole  Orlando  Burlington MA  01173
11 Skip   Lewis  Plymouth  MA  01382
```

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 Chapter 6, “Chapter 6: EDA exception handling.”

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:
Viewing the DDL scripts

To view the DDL script that UniVerse will use to generate the data on the external database, click the DDL Scripts icon on the toolbar, as shown in the following example:

From the Get DDL Scripts dialog box, click Get DDL Scripts appear in the window, as shown in the following example:
Converting data

To convert data from UniVerse to the external database, click the Convert Data icon, as shown in the following example:

![Convert Icon](image-url)
A dialog box similar to the following example appears:

Click **Force** if you want 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 the DDL scripts during the conversion process.
Click EDA Convert. If the conversion is successful, UniVerse reports the number of records converted to DB2. If the conversion is not successful, error messages are reported in the window.

Viewing EDA files

To see which files have been converted from UniVerse to the external database, from the EDA Schema Manager, click the plus sign next to EDA Files, as shown in the following example:
## Listing and selecting data

You can use RetrieVe, UniVerse SQL, and UniVerse BASIC to access the data on the external server.

### Listing data using RetrieVe

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

```
LIST CUSTOMER FNAME LNAME CITY STATE
1
CUSTOMER.. First Name.. Last Name....... City........... State
2 Diana Morris Waltham MA
4 Jill Kahn Boston MA
3 Betty Burke White River Jun VT
6 David Argonne Bedford MA
5 Kenneth Williams Providence RI
7 Martha Gill Derry NH
10 Andrew McCall Brattleboro VT
8 Steven Holland Lowell MA
12 Laurie Patry Littleton MA
1 Samuel Smith Concord NH
9 Nicole Orlando Burlington MA
11 Skip Lewis Plymouth MA
```

12 records listed.
### Listing data using UniVerse SQL

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

```sql
SELECT FNAME, LNAME, CITY, STATE FROM CUSTOMER;
```

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>City</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morris</td>
<td>Waltham</td>
<td>MA</td>
<td></td>
</tr>
<tr>
<td>Kahn</td>
<td>Boston</td>
<td>MA</td>
<td></td>
</tr>
<tr>
<td>Burke</td>
<td>White River Jun</td>
<td>VT</td>
<td></td>
</tr>
<tr>
<td>Argonne</td>
<td>Bedford</td>
<td>MA</td>
<td></td>
</tr>
<tr>
<td>Williams</td>
<td>Providence</td>
<td>RI</td>
<td></td>
</tr>
<tr>
<td>Gill</td>
<td>Derry</td>
<td>NH</td>
<td></td>
</tr>
<tr>
<td>McCaig</td>
<td>Brattleboro</td>
<td>VT</td>
<td></td>
</tr>
<tr>
<td>Holland</td>
<td>Lowell</td>
<td>MA</td>
<td></td>
</tr>
<tr>
<td>Patry</td>
<td>Littleton</td>
<td>MA</td>
<td></td>
</tr>
<tr>
<td>Smith</td>
<td>Concord</td>
<td>NH</td>
<td></td>
</tr>
<tr>
<td>Orlando</td>
<td>Burlington</td>
<td>MA</td>
<td></td>
</tr>
<tr>
<td>Lewis</td>
<td>Plymouth</td>
<td>MA</td>
<td></td>
</tr>
</tbody>
</table>

12 records listed.
Chapter 3: External database drivers supplied with EDA

EDA Oracle driver .......................... 3-2
EDA DB2 driver .............................. 3-5
EDA SQL Server driver .................... 3-7
EDA SQL Server driver for UNIX and Linux .............................. 3-9
EDA Oracle driver

UniVerse provides an EDA Oracle driver at this release. 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.

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

```
ORDEVDB=
  (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCF)(HOST=test.com)(PORT = 1521)
  (CONNECT_DATA =
   (SERVER = DEDICATED)
   (SERVICE_NAME = ORDEVDB)
  )
  )
```

On UNIX platforms, the tnsnames.ora file must reside in the directory you specified as ORACLEPATH in the edaconfig file.

On Windows platforms, the tnsnames.ora file must reside in $ORACLE_HOME\NETWORK\ADMIN.
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>

| Oracle Path Locations by Platform |

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 ORDEVDEB.
## Oracle data type mapping

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

<table>
<thead>
<tr>
<th>UniVerse 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>Unmapped fields</td>
<td>CLOB</td>
</tr>
</tbody>
</table>

*Note: When mapping UniVerse time data to SQL Server DATETIME, the date portion is filled with 01/01/2000.*
EDA DB2 driver

UniVerse provides an EDA DB2 driver at this release. 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.

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:

```bash
DRIVER=DB2
DB2PATH=/home/db2inst1/sqlib
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 UniVerse is installed. Create a database on the DB2 server to which UniVerse can connect. If you install the DB2 client after you install UniVerse, 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 UniVerse data to DB2 data types:

<table>
<thead>
<tr>
<th>UniVerse 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>

Mapping UniVerse Data to Oracle Data Type
EDA SQL Server driver

UniVerse provides an EDA SQL Server driver at this release. 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 support SQL Server 2005 and greater.

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

Install SQL Server and create ODBC data source

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

Create a database on SQL Server to which UniVerse 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 and 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 $uvhome for the EDA SQL Server driver log. Add the following line 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 UniVerse data to SQL Server data types.

<table>
<thead>
<tr>
<th>UniVerse Data</th>
<th>SQL Server Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characters</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>Date</td>
<td>DATETIME</td>
</tr>
<tr>
<td>Time</td>
<td>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:** When mapping UniVerse date data to SQL Server DATETIME, the time portion is filled with 00:00:00.

When mapping UniVerse time data to SQL Server DATETIME, the date portion is filled with 01/01/1753.
EDA SQL Server driver for UNIX and Linux

UniVerse 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 UniVerse 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.

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 UniVerse 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.

UniVerse 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
```

UniVerse 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          =
Setup            = /usr/local/easysoft/sqlserver/lib/libessqlsrv.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 4.

If you connect to SQL Server, the data types are described in “SQL Server data types” on page 8.

If you connect to DB2, the data types are described in “DB2 data type mapping” on page 6.
# Chapter 4: External Database Access Driver API

External Database Access Driver API ............................................. 4-2  
Registering an EDA driver ....................................................... 4-3  
EDA Driver functions .............................................................. 4-4  
  EDADRV_LoadSymbols .......................................................... 4-5  
  EDADRV_Connect ............................................................... 4-8  
  EDADRV_Disconnect ........................................................... 4-9  
  EDADRV_EndTransaction ..................................................... 4-10  
  EDADRV_PrepareStmt ......................................................... 4-11  
  EDADRV_ExecuteStmt ......................................................... 4-13  
  EDADRV_CloseStmt ............................................................ 4-15  
  EDADRV_DropStmt ............................................................. 4-16  
  EDADRV_FetchStmt ............................................................ 4-17  
  EDADRV_Perform ............................................................... 4-19  
  EDADRV_GetEDAAttr ........................................................... 4-21  
  EDADRV_GetErrMsg ............................................................ 4-23  
  EDADRV_Cleanup ............................................................. 4-24  
  EDADRV_FreeResult ........................................................... 4-24  
  EDADRV_GetDBInfo ............................................................ 4-25  
  EDADRV_GetSpecialInfo ....................................................... 4-27  
ECL Command ........................................................................... 4-30  
EDA Driver log files .................................................................. 4-31  
The EDA Driver header file ......................................................... 4-32
External Database Access (EDA) enables you to convert data stored in the UniVerse database to a 1NF database, such as IBM DB2, Oracle, and SQL Server, then access that data using existing UniVerse BASIC programs, Retrieve, or 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 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 No</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>■ IBMU2</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><em>Note: Do not include the .dll extension when defining the name of the driver.</em></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 Attributes
# 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>

**EDA Driver Functions**
EDADRV_LoadSymbols

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

Syntax

RETCODE EDA_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>

Description

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.
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.

<table>
<thead>
<tr>
<th>Array Index</th>
<th>Array Element Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EDASYM_DBTYPE</td>
<td>The driver’s data model. At this release, UniVerse only supports EDA_1NF.</td>
</tr>
<tr>
<td>1</td>
<td>EDASYM_DBFAMILY</td>
<td>The name of the driver database, such as DBMS_DB2, DBMS_MS-SQLSERVER, DBMS_OTHERS. For more information, see “Registering an EDA driver” on page 3.</td>
</tr>
<tr>
<td>2</td>
<td>EDASYM_VERSION</td>
<td>The version of this driver. This is for information purposes only, EDA does not use this value.</td>
</tr>
<tr>
<td>3</td>
<td>EDASYM_CONNECT</td>
<td>The pointer to EDADRV_Connect.</td>
</tr>
<tr>
<td>4</td>
<td>EDASYM_DISCONNECT</td>
<td>The pointer to EDADRV_Disconnect.</td>
</tr>
<tr>
<td>5</td>
<td>EDASYM_END_TRANSACTION</td>
<td>The pointer to EDADRV_EndTransaction.</td>
</tr>
<tr>
<td>6</td>
<td>EDASYM_PREPARE_STMT</td>
<td>The pointer to EDADRV_PrepareStmt.</td>
</tr>
<tr>
<td>7</td>
<td>EDASYM_DROP_STMT</td>
<td>The pointer to EDADRV_DropStmt.</td>
</tr>
<tr>
<td>8</td>
<td>EDASYM_EXECUTE_STMT</td>
<td>The pointer to EDADRV_Execute.</td>
</tr>
<tr>
<td>9</td>
<td>EDASYM_CLOSE_STMT</td>
<td>The pointer to EDADRV_Close.</td>
</tr>
<tr>
<td>10</td>
<td>EDASYM_FETCH_STMT</td>
<td>The pointer to EDADRV_FetchStmt.</td>
</tr>
<tr>
<td>11</td>
<td>EDASYM_PERFORM</td>
<td>The pointer to EDADRV_Perform.</td>
</tr>
</tbody>
</table>

**Driver Functions**
Return codes

The following table describes the EDADRV_LoadSymbols 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>108</td>
<td>EDADRV_CONFIG_NOT_EXIST</td>
<td>edaconfig file does not exist.</td>
</tr>
<tr>
<td>110</td>
<td>EDADRV_DB2INSTANCE_NOT_SET</td>
<td>DB2INSTANCE in edaconfig file not set.</td>
</tr>
</tbody>
</table>
EDADRV_Connect

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

**Syntax**

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 variable**

The following table describes the output variable.

<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>

**Description**

The EDADRV_Connect function makes a connection to an external database.
**Return codes**

The following table describes the EDADRV_Connect 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.

**Syntax**

RETCode EDADRV_Disconnect(connhdl)

**Input variable**

The following table describes the input variable.

<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>

**Description**

The EDADRV_Disconnect function disconnects from an external database and releases the connection handle.
**Return codes**

The following table describes the EDADRV_Disconnect 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>
</tbody>
</table>

**EDADRV_EndTransaction**

EDA calls the EDADRV_EndTransaction function to end a transaction on an external database.

**Syntax**

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>connhdl</td>
<td>The connection handle.</td>
</tr>
</tbody>
</table>
| int                 | trans_flag | The action to take if the transaction ends. Valid values are:  
|                     |           |   - 0 – EDA_COMMIT. Commit the transaction.  
|                     |           |   - 1 – EDA_ROLLBACK. Rollback the transaction. |

**EDADRV_EndTransaction Input Variables**
Description

The EDADRV_EndTransaction function commits or rolls back a transaction on the external database, depending on the value of trans_flag.

Return codes

The following table describes the EDADRV_EndTransaction 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_EndTransaction Return Codes

EDADRV_PrepareStmt

EDA calls the EDADRV_PrepareStmt function to prepare a statement.

Syntax

RETCODE EDADRV_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>

EDADRV_PrepareStmt Input Variables
Output variable

The following table describes the output variable.

<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>

Description

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 EDADRV_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 EDADRV_ExecuteStmt.
Return codes

The following table describes the EDADRV_PrepareStmt 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>112</td>
<td>EDADRV_INVALID_DATATYPE</td>
<td>Invalid data type</td>
</tr>
</tbody>
</table>

EDADRV_PrepareStmt Return Codes

EDADRV_ExecuteStmt

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

Syntax

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>

EDADRV_ExecuteStmt Input Variables
Output Variable

The following table describes the output variable.

<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>

Description

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 EDADRV_GetADDAttr function. The EDADRV_ExecuteStmt function binds each parameter and executes a statement that has already been prepared with the EDADRV_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.

Return codes

The following table describes the EDADRV_ExecuteStmt 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>
</tbody>
</table>
EDA Driver functions 4-15

EDA calls the EDADRV_CloseStmt function to close a statement.

**Syntax**

RETCODE EDADRV_CloseStmt(stmthdl)

**Input variable**

The following table describes the input parameter.

<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>

**Description**

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.

EDADRV_ExecuteStmt Return Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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
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.

Return codes

The following table describes the EDADRV_CloseStmt 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_CloseStmt Return Codes

EDADRV_DropStmt

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

Syntax

RETCODE EDADRV_DropStmt(stmthdl)

Input variable

The following table describes the input parameter.

<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>

EDADRV_DropStmt
Description

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.

Return codes

The following table describes the EDADRV_DropStmt 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 Return Codes

EDADRV_FetchStmt

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

Syntax

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:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 0 – Fetch Forward</td>
</tr>
<tr>
<td>int</td>
<td>numrows</td>
<td>The number of rows to fetch.</td>
</tr>
</tbody>
</table>

EDADRV_FetchStmt Input Variables

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>

EDADRV_FetchStmt Output Variables

Description

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.
**Return codes**

The following table describes the EDADRV_FetchStmt 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>
<tr>
<td>104</td>
<td>EDADRV_INVALID_FETCH_DIR</td>
<td>Invalid fetch direction.</td>
</tr>
</tbody>
</table>

**EDADRV_Perform**

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

**Syntax**

RETCODE EDADRV_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>

**Description**

The EDADRV_Perform function combines the processing of the EDADRV_PrepareStmt and the EDADRV_Execute functions, and if the statement is a query, it also fetches the entire result set as in EDADRV_FetchStmt. See the above functions for a description of the processing performed by the EDADRV_Perform function. The driver designer may choose to either Prepare and Execute or Execute Direct on the external database side.

**Return codes**

The following table describes the EDADRV_Perform 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_SYSERV_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>102</td>
<td>EDADRV_INVALID_CONN_ID</td>
<td>Invalid connection handle.</td>
</tr>
</tbody>
</table>
EDA calls the EDADRV_GetEDAAttr function to communicate an EDA attribute value to the driver.

**Syntax**

```
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>attribute_type</td>
<td>The name of the EDA driver attribute.</td>
</tr>
<tr>
<td>EDA_T_SYMBOL</td>
<td>value</td>
<td>The value for the EDA driver attribute.</td>
</tr>
</tbody>
</table>

**Description**

The EDADRV_GetEDAAttr function receives an attribute name – value pair.
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>EDADRV_T_STR</td>
<td>EDADRV_ATTR_SYSNAME</td>
<td>Valid values are UniData or UniVerse.</td>
</tr>
<tr>
<td>EDADRV_T_STR</td>
<td>EDADRV_ATTR_VERSION</td>
<td>Valid values are 7.1 or 7.2.</td>
</tr>
<tr>
<td>EDADRV_T_INT</td>
<td>EDADRV_ATTR_RM</td>
<td>The ASCII character representing a record mark (RM), such as ^255.</td>
</tr>
<tr>
<td>EDADRV_T_INT</td>
<td>EDADRV_ATTR_FM</td>
<td>The ASCII character representing a field mark (FM), such as ^254.</td>
</tr>
<tr>
<td>EDADRV_T_INT</td>
<td>EDADRV_ATTR_VM</td>
<td>The ASCII character representing a value mark (VM), such as ^253.</td>
</tr>
<tr>
<td>EDADRV_T_INT</td>
<td>EDADRV_ATTR_SM</td>
<td>The ASCII character representing a subvalue mark (SM), such as ^252.</td>
</tr>
<tr>
<td>EDADRV_T_INT</td>
<td>EDADRV_ATTR_TM</td>
<td>The ASCII character representing a text mark (TM), such as ^251.</td>
</tr>
<tr>
<td>EDADRV_T_INT</td>
<td>EDADRV_NULLVAL</td>
<td>The ASCII character representing the null value.</td>
</tr>
</tbody>
</table>

**EDADRV_GetEDAAttr Values**

**Return codes**

The following table describes the EDADRV_GetEDAAttr 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>EDADRV_INVALID_DRV_ATTR</td>
<td>Invalid driver attribute.</td>
</tr>
</tbody>
</table>

**EDADRV_GetEDAAttr Codes**
EDA calls the EDADRV_GetErrmsg function to retrieve the last error message.

**Syntax**

RETCODE EDADRV_GetErrmsg(errmsg)

**Output variable**

The following table describes the output variable.

<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>

**Description**

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.

**Return codes**

The following table describes the EDADRV_GetErrmsg 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>
**EDADRVR_Cleanup**

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

**Syntax**

RETCODE EDADRVR_Cleanup

**Description**

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.

**Return codes**

The following table describes the EDADRVR_Cleanup 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>

**EDADRVR_FreeResult**

EDA calls the EDADRVR_FreeResult function to free the buffer allocated for the result set.

**Syntax**

RETCODE EDADRVR_FreeResult(buf)
Input variable

The following table describes the input variable.

<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>

Description

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.

Return codes

The following table describes the EDADRV_FreeResult 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

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>▪ EDA_DRV_DBMS_NAME – The name of the database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ EDA_DRV_DBMS_VERSION – The database version.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ EDA_DRV_SERVER_NAME – The name of the instance on the external database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ EDA_DRV_DATABASE_NAME – The name of the database.</td>
</tr>
</tbody>
</table>

**EDADRV_GetDBInfo Input Variables**

**Output variable**

The following table describes the output variable.

<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>

**EDADRV_GetDBInfo Output Variable**

**Description**

The EDA_DRV_GetDBInfo function returns general information about the database to which the application is currently connected.
**Return codes**

The following table describes the EDADRV_GetDBInfo 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.

**Syntax**

RETCODE EDADRV_GetSpecialInfo(connhdl, infotype, parameters, dbinfo)
### Input variables

The following table describes the input variables for the EDADRV_GetSpecialInfo function.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connhdl</td>
<td>The connection handle.</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 – Returns the external database statement of rename table.  
- 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 variable

The following table describes the output variable for the EDADRV_GetSpecialInfo function.

<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>

---

4-28 External Database Access (EDA)
EDA Driver functions 4-29

Description

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.

Return codes

The following table describes the EDADRV_GetSpecialInfo 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>

EDADRV_GetSpecialInfo Return Codes
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

You can enable logging of the EDA Driver by setting the LOGLEVEL value in the edaconfig file to 1 or 2. The edaconfig file is located in $UVHOME.

You can set the following log levels:

1 – Logs the status of the EDA Driver API functions and the input/output parameters.

2 – Logs the status of the CLI or ODBC functions and the input/output parameters.

The name of the log is EDADRV.pid, and is located in the /tmp directory on UNIX platforms, or $UVHOME\Temp on Windows platforms.
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 ifndef edadriv_public_H_DEFINED
#define edadriv_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 EDADRV_LOAD_SYMBOLS.
* EDADRV_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 */
/
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;
/
#define EDAT_FUNC0
#define EDAT_INT1
#define EDAT_STRING2
```
typedef struct EDA_T_STRING {
  char * ata; /* ata transferring between driver API */
} EDA_T_STRING;

typedef EDA_T_STRING EDA_T_RESULT;

typedef struct EDA_T_PTYPE {
  short inout; /* IN/OUT */
  short datatype; /* parameter datatype */
} EDA_T_PTYPE; /* define a parameter type */

#define EDAPARAM_IN0 /* Input parameter */
#define EDAPARAM_OUT1 /* Output parameter */
#define EDAPARAM_INOUT2 /* In/Out Parameter */

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 */

#define EDASTMT_SQL_DDL1 /* SQL DDL statement */
#define EDASTMT_SQL_DML2 /* SQL DML statement */
#define EDASTMT_SQL_QUERY3 /* SQL QUERY statement */
#define EDASTMT_PROCEDURE4 /* Stored Procedure */
#define EDASTMT_XQUERY5 /* XQUERY statement */

typedef EDA_T_CONN_HDL;
#define INVALID_CONN_HDL(hdl) ((hdl)<0)
#define CLEAR_CONN_HDL(hdl) ((hdl)=-1) /* Clear a handle */

typedef EDA_T_STMT_HDL;
#define INVALID_STMT_HDL(hdl) ((hdl)<0)
#define CLEAR_STMT_HDL(hdl) ((hdl)=-1) /* Clear handle */

typedef long EDA_T_CONN_HDL;
typedef long EDA_T_STMT_HDL; /* Positive integer */

#define EDA_T_STRING EDA_T_RESULT;
#define EDA_T_PTYPE EDA_T_RESULT;
#define EDA_T_STMT EDA_T_RESULT;
#define EDA_T_CONN_HDL EDA_T_RESULT;
#define EDA_T_STMT_HDL EDA_T_RESULT;

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

typedef long EDA_T_STMT_HDL; /* Positive integer */
#define INVALID_STMT_HDL(hdl) ((hdl)<0)
#define CLEAR_STMT_HDL(hdl) ((hdl)=-1) /* Clear handle */

typedef struct EDA_T_STRING {
  char * ata; /* ata transferring between driver API */
} EDA_T_STRING;

typedef EDA_T_STRING EDA_T_RESULT;

typedef struct EDA_T_PTYPE {
  short inout; /* IN/OUT */
  short datatype; /* parameter datatype */
} EDA_T_PTYPE; /* define a parameter type */

#define EDAPARAM_IN0 /* Input parameter */
#define EDAPARAM_OUT1 /* Output parameter */
#define EDAPARAM_INOUT2 /* In/Out Parameter */

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 */

#define EDASTMT_SQL_DDL1 /* SQL DDL statement */
#define EDASTMT_SQL_DML2 /* SQL DML statement */
#define EDASTMT_SQL_QUERY3 /* SQL QUERY statement */
#define EDASTMT_PROCEDURE4 /* Stored Procedure */
#define EDASTMT_XQUERY5 /* XQUERY statement */

typedef EDA_T_CONN_HDL;
#define INVALID_CONN_HDL(hdl) ((hdl)<0)
#define CLEAR_CONN_HDL(hdl) ((hdl)=-1) /* Clear a handle */

typedef EDA_T_STMT_HDL;
#define INVALID_STMT_HDL(hdl) ((hdl)<0)
#define CLEAR_STMT_HDL(hdl) ((hdl)=-1) /* Clear handle */

typedef struct EDA_T_STRING {
  char * ata; /* ata transferring between driver API */
} EDA_T_STRING;

typedef EDA_T_STRING EDA_T_RESULT;

typedef struct EDA_T_PTYPE {
  short inout; /* IN/OUT */
  short datatype; /* parameter datatype */
} EDA_T_PTYPE; /* define a parameter type */

#define EDAPARAM_IN0 /* Input parameter */
#define EDAPARAM_OUT1 /* Output parameter */
#define EDAPARAM_INOUT2 /* In/Out Parameter */

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 */

#define EDASTMT_SQL_DDL1 /* SQL DDL statement */
#define EDASTMT_SQL_DML2 /* SQL DML statement */
#define EDASTMT_SQL_QUERY3 /* SQL QUERY statement */
#define EDASTMT_PROCEDURE4 /* Stored Procedure */
#define EDASTMT_XQUERY5 /* XQUERY statement */

typedef EDA_T_CONN_HDL;
#define INVALID_CONN_HDL(hdl) ((hdl)<0)
#define CLEAR_CONN_HDL(hdl) ((hdl)=-1) /* Clear a handle */

typedef EDA_T_STMT_HDL;
#define INVALID_STMT_HDL(hdl) ((hdl)<0)
#define CLEAR_STMT_HDL(hdl) ((hdl)=-1) /* Clear handle */
**Driver Loading Function Name**

```c
#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**
  - `0`: Succeeded;
  - `errcode` if failed.
- **GLOBAL**

---

**Driver Symbol Codes**

```c
#define EDA_SYMBOL_NUM17
```

---

**Driver Data Type Code: Driver Data Type**

```c
#define EDASYM_DBTYPE0 /* Integer: Driver Data Type code */
#define EDASYM_DBFAMILY1 /* Integer: Driver DBMS Family code */
#define EDASYM_VERSION2 /* String: the version string */
#define EDASYM_CONNECT3 /* Function: Make Connection */
#define EDASYM_DISCONNECT4 /* Function: Disconnect */
#define EDASYM_END_TRANSACTIONS5 /* Function: End Transaction */
#define EDASYM_PREPARE_STMT6 /* Function: Prepare Statement */
#define EDASYM_DROP_STMT7 /* Function: Drop Statement */
#define EDASYM_EXECUTE_STMT8 /* Function: Execute Statement */
#define EDASYM_CLOSE_STMT9 /* Function: Close Statement */
#define EDASYM_FETCH_STMT10 /* Function: Fetch Statement */
#define EDASYM_PERFORM11 /* Function: Perform a command */
#define EDASYM_GET_EDA_ATTR12 /* Function: Get EDA attribute */
#define EDASYM_GET_ERRMSG13 /* Function: Get last error message */
#define EDASYM_CLEANUP14 /* Function: Cleanup the driver */
#define EDASYM_GET_DBINFO15 /* Function: Get external DB info */
#define EDASYM_FREE_RESULT16 /* Function: Free result buffer */
#define EDASYM_GET_SPECIALINFO17 /* Function: Get special info */
```

---

**Driver DBMS Family Code: Driver DBMS Family**

```c
#define EDA_UNKNOW 0
#define EDA_INF 1 /* Relational Database */
#define EDA_NF2 2 /* Multi-valued Database */
#define EDA_XML 3 /* XML Database */
#define EDA_OBJ 4 /* Object Database */
```

---

**Driver Data Type Code: Driver Data Type**

```c
#define EDASYM_DBTYPE0 /* Integer: Driver Data Type code */
#define EDASYM_DBFAMILY1 /* Integer: Driver DBMS Family code */
#define EDASYM_VERSION2 /* String: the version string */
#define EDASYM_CONNECT3 /* Function: Make Connection */
#define EDASYM_DISCONNECT4 /* Function: Disconnect */
#define EDASYM_END_TRANSACTIONS5 /* Function: End Transaction */
#define EDASYM_PREPARE_STMT6 /* Function: Prepare Statement */
#define EDASYM_DROP_STMT7 /* Function: Drop Statement */
#define EDASYM_EXECUTE_STMT8 /* Function: Execute Statement */
#define EDASYM_CLOSE_STMT9 /* Function: Close Statement */
#define EDASYM_FETCH_STMT10 /* Function: Fetch Statement */
#define EDASYM_PERFORM11 /* Function: Perform a command */
#define EDASYM_GET_EDA_ATTR12 /* Function: Get EDA attribute */
#define EDASYM_GET_ERRMSG13 /* Function: Get last error message */
#define EDASYM_CLEANUP14 /* Function: Cleanup the driver */
#define EDASYM_GET_DBINFO15 /* Function: Get external DB info */
#define EDASYM_FREE_RESULT16 /* Function: Free result buffer */
#define EDASYM_GET_SPECIALINFO17 /* Function: Get special info */
```

---

**Section 3: Definition of EDA API Functions**
The EDA Driver header file 4-35

---

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;
ercode 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;
ercode 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;
ercode if failed.
GLOBAL:

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:

---

#define EDA_COMMIT0
#define EDA_ROLLBACK1

---
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:
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:
DESCRIPTION: EDADRV_DropStmt drop a statement. 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

GLOBAL:
*******************************************************************************
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*******************************************************************************/

/** 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_INF: 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 :
*******************************************************************************/

#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 :
*******************************************************************************/

/*******************************************************************************/

/** Function Symbol: EDASYM_GET_EDA_ATTR
 */

The EDA Driver header file 4-37
* FUNCTION     : EDADRv_GetEDAAttr: Get attribute from EDA engine.
* DESCRIPTION: Get attribute from U2 EDA engine.
* CALLED BY: Driver Manager
* PARAMETERS :
  * int attrname;IN: driver attribute name, defined below
  * EDA_T_SYMBOL value;IN: driver attribute value.
* RETURN :
  * 0: Succeeded;
  * errcode if failed.
* GLOBAL :

********************************************************************
** Driver Attribute Names ************
#define EDADRv_ATTR_SYSNAME/* EDADRv_T_STR: UniData or UniVerse*/
#define EDADRv_ATTR_VERSION/* EDADRv_T_STR: U2 Version string */
#define EDADRv_ATTR_RM/* EDADRv_T_INT: U_RM */
#define EDADRv_ATTR_FM/* EDADRv_T_INT: U_FM */
#define EDADRv_ATTR_VMS/* EDADRv_T_INT: U_VM */
#define EDADRv_ATTR_SM/* EDADRv_T_INT: U_SM */
#define EDADRv_ATTR_TM/* EDADRv_T_INT: U_TM */
#define EDADRv_ATTR_NULLVAL/* EDADRv_T_INT: U_NULLVAL */
/* More to be defined */
********************************************************************

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: EDAASYM_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.
int infotype;IN: information type
EDA_T_RESULT *dbinfo;OUT: the DB info.
RETURN :
0: Succeeded;
errcode if failed.
GLOBAL:
*********************************************************************/

******* Infotypes in EDADRV_GetDBInfo *******
#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: EDAASYM_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_MEMALLOC101/* Internal memory allocation error */
#define EDADRV_INVALID_CONN_ID102/* Invalid connection handle */
#define EDADRV_INVALID_STMT_ID103/* Invalid statement handle */
#define EDADRV_INVALID_FETCH_DIR104/* Invalid fetch direction */
#define EDADRV_INVALID_EDA_ATTRIB105/* Invalid EDA attribute */
#define EDADRV_TOO_MANY_DATA106/* Too many data fetched */
#define EDADRV_GET_UDTHOME_ERROR107/* Get UDTHOME error */
#define EDADRV_CONFIG_NOT_EXIST108/* edaconfig file not exist */
#define EDADRV_OPEN_CONFIG_ERROR109/* Open edaconfig error */

The EDA Driver header file 4-39
#define EDADRV_DB2INSTANCE_NOT_SET 110/* DB2INSTANCE not set */
#define EDADRV_INVALID_PARAM_TYPE 111/* Invalid parameter type */
#define EDADRV_INVALID_DATATYPE 112/* Invalid data type */
#define EDADRV_TOO_MANY_OUT_PARAM 113/* Too many output parameters */
#define EDADRV_INVALID_TRANS_FLAG 114/* Invalid end transaction flag */
#define EDADRV_INVALID_INFOTYPE 115/* Invalid information type */
/* More to be defined */

#endif
Chapter 5: EDA ECL commands

Connecting to the external database . . . . . . . . . . 5-2
Disconnecting from the external database . . . . . . . . 5-3
Converting UniVerse data to an external database . . . . 5-3
VERIFY.EDAMAP . . . . . . . . . . . . . . . . 5-4
LIST.EDAMAP . . . . . . . . . . . . . . . . . 5-5
SAVE.EDAMAP . . . . . . . . . . . . . . . . . 5-7
SELECT.EDA.NONCONFORMING . . . . . . . . . . 5-8
Retrieving information about the EDA driver . . . . . . . 5-9
This chapter describes ECL commands you can use to connect to external databases and to convert UniVerse data to an external database, to verify your EDA Schema, to list your EDA Schema, to save your EDA Schema, and view nonconforming UniVerse records.

## Connecting to the external database

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, UniVerse maintains the connection until you issue the EDA.DISCONNECT command.

### Syntax

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

### Parameters

The following table describes each parameter of the syntax.

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

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.

Converting UniVerse data to an external database

Use the EDA.CONVERT command to convert UniVerse 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 UniVerse file in the UniVerse database.

If the UniVerse 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.

Syntax

```
EDA.CONVERT {[XMAP] eda_schema | EDA.FILE [DICT] eda_file | DEFAULT.MAP} [DATA.SOURCE data_source] [OBJECT.SET [name_space.]primary_table] [FILE.NAME target_file] [FORCE | VERBOSE]
```
Parameters

The following table describes each parameter of the syntax.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eda_schema</code></td>
<td>Specifies the name of the EDA schema to use for the conversion. The schema resides either in the &amp;EDAMAP&amp; or &amp;EDAXMAP&amp; file.</td>
</tr>
<tr>
<td><code>eda_file</code></td>
<td>Specifies the name of the EDA file from which UniVerse extracts the EDA schema. If you specify FILE.NAME <code>target_file</code>, UniVerse uses the schema to convert <code>target_file</code>, UniVerse remaps <code>eda_file</code>.</td>
</tr>
<tr>
<td><code>DEFAULT.MAP</code></td>
<td>Specifies only to map the primary key (@ID) when converting a UniVerse file to EDA.</td>
</tr>
<tr>
<td><code>data_source</code></td>
<td>Specifies the data source name to use for the conversion.</td>
</tr>
<tr>
<td><code>primary_table</code></td>
<td>Specifies the name of the primary table, containing singlevalued attributes, to use for the conversion. If you also specify <code>name_space</code>, UniVerse uses it as the external schema name for the target external table/view set.</td>
</tr>
<tr>
<td><code>target_file</code></td>
<td>Specifies the name of the UniVerse file to convert. If you also specify <code>eda_schema</code>, <code>target_file</code> overrides the name of the UniVerse file contained in <code>eda_schema</code>. If you specify <code>eda_file</code>, UniVerse extracts the EDA schema from <code>eda_file</code> and uses it to convert <code>target_file</code> to EDA.</td>
</tr>
<tr>
<td><code>FORCE</code></td>
<td>Specifies that all existing external tables, views, indexes and user-defined functions are dropped prior to remapping the file.</td>
</tr>
<tr>
<td><code>VERBOSE</code></td>
<td>Displays the external Data Definition Language (DDL) used in the conversion process.</td>
</tr>
</tbody>
</table>

EDA.CONVERT Parameters

**VERIFY.EDAMAP**

The VERIFY.EDAMAP command verifies the EDA schema.
Syntax

VERIFY.EDAMAP [\{XMAP\} eda_schema | EDA.FILE [DICT] eda_file | DEFAULT.MAP] [DATA.SOURCE data_source] [OBJECT.SET \{name_space.\}primary_table] [FILE.NAME target_file [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>eda_schema</td>
<td>Specifies the name of the EDA schema to verify.</td>
</tr>
<tr>
<td>eda_file</td>
<td>Specifies the name of the EDA file whose schema is to be extracted and verified. If you specify FILE.NAME target_file, target_name replaces the UniVerse file name in the schema UniVerse 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>data_source</td>
<td>Specifies the data source name to use when verifying the schema.</td>
</tr>
<tr>
<td>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 name_space, UniVerse uses it as the external schema name.</td>
</tr>
<tr>
<td>target_file</td>
<td>Specifies the name of the UniVerse 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>

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] [DATA.SOURCE 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 UniVerse file name in the schema UniVerse displays.</td>
</tr>
<tr>
<td>DEFAULT.MAP</td>
<td>Specifies to only display the primary key (@ID), irrespective of the attributes actually mapped of the schema you specify.</td>
</tr>
<tr>
<td>data_source</td>
<td>Specifies the data source name to use when displaying the schema.</td>
</tr>
<tr>
<td>primary_table</td>
<td>Specifies the name of the primary table, containing only singlevalued attributes, to use when displaying the schema. If you also specify name_space, UniVerse uses it for Name Space (External Schema Name) in the display.</td>
</tr>
<tr>
<td>target_file</td>
<td>Specifies the name of the UniVerse file to use when displaying the schema.</td>
</tr>
<tr>
<td>XMAP</td>
<td>Specifies to display the EDA schema in XML format.</td>
</tr>
<tr>
<td>OBJECT.TREE</td>
<td>Specifies to display the logical tree structure of the external table and view.</td>
</tr>
<tr>
<td>DDL</td>
<td>Specifies to display the external Data Definition Language (DDL) statements used in the conversion process.</td>
</tr>
</tbody>
</table>

LIST.EDAMAP Parameters
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} [DATA.SOURCE data_source] [OBJECT SET [name_space.]primary_table] [FILE.NAME 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>UniVerse file name in the schema UniVerse displays.</td>
</tr>
<tr>
<td>DEFAULT.MAP</td>
<td>Specifies to only save the primary key (@ID) mapping,</td>
</tr>
<tr>
<td></td>
<td>irrespective of the attributes actually mapped of the schema you</td>
</tr>
<tr>
<td></td>
<td>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, UniVerse uses it for Name Space</td>
</tr>
<tr>
<td></td>
<td>(External Schema Name).</td>
</tr>
</tbody>
</table>

SAVE.EDAMAP Parameters
The SELECT.EDA.NONCONFORMING command creates a select list of all nonconforming UniVerse 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 UniVerse 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:

```plaintext
: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 CUSTOMER 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).

Retrieving information about the EDA driver

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:
External Database Access (EDA)

- 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
Chapter 6: EDA exception handling
Exceptions can occur when an INSERT, DELETE, or UPDATE operation fails on the external database tables, or when UniVerse 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 the EDA schema in Chapter 2, "Chapter 2: The EDA Schema Manager."

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 Nonconforming record in Chapter 2, "Chapter 2: The EDA Schema Manager."

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 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 UniVerse 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.
- UniVerse inserts or updates data to the EDA file at runtime.
- UniVerse deletes the EDA file record at runtime.
# Chapter 7: EDA Replication

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA Replication</td>
<td>7-2</td>
</tr>
<tr>
<td>Setting up a server</td>
<td>7-3</td>
</tr>
<tr>
<td>Create a new UniVerse server connection</td>
<td>7-4</td>
</tr>
<tr>
<td>Defining a data source</td>
<td>7-8</td>
</tr>
<tr>
<td>Defining EDA replication parameters</td>
<td>7-13</td>
</tr>
</tbody>
</table>
EDA Replication

At this release, EDA Replication is introduced. EDA Replication is useful if you want to maintain an account from which you can create reports. This new version of UniVerse 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 UniVerse. When you store data in UniVerse, it is simultaneously replicated to Oracle, IBM DB2, or Microsoft SQL Server. Use the replicated database for data-mining or reporting while you use UniVerse 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 UniVerse files to EDA files.
Setting up a server

To access the EDA Replication Config Tool, from the Start menu, select All Programs, then select Rocket U2, then select EDA Replication Config Tool. The EDA Replication Tool appears, as shown in the following example:
Create a new UniVerse server connection

To create a new UniVerse server connection, right-click U2 Servers, and from the context menu, select New U2 Server. A dialog box similar to the following example appears:

![Create a New U2 Server dialog box]

**Enter server name**

In the Name box, enter a unique identifier for the new server.

**Enter host name**

In the Host box, enter the network name of the host computer where the UniVerse database resides, or the IP address.
Select the database

From the U2 database options, select UniVerse.

If you want to define the protocol type, RPC port number, RPC service name, or the logon account, click Advanced. A dialog box similar to the following example appears:

![Dialog box example]

Protocol type

In the Protocol Type box, choose the type of communication you are using the server. You can choose Default, TCP/IP, or Lan Manager. The default is TCP/IP.
**RPC port number**

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

**RPC service name**

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

**Logon account**

In the **Logon Account** box, enter the name of the account to which you want to log on when accessing UniVerse.

**Commands to execute**

Click **Add** in the **Commands to Execute** box to enter commands you want to execute when you log on to the server. The following dialog box appears:

![Create a New U2 Server](image)

**Specify session to run/debug UniVerse BASIC programs**

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

In the **Port Number** box, enter the port number you want to use, if you do not want to use the default port number of 23.
Select the **Use Device License** check box if you want to enable device licensing when connecting to the server.

Click **Finish** to establish the UniVerse server. The new server appears in the **U2 Resource** view of the **EDA Replication Config Tool**.

**Connect to UniVerse server**

To connect to the UniVerse server, right-click the server name, then click **Connect**. The following dialog box appears:

In the **User ID** box, enter the User ID for the machine on which UniVerse is running. In the **Password** box, enter the password for this user.

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.
If you are using a proxy server, select the **Use Proxy Server** check box.

In the Proxy Host field, enter the name or IP address of the computer on which the proxy server is running.

In the **Proxy Port** field, enter the number of the port on which the proxy server listens for communication from UniData or UniVerse.

To connect to the U2 server, click **Finish**.

The Accounts and existing Data Source definitions appear in the **U2 Server** list, as shown in the following example:

```
<table>
<thead>
<tr>
<th>Servers (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniVerse local</td>
</tr>
<tr>
<td>&gt; Accounts (2)</td>
</tr>
<tr>
<td>&gt; EDA Data Sources</td>
</tr>
</tbody>
</table>
```

**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 UniVerse server, right-click **Data Sources**, then click **New EDA Data Source**. The **Create a New EDA Data Source** dialog box appears, as shown in the following example:

![Create a New EDA Data Source dialog box](image)

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, as shown in the following example:

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, as shown in the following example:

In the **Logon 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 UniVerse database operation, such as a READ or WRITE, corresponds to an external transaction.

In the **Qualified Users** box, enter the UniVerse user IDs of users who can access the external server from the UniVerse account using the external Logon User ID you specify. Separate the users by a “|” symbol. If all UniVerse users can access the external account, enter an asterisk (*).
The following example shows a completed **EDA Data Source** dialog box:

To test the connection to the external instance, click **Test**. If the connection is successful, a message similar to the following example appears:
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 uv 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 UniVerse with the Replication feature, UniVerse 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>

REP_FLAG Values

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, click Configure Replication System. The Configure Replication System dialog box appears, as shown in the following example:

From the Replication Systems list, select the system to which you want to replicate data. 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:

![Configure Replication System dialog box](image)

**Define replication system ID**

System ID is the name you define 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 characters: ~ ! @ $ % ^ & * - + . / \.

**Define host name**

The Host Name defines the host name of the replication system location. A system can have only one host name.
Define UniVerse version

In the Version box, select the version of UniVerse on the system location. The version number must be 112 or higher.

DHCP check box

The DHCP check box specifies 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.

Select auto resume

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

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.

Select Connect Authorization

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.

In order 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.
Define 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 time out 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.

If 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.

Define exception action

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

The account definition is automatically populated with the account you previously defined. To define a different account to which you want to replicate EDA data, click Add in the Account Definitions area. The Replication System Definition dialog box appears, as shown in the following example:

In the Account Name box, enter the name of the account. In the Account Path box, select the path to the account using the Account Path arrow. Click Finish to save the definitions.

To save your settings, click Save Changes.
Choosing files to replicate

Click Choose Files to Replicate. If you have not previously defined a group, the following message appears:
Click **Create** to define a replication group. The **Choose Files to Replicate** dialog box appears, as shown in the following example:

![Choose Files to Replicate dialog box]

**Defining group name**

Enter a unique name for the subscribing group in the **Group ID** box.

**Defining replication level**

In the **Level** box, select the level of replication. For EDA Replication, only FILE-level replication is supported.
Select files to publish

In the Files area of the Publishing Group Details dialog box, click Add to select the files you want to publish. A dialog box similar to the following example appears:
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.

Define replication distribution details

In the Distributions area, click Add to define replication distribution details. The Replication Distribution Details dialog box appears, as shown in the following example:

![Replication Distribution Details dialog box](image)

Select the local system using the System Name arrow.

Select the Replication mode you want to use. For information about types of UniVerse Replication, see Chapter 1, “Introduction and Terminology” in the UniVerse Data Replication manual. Click Finish.
Set 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 Chapter 2, “Installing and Configuring UniVerse Data Replication” in the UniVerse Data Replication manual.

To save your settings, click Save Changes.

Synchronizing replication files

Click Synchronize replication files. The Synchronize replication files dialog box appears, as shown in the following example:

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 account, UniVerse displays the following message box:

![EDA Replication Config Tool](image)

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

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

Make absolutely sure no users are updating the database when you synchronize the files.

When the synchronization is complete, UniVerse 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 attribute, 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:

![Create Default EDA Schemas dialog box](image)

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

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

```plaintext
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:

![U2 Dictionary Attributes dialog box](image)

When you have finished selecting the dictionary attributes for which you want to create schemas, click **Finish**.

UniVerse 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 dialog box](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 including DDL scripts

Click **Finish**. UniVerse 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

Map selected fields ........................................ 8-1
Avoid multiple multivalued associations .............. 8-1
Avoid restrictive data types ............................... 8-1
RECORD_BLOB ......................................... 8-2
Updating an EDA tile from the external database .... 8-3
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 UniVerse server. The more fields you select to map, the greater the chance for invalid data type errors, and the worse the performance.

Singlevalued fields have much less performance overhead than multivalued fields.

**Avoid multiple multivalued associations**

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

If you need to map multiple associations, use the WHOLE RECORD option to increase performance and avoid external database errors. See “Whole record” in Chapter 2, “Chapter 2: The EDA Schema Manager,” for more information.

**Avoid restrictive data types**

Mapping UniVerse 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) data type:

```sql
SELECT * FROM TEST WHERE field1 = “ABC”;
```
UniVerse will not return any results because ABC actually appears as “ABC       “ on the EDA server. If you choose VARCHAR(10) as the data type, Universe 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 UniVerse file. If the insert operation of the record ID fails, UniVerse 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 UniVerse converts these types of data, using OCONV to map the data to the external database and ICONV to use the data in UniVerse, different results may occur. For example, if you have two records with the record IDs of “1” and “1.0” in the UniVerse 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.

**RECORD_BLOB**

When you specify NONCONFORMING or WHOLE_RECORD, UniVerse stores unmapped fields or the whole record in a record blob on the external database. The default size of the record blob is 16K. Set the size of the record blob to the largest record size in the UniVerse file to avoid insert failures.
Updating an EDA tile 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 UniVerse 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 UniVerse 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 I-descriptor 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 Nonconforming record in Chapter 2, “Chapter 2: The EDA Schema Manager” for more information.