Rocket UniData

External Database Access (EDA)

Version 8.2.2

September 2020
UDT-822-EDA-1

Notices

Edition

Publication date: September 2020
Book number: UDT-822-EDA-1
Product version: Version 8.2.2

Copyright

Trademarks

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

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.

Corporate information

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

Website: www.rocketsoftware.com

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

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

Country	Toll-free telephone number
United States	1-855-577-4323
Australia	1-800-823-405
Belgium	0800-266-65
Canada	1-855-577-4323
China	400-120-9242
France	08-05-08-05-62
Germany	0800-180-0882
Italy	800-878-295
Japan	0800-170-5464
Netherlands	0-800-022-2961
New Zealand	0800-003210
South Africa	0-800-980-818
United Kingdom	0800-520-0439

Contacting Technical Support

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

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

Contents

Notices ... 2

Corporate information ... 3

Chapter 1: External Database Access ... 7

First Normal Form (1NF) vs. Non-First Normal Form ... 7

Mapping UniData data to the 1NF data model ... 7

Table concepts ... 7

Representing multivalues and multi-subvalues ... 8

Primary and foreign keys ... 8

Associations ... 8

Virtual attributes and I-descriptors ... 9

Mapping example ... 9

Chapter 2: EDA Schema Manager ... 12

Starting the EDA Schema Manager ... 12

Creating a new U2 server connection ... 13

Connecting to the U2 server ... 15

Managing connections ... 15

About data sources ... 15

Connecting to SQL server, Oracle, or IBM DB2 ... 16

Connecting to Microsoft SQL server using the native client ... 17

Defining a data source ... 17

Creating EDA schemas ... 19

Defining attribute details ... 22

Defining options ... 24

Viewing EDA server details ... 25

Viewing U2 server details ... 26

EDA schema examples ... 27

External DATETIME field ... 27

Scalar function ... 28

TRANS function ... 29

Table function example ... 32

Using SQL Server to create EDA schema maps ... 35

Adding a FULLNAME virtual field to the CUSTOMER schema map ... 35

Adding an UPCASE.LNAME virtual field to the CUSTOMER schema map ... 36

Adding the UPCASE.LNAME virtual field to the CUSTOMER schema map ... 36

Adding the DESCRIPTION virtual field to the CUSTOMER schema map ... 36

Verifying EDA schemas ... 37

Verification example ... 39

Viewing the EDA schema ... 40

Viewing the DDL scripts ... 40

Converting data from U2 to an external database ... 41

Viewing EDA files ... 42

Accessing data in an external database ... 43

Listing data using UniQuery ... 43

Listing data using UniData/UniVerse SQL ... 44

Chapter 3: External database supplied drivers ... 45

EDA Oracle driver ... 45

Set up the EDA environment ... 45

Set up the Oracle connection file ... 46

Set up dynamic-loading library ... 46

Create the EDA data source ... 46

Oracle data type mapping....46
EDA DB2 driver....47
Set up the EDA environment....47
Install DB2 or the DB2 client....47
Set up connection to the DB2 database....47
Create the EDA data source....47
DB2 data type mapping....48
EDA SQL Server driver....48
Install SQL Server....48
Create the EDA data source....48
Set up the EDA configuration file....49
SQL Server data type mapping....49
EDA Common driver....49
MySQL data type mapping....50
PostgreSQL data type mapping....50
EDA Common driver for Windows....50
Install MySQL....50
Create the EDA data source....51
Set up the EDA configuration file....51
EDA Common driver for UNIX and Linux....51
Set up the EDA environment....51
Install unixODBC and third-party ODBC driver....52
Set up connection to external database....52
Set up the ODBC dynamic loading library path....54
Create the EDA data source....54
Automatic data type mapping....54
EDA driver log files....55
Chapter 4: External database access driver API....56
EDA driver API....56
Registering an EDA driver....56
EDA driver functions....57
EDADRV_LoadSymbols....57
EDADRV_Connect....59
EDADRV_Disconnect....59
EDADRV_EndTransaction....60
EDADRV_PrepareStmt....61
EDADRV_ExecuteStmt....62
EDADRV_CloseStmt....63
EDADRV_DropStmt....63
EDADRV_FetchStmt....64
EDADRV_Perform....65
EDADRV_GetEDAAttr....66
EDADRV_GetErrmsg....67
EDADRV_Cleanup....67
EDADRV_FreeResult....68
EDADRV_GetDBInfo....68
EDADRV_GetSpecialInfo....69
The EDA driver header file....70
Chapter 5: EDA ECL commands....78
ALTER.EDAMAP....78
EDA.CONNECT....78
EDA.CONVERT....79
EDA.DISCONNECT....80
EDA.EXCEPTION....80
EDA.LOG....80

EDA.VERSION....80
LIST.EDAMAP....81
SAVE.EDAMAP....82
SELECT.EDA.NONCONFORMING....82
VERIFY.EDAMAP....83

Chapter 6: EDA exception handling....84
The EDA_EXCEPTION file....84
INMAT....86

Chapter 7: EDA Replication....87
Setting up a server....87
Defining a data source....88
Defining EDA Replication parameters....88
Configuring replication parameters....89
Configuring the replication system....91
Choosing files to replicate....94
Synchronizing replication files....97
Creating EDA schemas for replicated files....99
Creating default EDA schemas....99
Creating EDA schemas....100
Converting replicated files to EDA files....101

Chapter 8: EDA best practices....103
Map selected fields....103
Avoid multiple multivalued associations....103
Avoid restrictive data types....103
Data types for record IDs....103
RECORD_BLOB....104
Updating an EDA file from the external database....104

Chapter 1: External Database Access

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

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

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

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

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

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

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

Mapping UniData data to the 1NF data model

If you have used UniData ODBC, JDBC or UniOLEDB, you are already familiar with the UniData tools that translate UniData nested relational data to adhere to ODBC/SQL rules, such as the Visual Schema Generator (VSG) or the Schema API. The 1NF mapping concepts used in EDA are the same as those used in VSG and the Schema API. The difference between VSG and the Schema API, on one hand, and EDA on the other, is that VSG and the Schema API leave the data in UniData files and only translate the data for access by such 1NF-based tools as MS Visual Basic, MS Access, Crystal Reports and WebSphere. The EDA Schema Manager maps and transfers UniData data into external database tables, and then allows the data to be accessed by both UniData applications and tools, and the external database applications and tools.

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 and multi-subvalues

To represent the three nested levels of data within UniData files or UniData SQL tables (singlevalued, multivalued and multi-subvalued), the EDA Schema Manager creates three 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.
- Multi-subvalued attributes (MS) – a table containing multi-subvalued attributes of the association.

These tables are “linked” through primary and foreign keys.

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

Primary and foreign keys

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

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

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

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

Associations

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

Following is an example of related information that would be stored in a U2 database as an association: The students in a school each taking several courses within a semester, the name and number of the courses, the grade the student received in the courses, the number of credit hours for

the courses, and the name of the teacher for the course. You do not want the grade for one course getting mistaken for that of another, and you want the correct course names related to the correct course number.

For each association, the EDA Schema Manager creates one multivalued attributes table, and if there are mapped multi-subvalued attributes, one multi-subvalued attributes table. If the U2 file contains more than one association, the EDA Schema Manager creates separate multivalued (MV) and multi-subvalued attribute (MS) tables for each association. One singlevalued attribute (S) table can be the parent of many multivalued attribute (MV) tables.

Virtual attributes and I-descriptors

You can map virtual attributes or V-type attributes (UniData) or I-descriptors (UniVerse) to an external database.

There are three types of mapping:

- Simple – a simple virtual attribute/I-descriptor, such as A + B. These are virtual attribute formulas that are translated to expressions and SCALAR functions.
- TRANS – a virtual attribute/I-descriptor that performs a TRANS operation. These are virtual attributes or I-descriptors that are mapped using TRANS or TABLE function type of mapping. TABLE function mapping is used for multiple TRANS operations.
- Materialized Virtual/I-descriptor – a virtual attribute/I-descriptor that is evaluated in U2, with the result stored in DB2. If you are mapping this type of virtual attribute/I-descriptor, select DATA as the type of mapping.

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

For more information about these types of attributes, see Defining attribute details, on page 22.

Mapping example

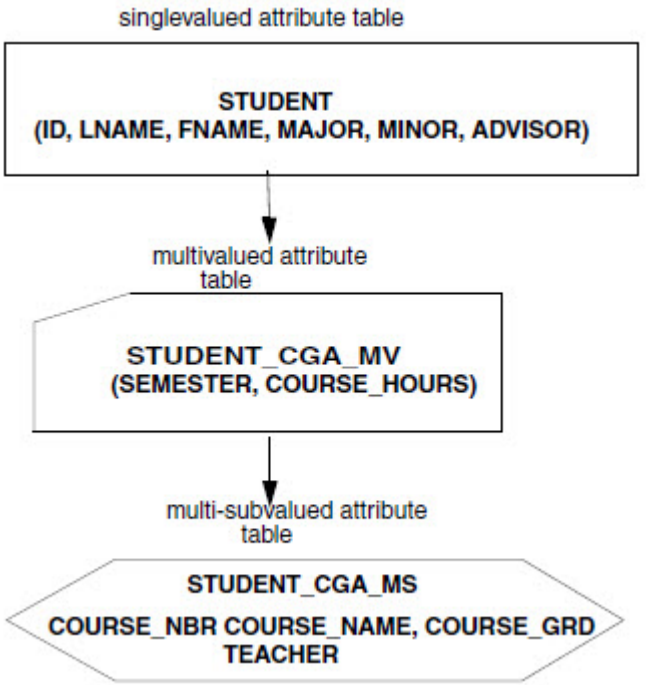
Consider the following dictionary from the UniData demo database `STUDENT` file:

```
LIST DICT STUDENT BY TYP BY @ID TYP LOC CONV NAME FORMAT SM ASSOC 11:58:17
Dec 1
2 2006 1
@ID............ TYP LOC.......... CONV NAME........... FORMAT SM ASSOC.....

@ID               D              0       STUDENT      12R###       S
                                                         -##-##
                                                          ##
ADVISOR           D              5       Advisor        8L         S
COURSE_GRD        D              8       GD             3L         MS CGA
COURSE_NBR        D              7       Crs #          5L         MS CGA
FNAME             D              2       First Name     10L        S
ID                D              0       STUDENT        12R###     S
                                                           -##-##
                                                              ##
```

```
LNAME           D              1       Last Name       15T      S
MAJOR           D              3       Major           4L       S
MINOR           D              4       Minor           4L       S
SEMESTER        D              6       Term            4L    MV CGA
COURSE_HOURS    I   TRANS('COURSE      Hours           5R    MS CGA
                    S',COURSE_NBR
                    ,CREDITS,'X')
COURSE_NAME     I   TRANS('COURSE      Course Name     25L   MS CGA
                    S',COURSE_NBR
                    ,'NAME','X')
CGA             PH  SEMESTER COUR
                     SE_NBR COURSE
                    _NAME COURSE_
                    GRD COURSE_HO
                     URS TEACHER
@ORIGINAL       SQ           @ID
@SYNONYM        SQ            ID
GPA1            V    SUBR('GPA1',C      MD3        GPA    5R      S
                     OURSE_HOURS,C
                     OURSE_GRD)
TEACHER         V   TRANS('COURSE           Teacher       10L    MS    CGA
                     S',COURSE_NBR
                    ,'TEACHER','X
                     ')
```

The following figure illustrates how the EDA Schema Manager creates tables for all attributes defined in this dictionary. One primary table is the parent of a multivalued table and the multivalued attribute table is the parent of a multi-subvalued table.



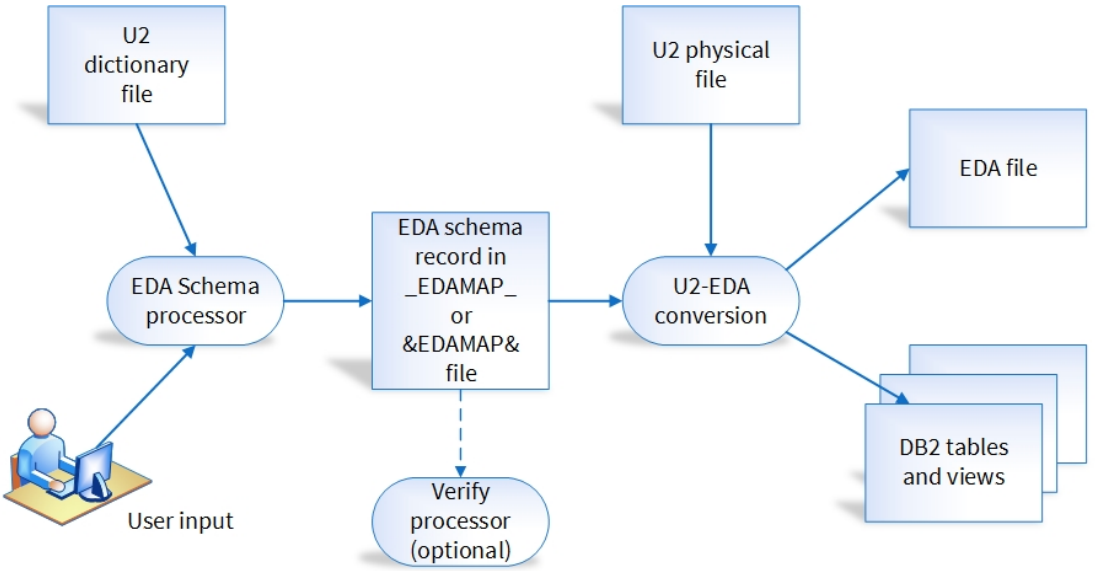
The figure shows table names assigned by The EDA Schema Manager on the external database.

- `STUDENT` – singlevalued attributes (S) table. Also called the primary table in this document.

- `STUDENT_CGA_MV` – a multivalued attributes (MV) table based on the association `CGA`.
- STUDENT_CGA_MS – a multi-subvalued attributes (MS) subtable based on the association CGA.

Chapter 2: EDA Schema Manager

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



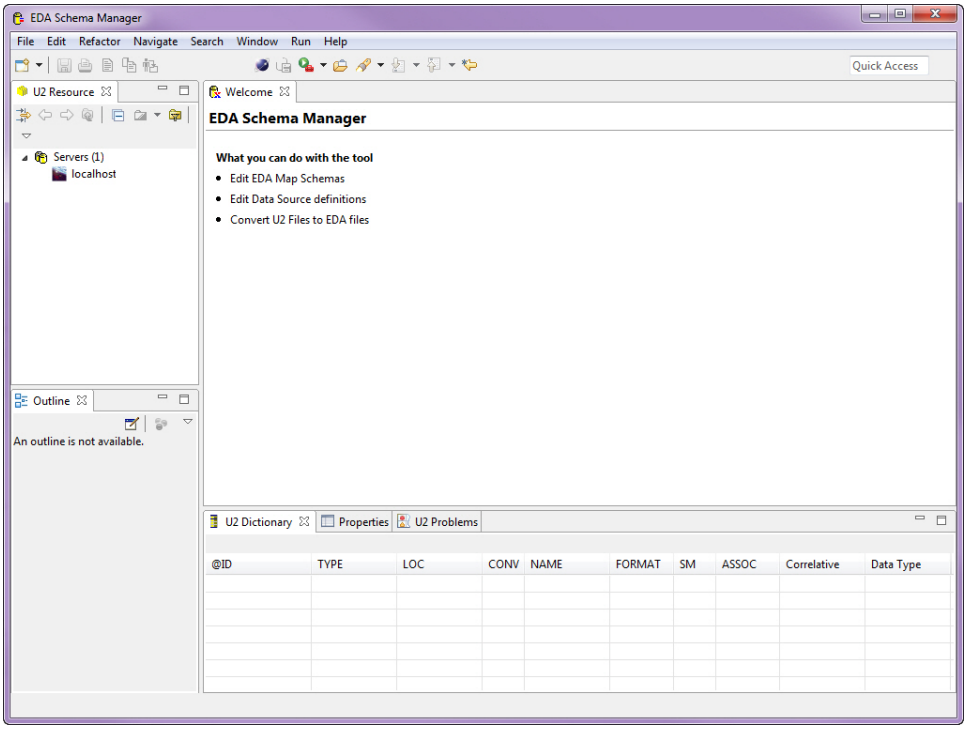
To convert U2 files to an external database format, EDA Schema Manager performs the following steps:

1. The EDA Schema processor receives information from the dictionary file for the data file you are converting and other user input.
2. From this information, the EDA Schema Processor creates an EDA schema. This EDA schema is a record in the `_EDAMAP_` file.
3. Optionally, you can verify the EDA schema.
4. The conversion process uses the EDA schema record and the U2 physical file to create tables and views in the external database.
5. The U2 physical file is replaced by an EDA file in the U2 account. The original data file is saved as *`filename`*`.edasave`.

Starting the EDA Schema Manager

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

From the Windows Start menu, click **Programs** → **Rocket U2** → **EDA Schema Manager**.



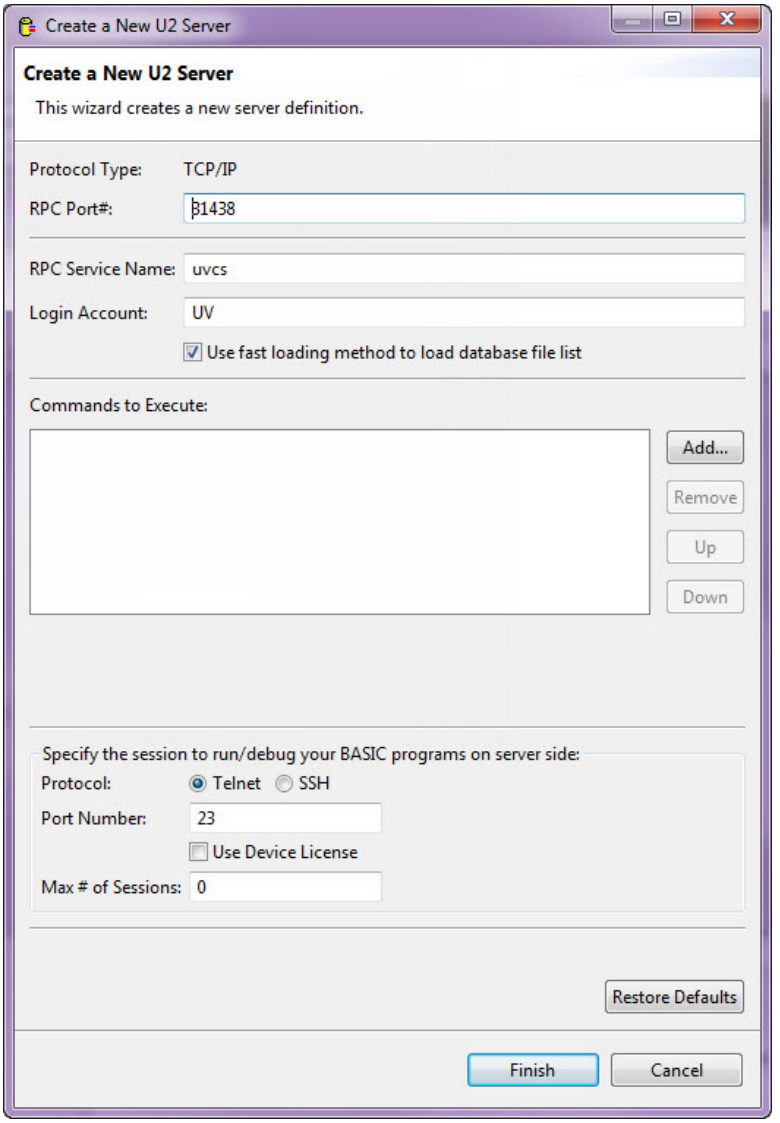
Creating a new U2 server connection

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

Procedure

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

   A dialog box similar to the following example appears:



a. In the **Protocol Type** field, select the type of communication you are using for the server. You can select **Default**, **TCP/IP**, or **Lan Manager**. The default is **TCP/IP**.
b. In the **RPC Port #** field, enter the port number of the UniRPC server running on the host. The default port number is 31438.
c. In the **RPC Service Name** field, enter the name of the RPC service on your system. For UniData, this is normally *udcs*. For UniVerse, this is normally *uvcs*.
d. In the **Login Account** field, enter the name of the account to which you want to log on when accessing the U2 database.
e. In the **Commands to Execute** field, click **Add** to enter commands that you want to execute when you log on to the server. Enter commands in the dialog box that appears, then click **OK** when finished.
f. In the **Specify the session to run/debug your BASIC programs on server side** area, specify the type of connection that you want to make to the server. You can specify **Telnet** or **SSH**.
g. In the **Port Number** field, enter the port number you want to use if you do not want to use the default port number of 23.
h. Select the **Use Device License** check box if you want to enable device licensing when connecting to the server.

i. In the **Max # of Sessions** field, enter a number.

6. Click **Finish**.

   The new server appears in the U2 Resource view.

Connecting to the U2 server

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

1. Right-click the server name, then click **Connect**.

   When you connect to the server, the Connect to U2 Server dialog box appears.

2. In the **User ID** field, enter the User ID for the machine where U2 is running.

   Note: Beginning at UniData 8.1.0, the udadm role has been added to UniData client/server processes (UO, UOJ, etc.), which allows non-root users to update data sources in the EDA Schema Manager. The udadm user or members of udadm group have special privileges that allow them to access UniData when they run BASIC programs. Beginning at this release, udadm or members of udadm group can successfully update EDA data sources.

3. Enter the corresponding password in the **Password** field.

   To store the password for future connections, select the Remember me check box. With this check box selected, Microsoft Windows stores the encrypted password on the client computer.

4. If you are using a proxy server, select the **Use Proxy Server** check box.
   a. In the **Proxy Host** field, enter the name or IP address of the computer on which the proxy server is running.
   b. In the **Proxy Port** field, enter the number of the port on which the proxy server listens for communication from UniData.

5. Click **Finish**.

   The accounts and existing data sources definitions appear in the U2 Resource view.

Managing connections

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

About data sources

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

- IBM DB2
- Microsoft SQL Server
- Oracle Database

Note: For more information regarding the external database that you are accessing, see .

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

Connecting to SQL server, Oracle, or IBM DB2

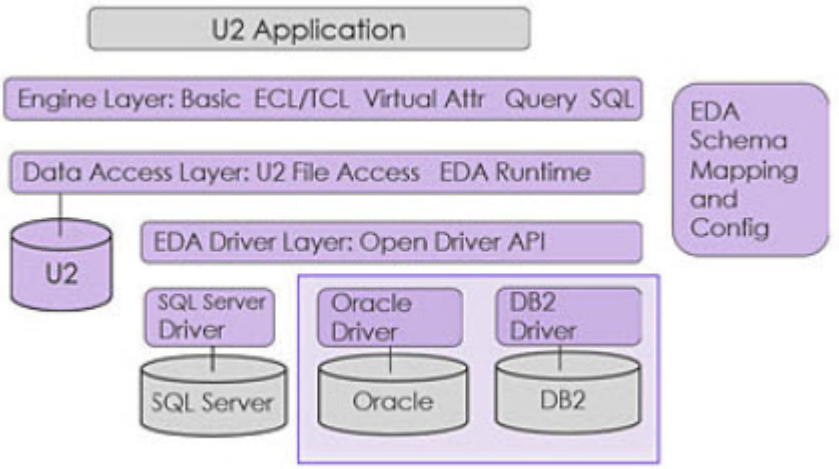
The U2 server can reside on UNIX, Linux, or Windows. After the SQL server, Oracle, or DB2 database server is installed, the appropriate ODBC driver must be installed on the U2 server machine. The drivers for access to the databases are:

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

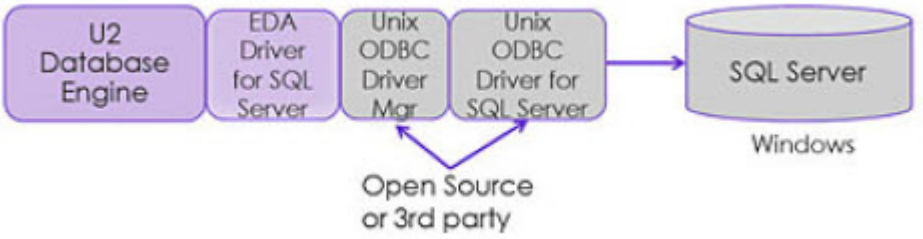
The installation program automatically places the EDA driver library for the following in the `$UDTBIN` (UniData) or `$UVBIN` (UniVerse) directory:

- SQL server (`libcomdrv`)
- EDA driver library for Oracle (`liboradrv`)
- EDA driver library for DB2 (`libdb2drv`)

Note: Microsoft provides SQL server ODBC drivers on RedHat and SUSE Linux platforms.



The following examples show how U2 connects to database servers:



Connecting to Microsoft SQL server using the native client

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

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

Defining a data source

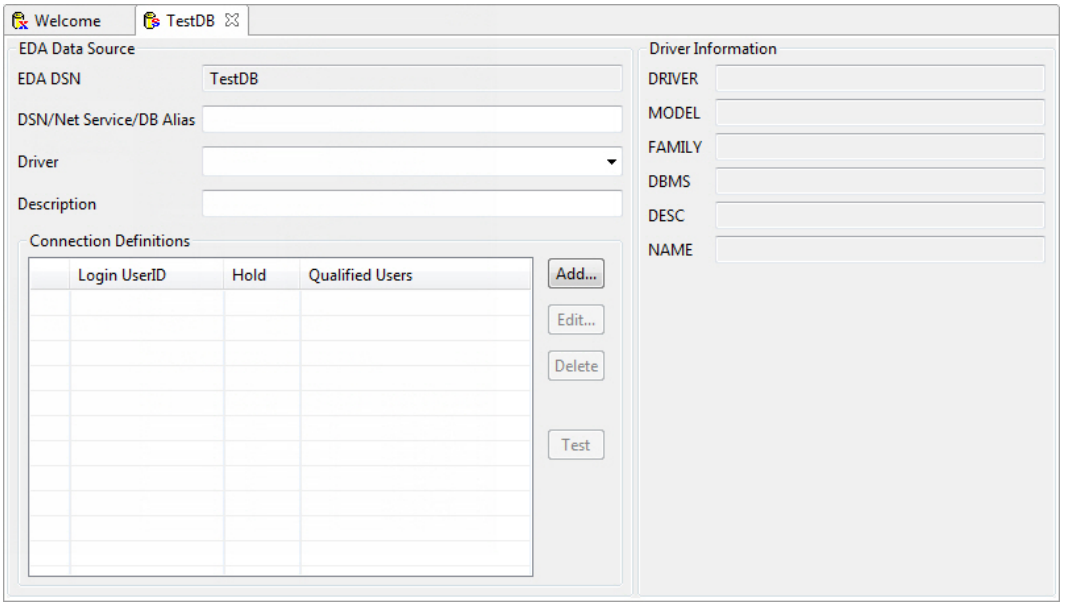
You must define a data source pointing to the external database to which you want to connect.

Procedure

1. From the U2 Resource view, expand the server you want to use, right-click **EDA Data Sources**, then click **New** → **EDA Data Source**.

2. In the Create a New EDA Data Source wizard, enter a unique name for the external data source in the **Data Source Name** field, then click **Finish**.

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



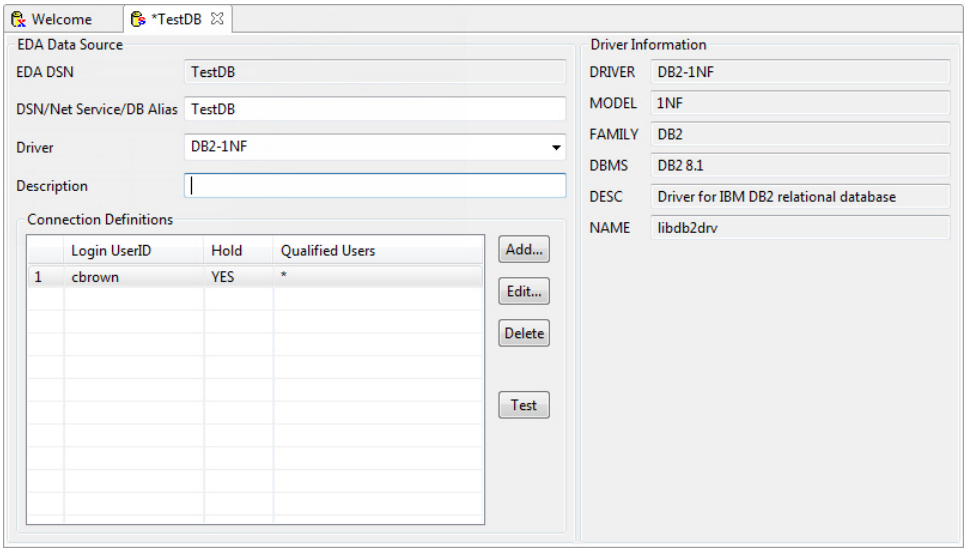
3. In the **External DSN** field, enter the name of the external database client that provides the connection to the desired external database instance. For Microsoft SQL Server, it is the name of the ODBC Data Source you have defined in ODBC Data Source Administration. For DB2, it is the database name specified in the `CATALOG DATABASE` command. For Oracle, it is the connection name defined in the `tnsnames.ora` file.
4. In the **Driver** field, enter the type of driver and enter any details about it in the **Description** field.
5. Click **Add**. In the EDA Data Source Connection dialog box that displays, enter the following information:
   a. In the **Login User ID** field, enter the user ID on the external server.
   b. In the **Password** field, enter the password corresponding to the user ID. Enter the password again in the **Re-enter Password** field.
   c. If you want to maintain the connection to the external server after a transaction commits, select **YES** from the **Hold Flag** drop-down menu. If you want to disconnect from the external server after the transaction commits, select **NO**.

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

   d. In the **Qualified Users** field, enter the U2 user IDs of users who can access the external server from the U2 account using the external Login User ID you specify. Separate the users by a vertical bar (|) character. If all U2 users can access the external account, enter an asterisk (*).
   e. Click **Finish**.

Results

The following example shows a completed EDA Data Source:



To test the connection to the external instance, click **Test**. A success message displays if the connection is successful; otherwise a detailed error message displays.

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

Creating EDA schemas

Prerequisites

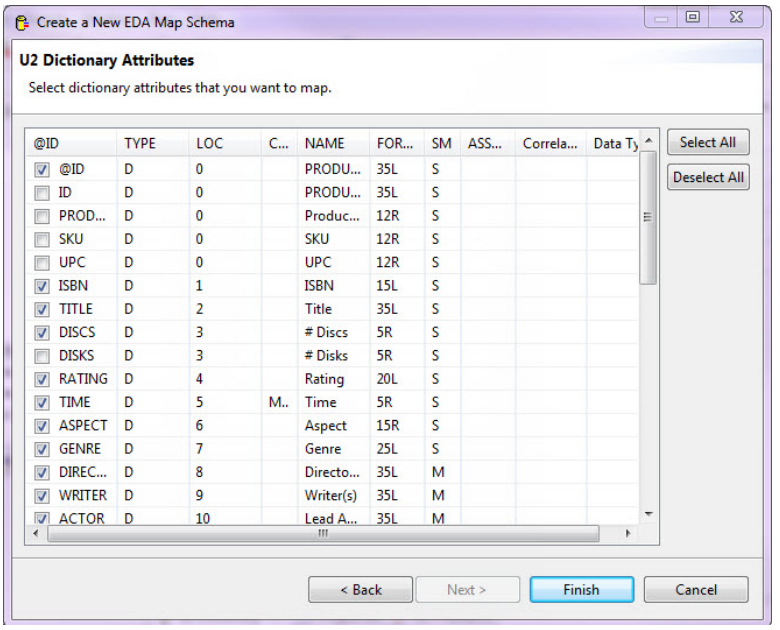
- Defining a data source, on page 17

Procedure

1. From the U2 Resource view, expand **Accounts** and then expand the account where the files you want to convert reside.
2. Right-click the **EDA Schema Files**, and select **New** → **EDA Map Schema**.
3. In the Create New EDA Map Schema wizard that displays, enter a unique name for the EDA schema in the **EDA Schema Name** field.
4. Select **EDA Schema** from the **Map Format** options, and click **Next**.
5. On the Source U2 file page, click the file for which you are creating a schema.

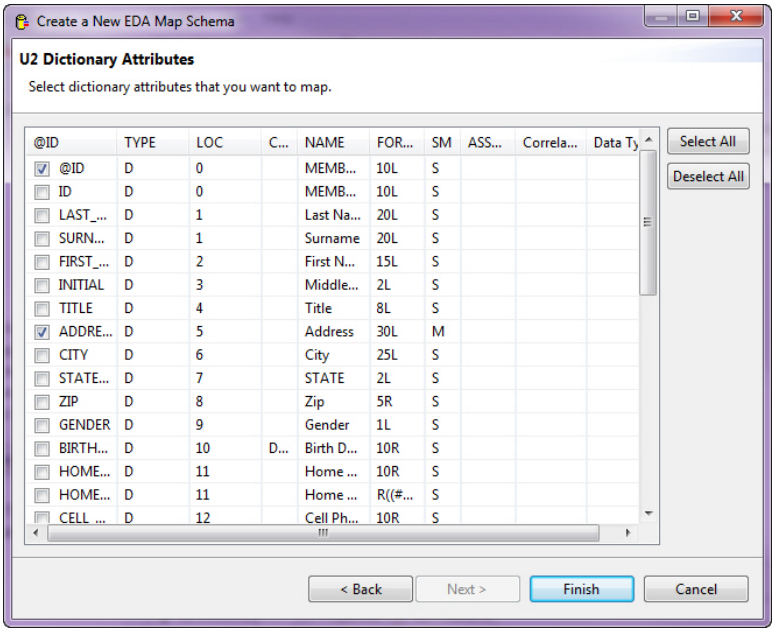
6. From the **Using Data Source** drop-down menu, select the data source you are using. Click **Next**.

   The U2 Dictionary Attributes page displays, as shown in the following example:



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

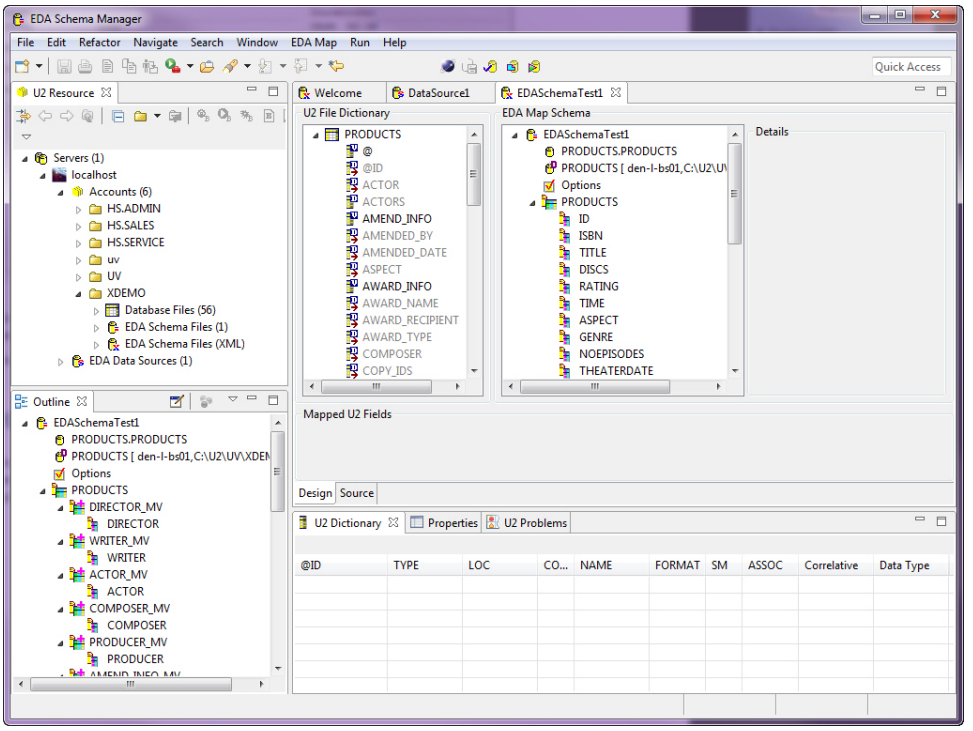
   If you want to selectively map U2 attributes to an external database, only select those attributes you want to map. If you do not select any dictionary attributes, the @ID attribute is automatically mapped. In the following example, only **@ID** and **ADDRESS** have been selected:



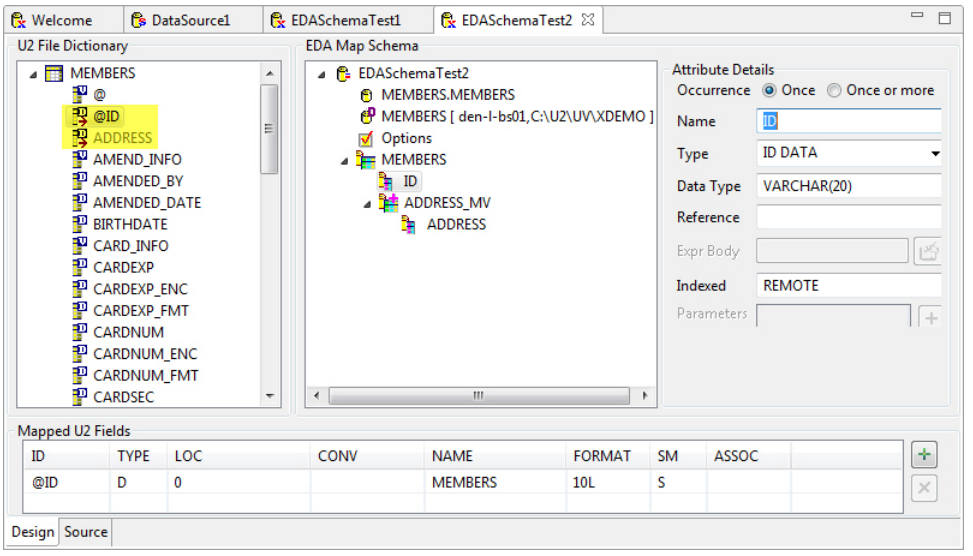
7. Click **Finish** when you have selected all the dictionary attributes for which you want to create schemas.

Results

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



The following example illustrates the appearance of the window after the **@ID** and **ADDRESS** attributes have been selected. Notice that a red arrow displays next to the attribute in the U2 File Dictionary portion of the window, indicating the attribute has been mapped. Click the mapped attribute you want to display in the EDA Map Schema view.



Note: The EDA Schema Manager allows 30-character column names for Oracle and DB2 and 60-character column names for SQL Server. If the dictionary ID length is longer, it will be truncated in the EDA Map Schema portion of the window.

Next step

- Defining attribute details, on page 22
- Defining options, on page 24

Defining attribute details

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

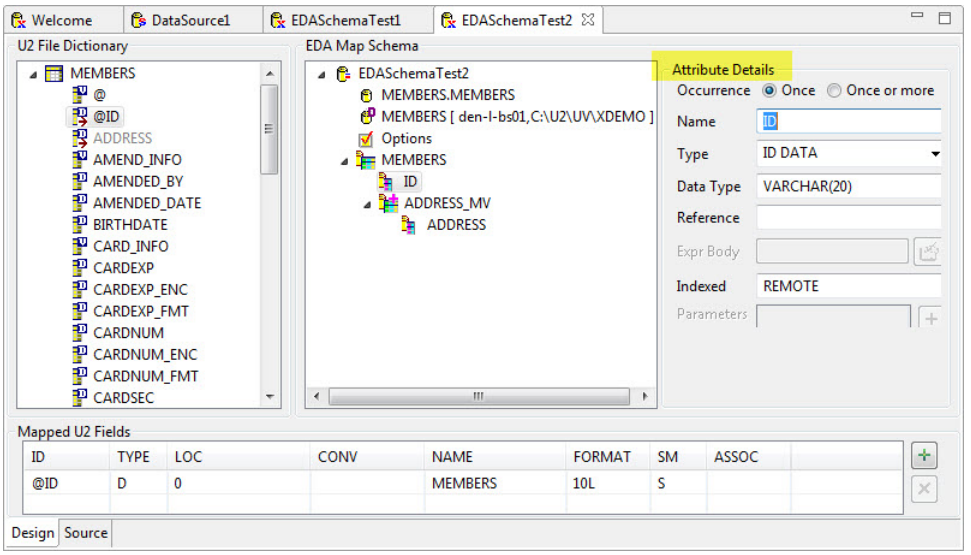
Prerequisites

- Creating EDA schemas, on page 19

About this task

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

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



Procedure

1. Select an **Occurrence**: **Once** or **Once or More**.
2. In the **Name** field, enter the name of the column in the resulting external table.
3. In the **Type** field, select the type of attribute. The following values are valid:

Option	Description
DATA	Used to store attribute values allowed by the data type you specify. This option creates a column in the external database. If this is a D-type attribute, its values are stored in this column. If it is defined as a virtual attribute or V-type (UniData) or I-descriptor (UniVerse), the virtual attribute/I-descriptor is evaluated in U2, and the result is stored in this column in the external database.
EXPRESSION	Used for virtual attributes (UniData) or I-descriptors (UniVerse) only. Enter the SQL expression for the virtual/I-descriptor attribute in the **Expr Body** field, such as FNAME CONCAT ' ' CONCAT LNAME.
ID DATA	The primary key in the external table.
NOT NULL DATA	Specifies that the external database column cannot contain the null value.
SCALAR FUNCTION	Used to define a scalar function to execute an equivalent virtual attribute/I-descriptor on the external database. For information about creating a scalar function, see Scalar function, on page 28.
TABLE FUNCTION	Used to define a table function For information about creating a table function, see Table function example, on page 32.
TRANS	Used for virtual attributes/ I-descriptors containing a TRANS clause only. If you specify TRANS, you must also specify **Reference** and **Parameters**. For more information, see TRANS function, on page 29.
UNIQUE DATA	Specifies that values in the external database column must be unique.

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

   The EDA Schema Manager automatically converts the data type based on the dictionary record. If an application attempts to insert or update an external attribute with a value that does not match the data type you define, the EDA system rejects the operation.

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

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

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

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

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

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

Results

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

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

Defining options

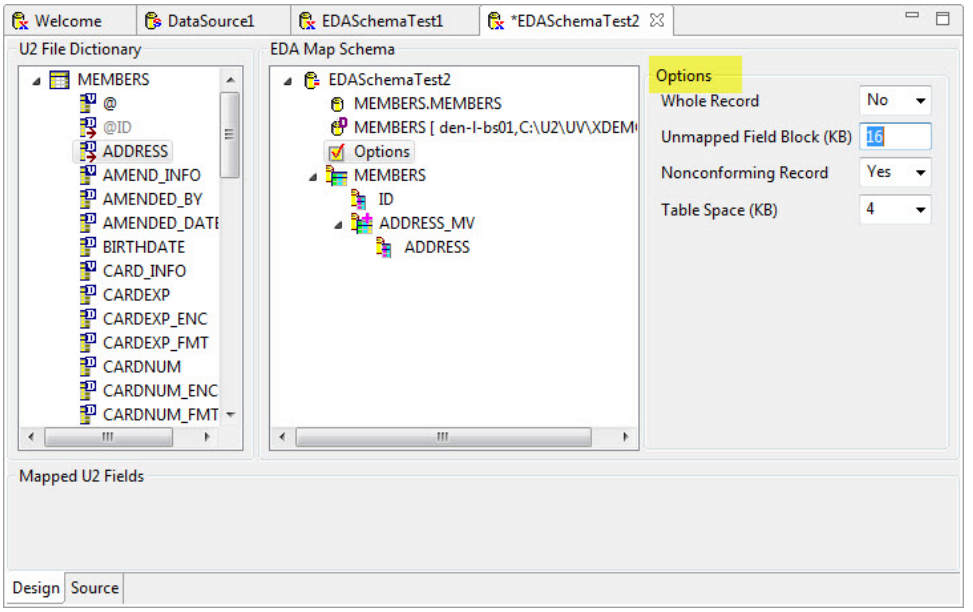
You can view the options from a mapped EDA Schema.

Prerequisites

- Creating EDA schemas, on page 19

Procedure

1. Click **Options** in the EDA Map Schema area of the U2 EDA Schema Manager.
   The Options area is to the right of the EDA Map Schema area.



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

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

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

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

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

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

   The RECORD_BLOB serves several purposes:

   - To hold all attributes that are not explicitly mapped
   - To hold the entire U2 record when the WHOLE_RECORD flag is set to **Yes**
   - To hold nonconforming records

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

4. From the **Nonconforming Record** drop-down menu, select **Yes** or **No**.

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

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

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

   For more information about retrieving nonconforming data, see .

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

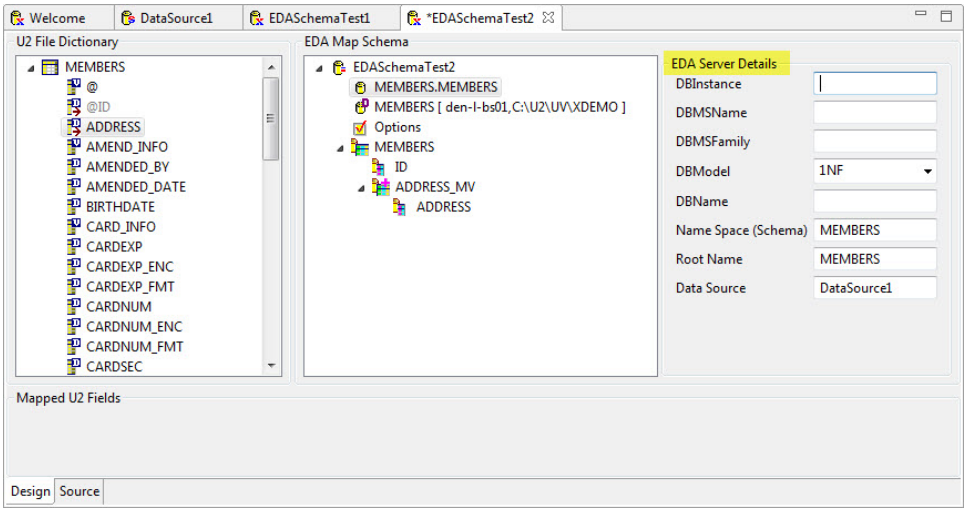
   A table space is the basic storage structure on an external database. By default, U2 creates all tables in USERSPACE 1, the default user table space. This table space has a 4 KB page size, so the length of a row of a table is limited to less than 4 KB. If the row length is exceeded, U2 generates an error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

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

Viewing EDA server details

To view information about the EDA server, click the external Schema Name/Table Name in the **EDA Map Schema** pane of the U2 EDA Schema Manager.

Information about the EDA server is displayed, as shown in the following example:



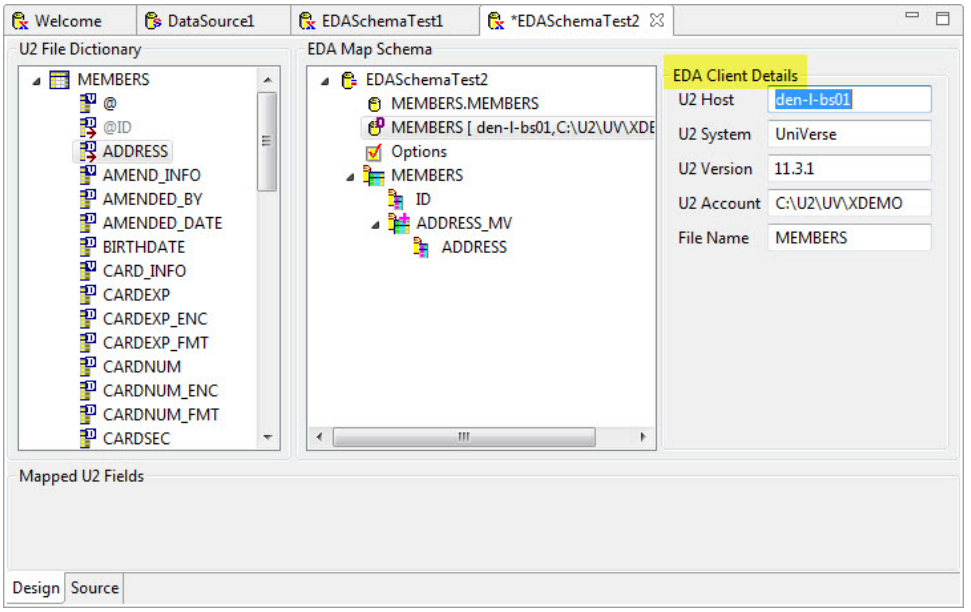
The U2 EDA Schema Manager displays the following details about the EDA server:

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

Viewing U2 server details

To view information about the U2 server, click the U2 file name in the **EDA Map Schema** area of the EDA Schema Manager.

Information about the U2 server appears, as shown in the following example:



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

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

EDA schema examples

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

- External DATETIME field
  This example shows how to map **Date** and **Time** fields to a **DATETIME** field in an external database.
- Scalar function
  This example illustrates a scalar function, used to execute an equivalent virtual attribute/I-descriptor on the external database.
- TRANS function
  This example illustrates the `TRANS` function, used for virtual attributes/I-descriptors containing a TRANS clause.
- Table function example
  U2 allows you to use the DB2 table function concept to evaluate multiple virtual attributes/I-descriptors at the same time. In some cases, you are able to map more than one virtual attribute/I-descriptor with one DB2 user-defined table function.

External DATETIME field

This example shows how to map **Date** and **Time** fields to a **DATETIME** field in an external database.

About this task

You can create a virtual field to map the **Date** and **Time** fields to a **DATETIME** field in an external database. The virtual field contains the content of both the **Date** field and the **Time** field, which is then mapped to an external database.

Procedure

1. Create the virtual field **T_DATETIME** in U2 DICT file:

```
T_DATE D 1 D4/ Date 10R S
T_TIME D 2 MTH Time 7R S
T_DATETIME I OCONV(T_DATE, 'D-YMD'):' ':OCONV(T_TIME, 'MTS') DateTime 20R S
```

2. In the EDA Map Schema, complete the following steps:
   a. Drag the **T_DATETIME** field to the EDA map schema.
   b. Change the data type of the **T_DATETIME** field as follows:

   - For Oracle, change the data type to TIMESTAMP.
   - For DB2, change the data type to TIMESTAMP.
   - For SQL, change the data type to DATETIME.
   - For MySQL, change the data type to DATETIME.
   - For Postgres, change the data type to TIMESTAMP.

Parent topic: EDA schema examples

Scalar function

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

About this task

Assume you have the following virtual attribute defined for the STUDENT file in your UniData database:

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

This virtual attribute converts the student's last name to uppercase.

Procedure

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

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

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

```
SYSFUN.UCASE(VARCHAR(30))
```

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

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

```
:LIST STUDENT UPCASE.LNAME
LIST STUDENT UPCASE.LNAME 10:57:37 Mar 14 2012 1
STUDENT... ..........................
424325656 MARTIN
414446545 OFFENBACH
978766676 MULLER
221345665 MILLER
291222021 SMITH
5 records listed
```

Parent topic: EDA schema examples

TRANS function

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

About this task

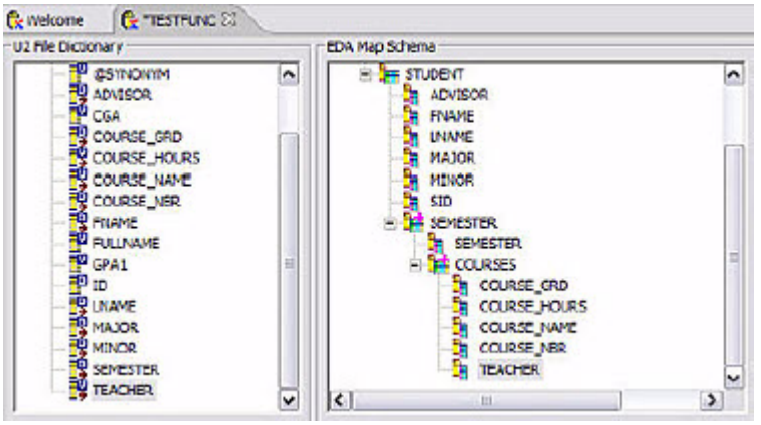
Assume you have the following U2 virtual attribute/I-descriptor defined in the dictionary of the STUDENT file:

```
001: V
002: TRANS('COURSES',COURSE_NBR,'TEACHER','X')
003:
004: Teacher
005: 10L
006: MS
007: CGA
```

This virtual attribute executes a translate from the STUDENT file to the COURSES file and returns TEACHER.

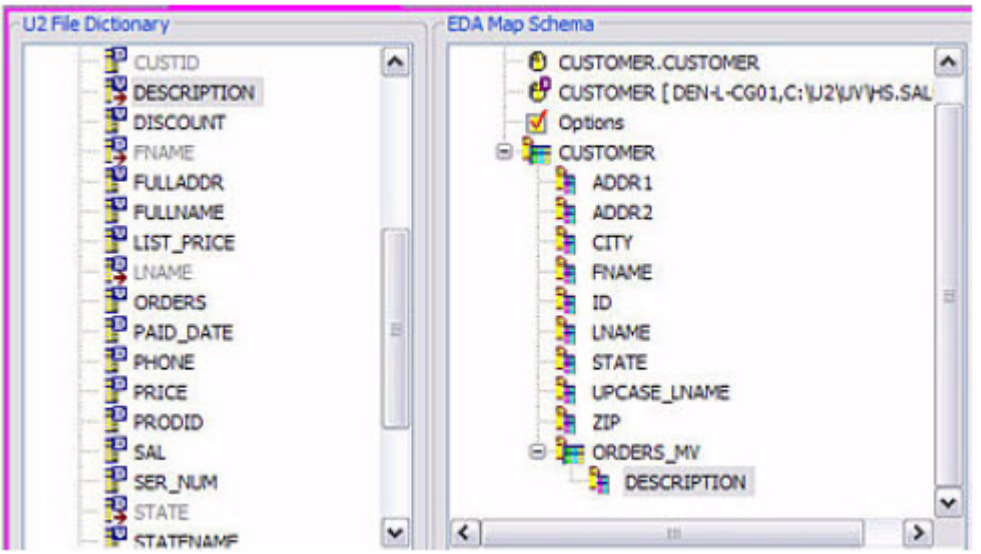
Procedure

1. To convert this virtual attribute to a `TRANS` function using the EDA Tool, drag TEACHER from the U2 File Dictionary pane under the COURSES node of STUDENT in the EDA Map Schema pane, as shown in the following example:

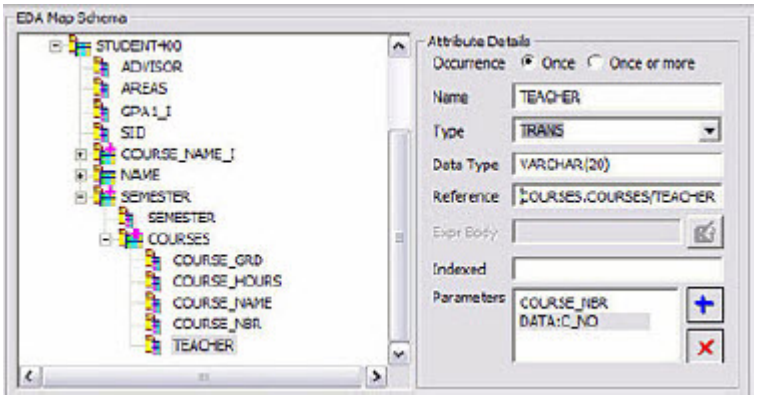


This I-descriptor executes a translate from the CUSTOMER file to the PRODUCTS file and returns DESCRIPTION.

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



3. Click TEACHER 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 `TEACHER` function:



4. In the **Attribute Details** pane, change the **Type** to **TRANS**.
5. In the **Data Type** field, define the data type for the output.

   Note: A **Data Type** is not required if **DATA** is selected as the **Type**.

6. In the **Reference** field, enter the name of the external table that contains the `TEACHER` information.

   In this example, the information resides in COURSES.COURSES/TEACHER.

   Note: A **Reference** is not required if **DATA** is selected as the **Type**.

7. In the **Parameters** field, click the plus sign (+) and enter the field to pass to the `TRANS` function.

   In this example, COURSE_NBR is passed to the `TRANS` function. The following example illustrates output from the TEACHER scalar function:

```
LIST STUDENT SEMESTER COURSE_NBR TEACHER 15:21:17 Mar 14 2012 1
STUDENT... Term Crs # Teacher...

221345665  FA93 EG110 Carnes
           MA220 Otis
           PY100 Masters
      SP94 EG140 Aaron
           EG240 Carnes
           MA221 Otis
291222021  SP94 FA100 Fried
414446545  FA93 CS104 Aaron
           MA101 Otis
           FA100 Fried
      SP94 CS105 Gibson
           MA102 Otis
           PY100 Masters
424325656  SP94 PY100 Masters
           PE100 Fisher
978766676  FA93 FA120 Fried
           FA230 Carnes
           HY101 Otis
      SP94 FA121 Carnes
           FA231 Fried
```

Parent topic: EDA schema examples

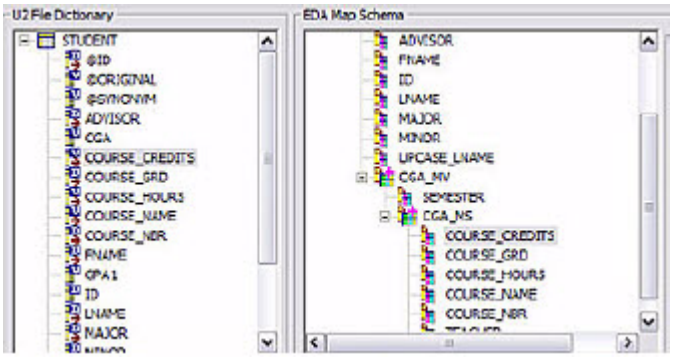
Table function example

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

Assume you create a monthly report containing the student names, semesters, course names, and credit hours. To create this report from UniData, you use the following virtual attributes for course name and credit hours:

```
:AE DICT STUDENT COURSE_NAME
001: I
002: TRANS('COURSES',COURSE_NBR,'NAME','X')
003:
004: Course Name
005: 25L
006: MS
007: CGA
:AE DICT STUDENT COURSE_CREDITS
001: V
002: TRANS('COURSES',COURSE_NBR,'CREDITS','X')
003:
004: Credits
005: 5R
006: MS
007: CGA
```

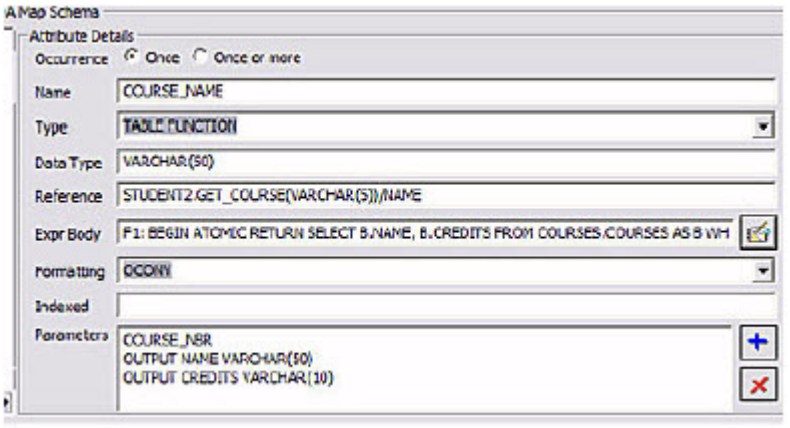
To map these virtual attributes, drag each one from the **U2 File Dictionary** pane to the CGA_MS node in the **EDA Map Schema** pane, as shown in the following example:



Although you map COURSE_NAME and COURSE_CREDITS separately, DB2 allows you to create one function for use with multiple attributes. You have to define the EDA Map Schema for both virtual attributes, 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 COURSE_NAME and COURSE_CREDITS on the DB2 database.

First, let's map the COURSE_NAME virtual attribute:



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

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

In the **Reference** field, enter the DB2 Schema name, the name of user-defined function you are defining in the **Expr Body** field, the data type for the input value, and the DB2 function attribute name, as shown in the following example:

```
STUDENT2.GET_COURSE(VARCHAR(5))/NAME
```

In this example, the user-defined function will be named GET_COURSE and reside in the STUDENT2 schema in the DB2 database. The data type of the input parameter is VARCHAR(5), and you are using the output parameter NAME.

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

```
F1:BEGIN ATOMIC RETURN SELECT B.NAME, B.CREDITS FROM COURSES.COURSES AS B
WHERE B.ID=COURSE_NBR; END
```

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

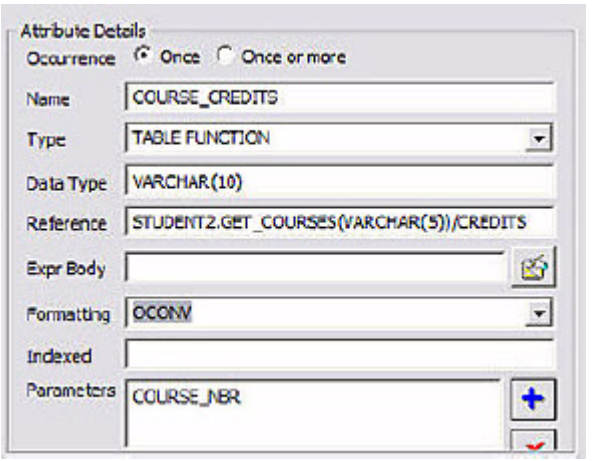
```
OUTPUT NAME VARCHAR(50)
OUTPUT CREDITS VARCHAR(10)
```

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

```
CREATE FUNCTION STUDENT3,GET_COURSE(COURSE_NBR VARCHAR(5)) RETURNS TABLE(
NAME VARCHAR(50),CREDITS VARCHAR(10))
F1:BEGIN ATOMIC RETURN SELECT B.NAME,B.CREDITS FROM COURSES.COURSES AS B
WHERE B.ID=COURSE_NBR; END
```

Next, map the COURSE_CREDITS virtual attribute.

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



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

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

In the **Reference** field, enter the external Schema name, the name of user-defined function you previously defined in the COURSE_NAME table function (GET_COURSE), the data type for the input value, and the external database function attribute name, as shown in the following example:

```
STUDENT2.GET_COURSE(VARCHAR(5))/CREDITS
```

In this example, the user-defined function GET_COURSE resides in the STUDENT2 schema in the DB2 database. The data type of the input parameter is VARCHAR(5), and you are using the output parameter CREDITS.

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

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

You can now create the monthly report listing the student names, semesters, course names, and credit hours. Since you defined COURSE_NAME and COURSE_CREDITS using the same DB2 table function, DB2 only needs to evaluate the function once, improving performance.

```
LIST STUDENT NAME SEMESTER COURSE_NAME COURSE_CREDITS 14:40:18 M
STUDENT... NAME..... Term Course Name..... Hours

221345665  BW             FA93 Engineering Principles       5
                               Calculus- I                   5
                               Introduction to Psycholog     3
                               y
                          SP94 Fluid Mechanics               3
                               Circut Theory                 3
                               Calculus - II                 5
291222021  FA             SP94 Visual Thinking              3
414446545  PY             FA93 Database Design              3
                               Math Principals               3
                               Visual Thinking               3
                          SP94 Database Design              3
                               Algebra                       3
                               Introduction to Psycholog     3
                               y
424325656  EG             SP94 Introduction to Psycholog    3
                               y
                               Golf - I                      3
978766676  PY             FA93 Finger Painting              5
                               Photography Principals        3
```

Parent topic: EDA schema examples

Using SQL Server to create EDA schema maps

The examples in this section detail how to create the schema maps for Microsoft SQL Server. When creating a new EDA Map Schema, the EDA tool only maps the D-type fields, not virtual fields. The primary key field is set to ID DATA type and the other fields are set to DATA type.

Adding a FULLNAME virtual field to the CUSTOMER schema map

The FULLNAME virtual field will concatenate the first name with the last name. The type field can be defined as DATA or EXPRESSION.

When it is defined as DATA type, the FULLNAME field will be physically created and part of the CUSTOMER.CUSTOMER table.

The following SQL syntax creates the CUSTOMER.CUSTOMER table:

```
CREATE TABLE CUSTOMER.CUSTOMER(FULLNAME VARCHAR(120), ID
VARCHAR(20) NOT NULL, SAL VARCHAR(10), FNAME VARCHAR(24), LNAME
VARCHAR(32), COMPANY VARCHAR(40), ADDR1 VARCHAR(60), ADDR2
VARCHAR(60), CITY VARCHAR(24), STATE VARCHAR(4), ZIP VARCHAR(60),
PHONE VARCHAR(60), NONCONFORMING_FLAG SMALLINT, UNMAPPED_U2FIELD
VARCHAR(MAX), PRIMARY KEY (ID))
```

When it is defined as an EXPRESSION type, the FULLNAME field will be part of the CUSTOMER.CUSTOMER_V view.

The following SQL syntax creates the CUSTOMER.CUSTOMER_V view:

```
CREATE VIEW CUSTOMER.CUSTOMER_V(FULLNAME, ID, SAL, FNAME,
LNAME, COMPANY, ADDR1, ADDR2, CITY, STATE, ZIP, PHONE,
NONCONFORMING_FLAG, UNMAPPED_U2FIELD)
AS SELECT FNAME+' '+LNAME, CUSTOMER.ID, CUSTOMER.SAL,
CUSTOMER.FNAME, CUSTOMER.LNAME, CUSTOMER.COMPANY, CUSTOMER.ADDR1,
CUSTOMER.ADDR2, CUSTOMER.CITY, CUSTOMER.STATE, CUSTOMER.ZIP,
```

```
CUSTOMER.PHONE, CUSTOMER.NONCONFORMING_FLAG,
CUSTOMER.UNMAPPED_U2FIELD FROM CUSTOMER.CUSTOMER CUSTOMER
```

Adding an UPCASE.LNAME virtual field to the CUSTOMER schema map

The UPCASE.LNAME virtual field will change the case of the last name to upper case using the SQL Server `UPPER` function. The type field can be defined as DATA or EXPRESSION.

The following SQL syntax creates the CUSTOMER.CUSTOMER_V view using the `UPPER` function.

```
CREATE VIEW CUSTOMER.CUSTOMER_V( UPCASE_LNAME, ID, SAL, FNAME, LNAME, COMPANY,
ADDR1, ADDR2, CITY, STATE, ZIP, PHONE, NONCONFORMING_FLAG, UNMAPPED_U2FIELD) AS
SELECT  UPPER(LNAME), CUSTOMER.ID, CUSTOMER.SAL, CUSTOMER.FNAME, CUSTOMER.LNAME,
CUSTOMER.COMPANY, CUSTOMER.ADDR1, CUSTOMER.ADDR2, CUSTOMER.CITY, CUSTOMER.STATE,
CUSTOMER.ZIP, CUSTOMER.PHONE, CUSTOMER.NONCONFORMING_FLAG, CUSTOMER.UNMAPPED_U2FIELD
FROM CUSTOMER.CUSTOMER CUSTOMER
```

Adding the UPCASE.LNAME virtual field to the CUSTOMER schema map

The following sample code creates a new CUSTOMER.XUPPERCASE function on SQL Server 2008, using a SCALAR function. In this example, the UPCASE.LNAME virtual field will change the case of the last name to upper case using a newly created SQL Server.

```
CREATE VIEW CUSTOMER.CUSTOMER_V(FULLNAME, ID, SAL, FNAME,
LNAME, COMPANY, ADDR1, ADDR2, CITY, STATE, ZIP, PHONE,
NONCONFORMING_FLAG, UNMAPPED_U2FIELD)
AS SELECT FNAME+' '+LNAME, CUSTOMER.ID, CUSTOMER.SAL,
CUSTOMER.FNAME, CUSTOMER.LNAME, CUSTOMER.COMPANY, CUSTOMER.ADDR1,
CUSTOMER.ADDR2, CUSTOMER.CITY, CUSTOMER.STATE, CUSTOMER.ZIP,
CUSTOMER.PHONE, CUSTOMER.NONCONFORMING_FLAG,
CUSTOMER.UNMAPPED_U2FIELD FROM CUSTOMER.CUSTOMER CUSTOMER
```

The following SQL syntax displays in the EDA Schema Manager tool:

```
CREATE VIEW CUSTOMER.CUSTOMER_V( UPCASE_LNAME, ID, SAL, FNAME,
LNAME, COMPANY, ADDR1, ADDR2, CITY, STATE, ZIP, PHONE, NONCONFORMING_FLAG,
UNMAPPED_U2FIELD) AS SELECT  CUSTOMER.XUPPERCASE(CUSTOMER.LNAME),
CUSTOMER.ID, CUSTOMER.SAL, CUSTOMER.FNAME, CUSTOMER.LNAME, CUSTOMER.COMPANY,
CUSTOMER.ADDR1, CUSTOMER.ADDR2, CUSTOMER.CITY, CUSTOMER.STATE, CUSTOMER.ZIP,
CUSTOMER.PHONE, CUSTOMER.NONCONFORMING_FLAG, CUSTOMER.UNMAPPED_U2FIELD
FROM CUSTOMER.CUSTOMER CUSTOMER
```

Adding the DESCRIPTION virtual field to the CUSTOMER schema map

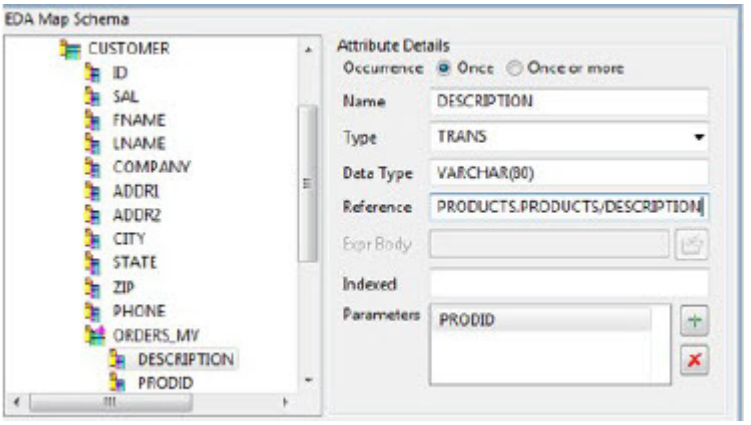
The DESCRIPTION virtual field retrieves the description information from the PRODUCTS.PRODUCTS table using the `TRANS` function. The type field is defined as `TRANS`.

Note: The PRODUCTS.PRODUCTS table must first be created on SQL Server 2008, before you use the `TRANS` function.

The PRODUCTS file is mapped to the PRODUCTS.PRODUCTS table in the EDA Schema Manager

The DESCRIPTION field is a multivalue virtual field that is part of CUSTOMER.ORDERS_MV_V view.

The following image shows the sample EDA Schema output and the associated SQL syntax:



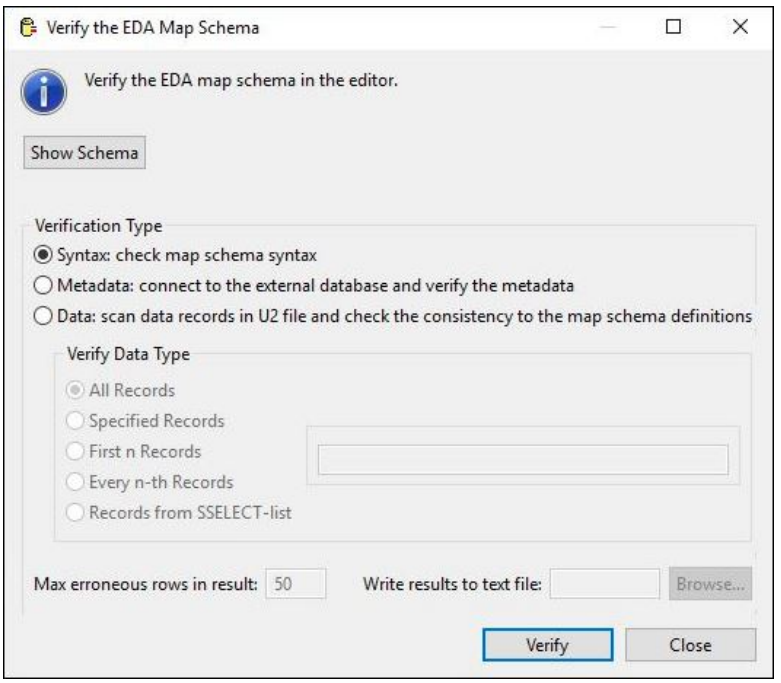

```
CREATE VIEW CUSTOMER.ORDERS_MV_V(ID, ORDERS_MV_POS, DESCRIPTION,
PRODID, SER_NUM, PRICE, BUY_DATE, PAID_DATE, SVC_PRICE, SVC_START,
SVC_END, SVC_PAID_DATE)
AS SELECT ORDERS_MV.ID, ORDERS_MV.ORDERS_MV_POS, VA2.DESCRIPTION,
ORDERS_MV.PRODID, ORDERS_MV.SER_NUM, ORDERS_MV.PRICE,
ORDERS_MV.BUY_DATE, ORDERS_MV.PAID_DATE, ORDERS_MV.SVC_PRICE,
ORDERS_MV.SVC_START, ORDERS_MV.SVC_END, ORDERS_MV.SVC_PAID_DATE
FROM CUSTOMER.ORDERS_MV ORDERS_MV LEFT OUTER JOIN
PRODUCTS.PRODUCTS VA2
ON ORDERS_MV.PRODID = VA2.ID
```

After the CUSTOMER.ORDERS_MV_V view is created, it can be accessed on SQL Server 2008.

Verifying EDA schemas

After you have created an EDA Schema, you can verify the EDA Schema.

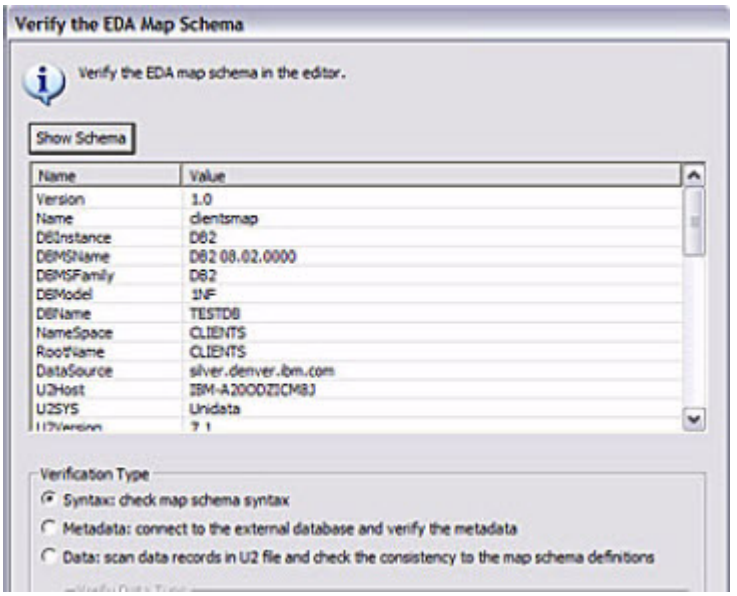
1. To verify the EDA Schema, click the **Verify** icon on the toolbar.
   A dialog box similar to the following example opens:



2. In the Verify the EDA Map Schema dialog box, select one of the following verification types:
   - **Syntax**: Verifies that the syntax of the SQL statements that creates the external tables is correct.
   - **Metadata**: Verifies that all the metadata required to create the external tables exists.
   - **Data**: Verifies that the UniData 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 UniData data file
     - **Specified Records**: You can enter specific record IDs to analyze. Separate each record ID with a right brace (}).
     - **First *n* Records**: The system verifies the first *n* records you specify.
     - **Every *n*-th Records**: The system verifies every *n*-th record you specify.
     - **Records from SSELECT-list**: The system verifies the selected records using the `SSLECT` command.
3. Indicate how you want verified results to be handled:
   - **Max erroneous rows in result**: Specify the maximum number of erroneous rows you want to display in the results. The default is 50 rows.
   - **Write results to text file**: Specify the location of the output text file that will store the results. The default is `C:\temp`.

4. To view the EDA schema, click **Show Schema**.

   The schema displays in the dialog box, as shown in the following example:



Verification example

The following example shows a listing from the STUDENT file:

```
LIST STUDENT ALL 16:35:46 Feb 24 2012 1
STUDENT..... Last Name...... First Name Major Minor Advisor. Term Crs # Gr

 521-81-4564 SMITH           Harry      CH    PSYCH Carnes   FA93 CS130 A
   IATRY
                                                              CS100 B
                                                              PY100 B
                                                        SP94 CS131 B
                                                              CS101 B
                                                              PE220 A
 424-32-5656 Martin          Sally      PY    EG    Fried    SP94 PY100 C
                                                              PE100 C
 414-44-6545 Offenbach       Karl       CS    PY    Otis     FA93 CS104 D
                                                              MA101 C
                                                              FA100 C
                                                        SP94 CS105 B
                                                              MA102 C
                                                              PY100 C
 978-76-6676 Muller          Gerhardt   FA    PY    Carnes   FA93 FA120 A
                                                              FA230 C
                                                              HY101 C
                                                        SP94 FA121 A
                                                              FA231 B
                                                              HY102 I
 221-34-5665 Miller          Susan      EG    BW    Otis     FA93 EG110 C
                                                              MA220 B
                                                              PY100 B
                                                        SP94 EG140 B
                                                              EG240 B
                                                              MA221 B
 291-22-2021 Smith           jojo       CS    FA    Eades    SP94 FA100 B
```

Notice that the Minor attribute for Record ID 521-81-4564 exceeds the specified length of 8 characters. When you verify the data the EDA Map Schema, the following error message appears:

```
In C:\U2\ud82\sys\CTLG\e\EDAMAPSUB at line 2056 EDA_write_tuple error, id =
"521814564"
In C:\U2\\sys\CTLG\e\EDAMAPSUB at line 2056 EDA DB2 Driver: [U2] [CLI
Driver] CLI0109E String data right truncation, SQLSTATE=22001
5 records passed data verification.
1 records failed on data verification.
```

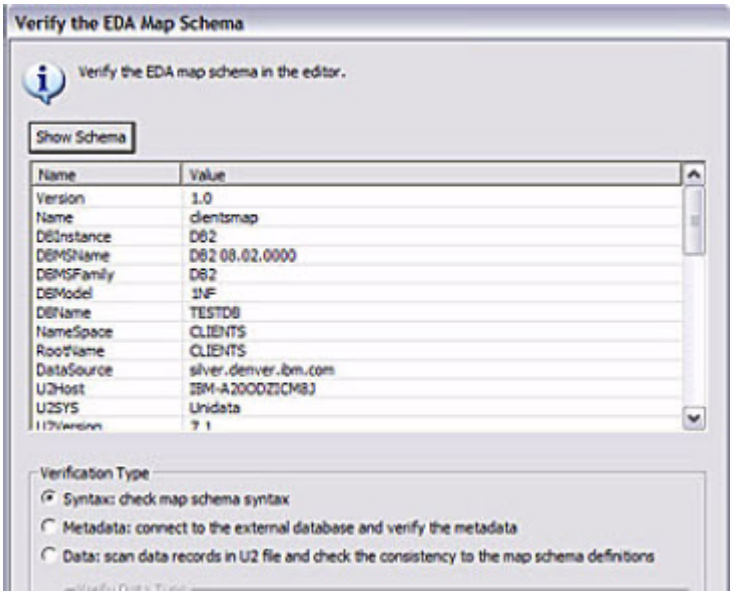
Fix the incorrect data in the records before converting the file to an EDA file. If you do not correct the data, the record will not be converted, and will appear in the `EDA_EXCEPTION` file.

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

To view the EDA schema, click **Show Schema**.

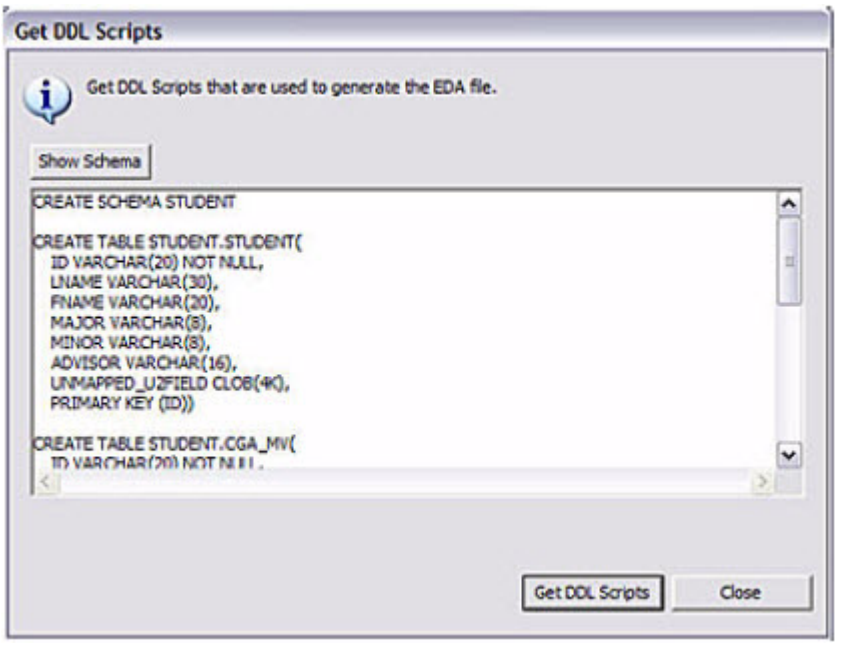
The following example shows the schema in the Verify the EDA Map Schema dialog box.



Viewing the DDL scripts

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

Get DDL scripts appear in the dialog box, as shown in the following example:



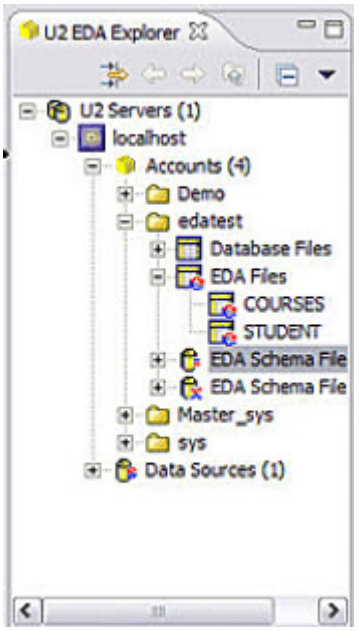
Converting data from U2 to an external database

You can convert data from U2 to the external database.

1. Click the **Convert Data** icon.
2. In the Convert the U2 File to EDA File dialog box, select the type of conversion to use:
   - Click **Force** to drop existing tables on the external database before creating new ones. You must select this option if you are reconverting data.
   - Click **Verbose** to display detailed messages and DDL scripts during the conversion process.
3. **Optional:** Click **EDA Alter**. The database suspends replication during the altering process and the new attribute is appended to the table without the table having to reload.

   When using the **EDA Alter** function, there are some limitations to keep in mind:
   - Only ADD attributes are allowed in the S-fields or the MV/MS fields.
   - Fields must be located in existing tables.
   - The server options cannot be changed. The client options can be different.
   - The WHOLE RECORD and UNMAPPED settings must not affect the BLOB field definitions. The settings must be either WHOLEREC=1, or UNMAPPED = 0 and Non-Conforming= No.
4. Click **EDA Convert**. If the conversion is successful, a message reports the total number of recordsconverted to the external database. If the conversion is not successful, error messages describe the problems.

5. To see which files were converted from U2 to the external database, from the **EDA Schema Manager**, click the plus sign (+) next to expand EDA Schema files.

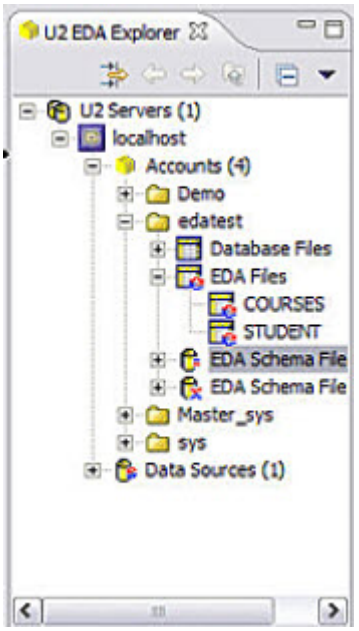


You can also determine if a file has been converted to the external database by executing the `FILE.STAT` command at the ECL prompt, as shown in the following example:

```
UniData Release 8.2.2 Build: (6093)
¬ Rocket Software, Inc. 1985-2015.
All rights reserved.
Current UniData home is C:\U2\ud82\.
Current working directory is C:\U2\ud82\Demo.
:FILE.STAT COURSES
COURSES is an EDA type file, FILE.STAT does not apply.
```

Viewing EDA files

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



You can also determine if a file has been converted to the external database by executing the `FILE.STAT` command at the ECL prompt, as shown in the following example:

```
UniData Release 8.2.2 Build: (6093)
¬ Rocket Software, Inc. 1985-2015.
All rights reserved.
Current UniData home is C:\U2\ud82\.
Current working directory is C:\U2\ud82\Demo.
:FILE.STAT COURSES
COURSES is an EDA type file, FILE.STAT does not apply.
```

Accessing data in an external database

Use UniQuery, UniData/UniVerse SQL, and UniBasic to access the data in the external database.

Listing data using UniQuery

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

```
:LIST COURSES ALL
LIST COURSES ALL 15:49:53 Jan 25 2012 1
COURSES... Course Name.............. Credits Teacher...

FA100      Visual Thinking                3 Fried
MA101      Math Principals                3 Otis
MA102      Algebra                        3 Otis
PY140      Abnormal Psychology            5 Masters
CS100      Intro to Computer Science      3 Gibson
FA120      Finger Painting                5 Fried
CS101      Intro to Computer Science      4 Gibson
FA121      Watercorlors                   3 Carnes
CS104      Database Design                3 Aaron
CS105      Database Design                3 Gibson
MA220      Calculus- I                    5 Otis
MA221      Calculus - II                  5 Otis
FA230      Photography Principals         3 Carnes
FA231      Photography Practicum          3 Fried
CS130      Intro to Operating             5 James
           Systems
EG110      Engineering Principles         5 Carnes
CS131      Intro to Operating             5 Aaron
           Systems
HY101      Western Civilization           3 Otis
```

Listing data using UniData/UniVerse SQL

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

```
:SQL SELECT * FROM COURSES;
:SQL SELECT * FROM COURSES;
Page   1
COURSES    Course Name               Credi Teacher
---------- ------------------------- ----- ----------
FA100      Visual Thinking               3 Fried
MA101      Math Principals               3 Otis
MA102      Algebra                       3 Otis
PY140      Abnormal Psychology           5 Masters
CS100      Intro to Computer Science     3 Gibson
FA120      Finger Painting               5 Fried
CS101      Intro to Computer Science     4 Gibson
FA121      Watercorlors                  3 Carnes
CS104      Database Design               3 Aaron
CS105      Database Design               3 Gibson
MA220      Calculus- I                   5 Otis
MA221      Calculus - II                 5 Otis
FA230      Photography Principals        3 Carnes
FA231      Photography Practicum         3 Fried
CS130      Intro to Operating            5 James
           Systems
EG110      Engineering Principles        5 Carnes
CS131      Intro to Operating            5 Aaron
           Systems
HY101      Western Civilization          3 Otis
```

Chapter 3: External database supplied drivers

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

The following supplied drivers are explained:

- EDA Oracle driver
  The EDA Oracle driver is a dynamic-loading library that the EDA engine uses to exchange data with an Oracle database.
- EDA DB2 driver
  The EDA DB2 driver is a dynamic-loading library which the EDA engine uses to exchange data with a DB2 database.
- EDA SQL Server driver
  The EDA SQL Server driver is a dynamic-loading library which the EDA engine uses to exchange data with a SQL Server database.
- EDA Common driver
  The EDA Common driver is used to connect to SQL Server, MySQL, or PostgreSQL from a UNIX or Linux platform. It is also used to connect to MySQL and PostgreSQL from a Windows platform.
- EDA driver log files
  The logging level for the EDA Oracle driver, EDA DB2 driver, EDA SQL Server driver, and the EDA Common driver can be set to values between 0 and 8.

EDA Oracle driver

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

The EDA Oracle driver supports Oracle Version 11g.

Parent topic: External database supplied drivers

Set up the EDA environment

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

```
DRIVER=ORACLE
ORACLEPATH=/test1/oracle/instantclient_73_0
LOGLEVEL=0
```

You can change the LOGLEVEL from 0 to 8. The higher the log level, the more information is captured.

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

Set up the Oracle connection file

Set up the `tnsnames.ora` file with the details required 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:

Platform	Oracle path location
AIX	Add Oracle library path to LIBPATH
HP	Add Oracle library path to SHLIB_PATH
Windows	Add Oracle library path to PATH
Other	Add Oracle library path LD_LIBRARY_PATH

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 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 following table describes how EDA maps U2 data to Oracle data types:

U2 data	Oracle data type
Characters	VARCHAR
Date	DATE
Time	TIMESTAMP
Number (integer)	NUMBER(38)
Number (noninteger)	NUMBER

U2 data	Oracle data type
Unmapped fields	CLOB

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

EDA DB2 driver

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

The EDA DB2 driver supports DB2 8.0 and greater.

Parent topic: External database supplied drivers

Set up the EDA environment

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

```
DRIVER=DB2
DB2INSTANCE=db2inst1
DB2PATH=/home/db2inst1/sqllib
LOGLEVEL=2
```

You can change the LOGLEVEL from 0 to 8. The higher the log level, the more information is captured.

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

Install DB2 or the DB2 client

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

Set up connection to the DB2 database

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

Create the EDA data source

Create an EDA data source in the EDA Schema Manager.

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

DB2 data type mapping

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

U2 data	DB2 data type
Characters	VARCHAR
Date	DATE
Time	TIME
Number (integer)	INTEGER
Number (noninteger)	FLOAT
Unmapped fields	CLOB

EDA SQL Server driver

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

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

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

Parent topic: External database supplied drivers

Install SQL Server

You can install SQL Server on any machine. SQL Server Native Client must be installed on the same machine where UniData is installed.

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

Create an ODBC data source to use to connect to the SQL Server database.

Use **Control Panel** → **Administrative Tools** → **Data Sources (ODBC)** to open the ODBC Data Source Administrator dialog box.

Add a system DSN and select SQL Native Client as the driver.

Create the EDA data source

Create an EDA data source in the EDA Schema Manager.

Make sure you use the ODBC data source created in the Install SQL Server, on page 48 topic as the External DB Name.

Set up the EDA configuration file

Create an `edaconfig` file in $UDTHOME for the EDA SQL Server driver log.

Add the following information to the `edaconfig` file:

```
DRIVER=SQLSRV
LOGLEVEL=0
```

You can change the LOGLEVEL from 0 to 8. The higher the log level, the more information is captured.

SQL Server data type mapping

The following table describes how EDA maps UniData data to SQL Server data types.

EDA Common driver

The EDA Common driver is used to connect to SQL Server, MySQL, or PostgreSQL from a UNIX or Linux platform. It is also used to connect to MySQL and PostgreSQL from a Windows platform.

Support

- For SQL Server, the EDA Common driver supports:
  - SQL Server 2005 and greater
  - Microsoft ODBC Driver to SQL Server on Linux 17.4 and greater
- For MySQL, the EDA Common driver supports:
  - MySQL 5.7 and greater
  - MySQL Connector/ODBC 5.3 and greater
- For PostgreSQL, the EDA Common driver supports:
  - PostgreSQL 11.4 and greater
  - PostgreSQL ODBC driver 11.4 and greater

- MySQL data type mapping
  The following table describes how EDA maps UniData data to MySQL data types.
- PostgreSQL data type mapping
  The following table describes how EDA maps UniData data to PostgreSQL data types.
- EDA Common driver for Windows
  The EDA common driver for Windows is a dynamic-loading library that the EDA engine uses to exchange data with a MySQL database.
- EDA Common driver for UNIX and Linux
  The EDA common driver for UNIX and Linux is used to access UniData installations running on a UNIX or Linux platform to Microsoft SQL Server (running on Windows), MySQL or PostgreSQL (running on Windows, UNIX, or Linux).

Parent topic: External database supplied drivers

MySQL data type mapping

The following table describes how EDA maps UniData data to MySQL data types.

UniData data	MySQL data type
Characters	VARCHAR
Date	DATE (default) or DATETIME
Time	TIME (default) or DATETIME
Number (integer)	INT
Number (noninteger)	FLOAT
Unmapped fields	LONGTEXT

Note:

Use of the DATE and TIME MySQL type is supported on UniData versions 8.2.2 and later.

When mapping UniData date data to MySQL DATETIME, the time portion is filled with 00:00:00.

When mapping UniData time data to MySQL DATETIME, the date portion is filled with 01/01/1753.

Parent topic: EDA Common driver

PostgreSQL data type mapping

The following table describes how EDA maps UniData data to PostgreSQL data types.

UniData data	PostgreSQL data type
Characters	VARCHAR
Date	DATE
Time	TIME (default) or TIMESTAMP
Number (integer)	INTEGER
Number (noninteger)	REAL
Unmapped fields	TEXT

Parent topic: EDA Common driver

EDA Common driver for Windows

The EDA common driver for Windows is a dynamic-loading library that the EDA engine uses to exchange data with a MySQL database.

Parent topic: EDA Common driver

Install MySQL

MySQL server can be installed on any machine. MySQL Connector/ODBC must be installed on the same machine where UniData is installed.

Create a database on MySQL to which UniData can connect.

Create an ODBC data source to use to connect to the MySQL database.

Use **Control Panel** → **Administrative Tools** → **Data Sources (ODBC)** to open the ODBC Data Source Administrator dialog box.

Add a system DSN and select COM as the driver.

Create the EDA data source

Create an EDA data source in the EDA Schema Manager.

Make sure you use the ODBC data source created in the Install MySQL, on page 50 topic as the External DB Name.

Set up the EDA configuration file

Create an `edaconfig` file in $UDTHOME for the EDA Common driver log.

Add the following information to the `edaconfig` file:

```
DRIVER=SQLSRV
```

```
LOGLEVEL=0
```

You can change the LOGLEVEL from 0 to 8. The higher the log level, the more information is captured.

EDA Common driver for UNIX and Linux

The EDA common driver for UNIX and Linux is used to access UniData installations running on a UNIX or Linux platform to Microsoft SQL Server (running on Windows), MySQL or PostgreSQL (running on Windows, UNIX, or Linux).

The EDA common driver for UNIX and Linux is a dynamic-loading library. It uses unixODBC as the ODBC driver manager and a third-party product, such as the Easysoft ODBC driver, to communicate with SQL Server, MySQL, or PostgreSQL. The EDA common driver for UNIX and Linux is available on AIX, HP Itanium, Solaris SPARC, and Linux platforms.

Parent topic: EDA Common driver

Set up the EDA environment

Execute the operating system-level command `edasetup.sh` to set up the EDA environment.

The `edasetup.sh` command prompts you for the following information:

MySQL `edasetup.sh`

```
Please input DRIVER (DB2, ORACLE, ODBC):
ODBC
Please input DRIVERNAME (EASYSOFT or MSODBC or MYSQL or POSTGRES):
MYSQL
Please input ODBCDRVPATH (/usr/lib64):
/usr/ud82/include/edaconfig has been updated
```

The `edaconfig` file is created in the `/usr/udxx/include/usr/uvxx/include` folder, where *xx* is the UniData version.

The `edaconfig` file contains the following information:

MYSQL `edaconfig`

```
#
#DRIVER=ODBC
DRIVERNAME=MYSQL
ODBCDRVPATH=/usr/lib64
LOGLEVEL=0
~
```

You can change the LOGLEVEL from 0 to 8. The higher the log level, the more information is captured.

PostgreSQL `edasetup.sh`

```
-sh-4.2$ edasetup.sh
Please input DRIVER (DB2, ORACLE, ODBC):
ODBC
Please input DRIVERNAME (EASYSOFT or MSODBC or MYSQL or POSTGRES):
POSTGRES
Please input ODBCDRVPATH (/user/lib64):

/usr/ud82/include/edaconfig has been updated.
```

PostgreSQL `edaconfig`

```
-SH-4.2$ vim /usr/ud82/include/edaconfig
#
#
DRIVER=ODBC
DRIVERNAME=POSTGRES
ODBCDRVPATH=/usr/lib64
LOGLEVEL=0
~
```

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 UniData is installed.

If the third-party ODBC driver includes unixODBC, you only have to install the ODBC driver.

Set up connection to external database

You must set up a connection using unixODBC to connect to the external database.

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

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

SQL Server

```
Driver=ODBC Driver 17 for SQL Server
```

```
Description=SQL SERVER ODBC
Server=den-sq160.u2lab.rs.com\QASQLSERVER,63044
Port=
User=qa
Password=1234
Database=QATEST
Option=4
Mars_Connection=Yes
```

MYSQL

```
####Driver=MySQL ODBC 8.0 ANSI Driver
Driver=MySQL ODBC 8.0 ANSI Driver
Description=Connector/ODBC 8.0
Server=dlndevluwfhu101.dev.rocketsoftware.com
Port=
User=feng
Password=1234
Database=DEVDB
Option=4
```

PostgreSQL

```
Driver=PostgreSQL
Description=PostgreSQL Data Source
Servername=dendevmvascen07.dev.rocketsoftware.com
Port=5432
UserName=postgres
Password=no1wayQA
Database=postgres
ReadOnly=no
ServerType=Postgres
ConnSettings=UseServerSidePrepare=1
ByteaAsLongVarBinary=1
Optimizer=0
## ODBC log in /tmp if set
#Debug = 1
#CommLog = 1
```

UniData 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:

SQL Server

```
[ODBC Driver 17 for SQL Server]
Description=Microsoft ODBC Driver 17 for SQL Server
Driver=/opt/microsoft/msodbcsql17/lib64/libmsodbcsql-17.4.so.2.1
UsageCount=1
```

MYSQL

```
[MySQL ODBC 8.0 Unicode Driver]
Driver=/usr/lib64/libmyodbc8w.so
UsageCount=1

[MySQL ODBC 8.0 ANSI Driver]
```

```
Driver=/usr/lib64/libmyodbc8a.so
UsageCount=1
```

PostgreSQL

```
Description=ODBC for PostgreSQL
Driver=/usr/lib/psqlodbcw.so
Setup=/usr/lib/libodbcpsqlS.so
Driver64=/usr/lib64/psqlodbcw.so
Setup64=/usr/lib64/libodbcpsqlS.so
FileUsage=1
```

Set up the ODBC dynamic loading library path

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

Set up the ODBC driver library path as described for the following platforms:

- 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 Set up connection to external database, on page 52 as the External DB Name.

Automatic data type mapping

Automatic data type mapping is described for Oracle, DB2, SQL Server, MySQL, and PostgreSQL.

- f you connect to Oracle, the data types are described in Oracle data type mapping, on page 46
- If you connect to DB2, the data types are described in DB2 data type mapping, on page 48.
- If you connect to SQL Server, the data types are described in SQL Server data type mapping, on page 49.
- If you connect to MySQL, the data types are described in MySQL data type mapping, on page 50.
- If you connect to PostgreSQL, the data types are described in PostgreSQL data type mapping, on page 50.

EDA driver log files

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

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

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

The EDA driver log files will be created in the UniData `$TMP` folder. The log file names will be like `EDA_`*`driver_name_process_id`* (for example, EDADRV_odbc_6554) .

The `$UDTHOME/logs/eda.errlog` file is also available to collect the EDA connection error information.

The following is an example for the invalid password error:

```
EDA COM Driver: [unixODBC][Microsoft][ODBC Driver 13 for SQL Server][SQL Server]
Login failed for user 'sa'
```

Parent topic: External database supplied drivers

Chapter 4: External database access driver API

EDA driver API

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

Note: EDA was not designed to access data that already resides in a 1NF database. To access this type of data, use the UniBasic 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 `$UDTHOME/sys`. The following table describes each attribute of the EDA_DRIVER record:

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

EDA driver functions

The following table lists the EDA driver functions.

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

Note: See the `edadrv_public.h` file described in The EDA driver header file, on page 70 for the definition of all data types and symbols used in the EDA Driver API. The `edadrv_public.h` file resides in the `$UDTHOME/edadrv_exam` directory.

EDADRV_LoadSymbols

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

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

Syntax

```
RETCODE EDA_LoadSymbols(numsymbols, symbols)
```

Output variables

The following table describes the output variables.

Type	Argument	Description
int *	*numsymbols*	The number of symbols returned.

Type	Argument	Description
EDA_T_SYMBOL *	*symbols*	The Driver Symbol Array pointer.

Array template

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

Array index	Array element value	Description
0	EDASYM_DBTYPE	The driver's data model. At this release, UniData only supports EDA_1NF.
1	EDASYM_DBFAMILY	The name of the driver database, such as DBMS_DB2, DBMS_MSSQLSERVER, DBMS_OTHERS. For more information, see Registering an EDA driver, on page 56.
2	EDASYM_VERSION	The version of this driver. This is for information purposes only, EDA does not use this value.
3	EDASYM_CONNECT	The pointer to EDADRV_Connect.
4	EDASYM_DISCONNECT	The pointer to EDADRV_Disconnect.
5	EDASYM_END_TRANSACTION	The pointer to EDADRV_EndTransaction.
6	EDASYM_PREPARE_STMT	The pointer to EDADRV_PrepareStmt.
7	EDASYM_DROP_STMT	The pointer to EDADRV_DropStmt.
8	EDASYM_EXECUTE_STMT	The pointer to EDADRV_Execute.
9	EDASYM_CLOSE_STMT	The pointer to EDADRV_Close.
10	EDASYM_FETCH_STMT	The pointer to EDADRV_FetchStmt.
11	EDASYM_PERFORM	The pointer to EDADRV_Perform.
12	EDASYM_SET_ATTRIBUTE	The pointer to EDADRV_GetEDAAttr.
13	EDASYM_GET_ERRMSG	The pointer to EDADRV_GetErrmsg.
14	EDASYM_CLEANUP	The pointer to EDADRV_Cleanup.
15	EDASYM_GET_DBINFO	The pointer to EDADRV_GetDBInfo.
16	EDASYM_FREE_RESULT	The pointer to EDADRV_FreeResult.
17	EDASYM_GETSPECIALINFO	The pointer to EDADRV_GetSpecialInfo.

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.
101	EDADRV_ERR_MEMORY	Internal memory allocation error.
108	EDADRV_CONFIG_NOT_EXIST	`edaconfig` file does not exist.
109	EDADRV_OPEN_CONFIG_ERROR	Could not open `edaconfig` file.

Return code	Error	Description
110	EDADRV_DB2INSTANCE_NOT_SET	DB2INSTANCE in `edaconfig` file not set.

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.

Type	Argument	Description
char *	*datasource*	The name of the data source.
char *	*login_name*	The user login name.
char *	*password*	The password corresponding to the login name.

Output variables

The following table describes the output variables.

Type	Argument	Description
EDA_T_CONN_HDL *	*connhdl*	The connection handle.

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.
1	EDADRV_ERR_SYSTEM	External system error.
101	EDADRV_ERR_MEMORY	Internal memory allocation error.

EDADRV_Disconnect

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

Syntax

```
RETCODE EDADRV_Disconnect(connhdl)
```

Input variables

The following table describes the input variables.

Type	Argument	Description
EDA_T_CONN_HDL	*connhdl*	The connection handle.

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.
1	EDADRV_ERR_SYSTEM	External system error.
102	EDADRV_INVALID_CONN_ID	Invalid connection handle.

EDADRV_EndTransaction

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

Syntax

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

Input variables

The following table describes the input variables.

Type	Argument	Description
EDA_T_CONN_HDL	*connhdl*	The connection handle.
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.

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.
1	EDADRV_ERR_SYSTEM	External system error.
102	EDADRV_INVALID_CONN_ID	Invalid connection handle.
114	EDADRV_INVALID_TRANS_FLAG	Invalid END TRANSACTION flag.

EDADRV_PrepareStmt

EDA calls the EDADRV_PrepareStmt function to prepare a statement.

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

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

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

The 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.

Syntax

```
RETCODE EDADRV_PrepareStmt(connhdl,stmthdl,stmt)
```

Input variables

The following table describes the input variables.

Type	Argument	Description
EDA_T_CONN_HDL	*connhdl*	The connection handle.
EDA_T_STMT	*stmt*	The statement content.

Output variables

The following table describes the output variables.

Type	Argument	Description
EDA_T_STMT_HDL *	*stmthdl*	The statement handle.

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.
1	EDADRV_ERR_SYSTEM	External system error.
101	EDADRV_ERR_MEMORY	Internal memory allocation error.
102	EDADRV_INVALID_CONN_ID	Invalid connection handle.
112	EDADRV_INVALID_DATATYPE	Invalid data type

EDADRV_ExecuteStmt

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

EDA provides parameter values for each input and input/output parameter. Parameter values are supplied in a string format, separated by field marks. The value of the field mark is determined by calling the 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.

Syntax

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

Input variables

The following table describes the input variables.

Type	Argument	Description
EDA_T_STMT_HDL	*stmthdl*	The statement handle.
EDA_T_STRING	*parameters*	The statement parameters.

Output variables

The following table describes the output variables.

Type	Argument	Description
int *	*rowcount*	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.

Return codes

The following table describes the return codes.

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

EDADRV_CloseStmt

EDA calls the EDADRV_CloseStmt function to close a statement.

Once a cursor is opened by the execution of the EDADRV_ExecuteStmt function, it remains open even after all the rows have been fetched. The EDADRV_CloseStatement closes any open cursors associated with the statement handle.

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

Syntax

```
RETCODE EDADRV_CloseStmt(stmthdl)
```

Input variables

The following table describes the input variables.

Type	Argument	Description
EDA_T_STMT_HDL	*stmthdl*	Statement handle.

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.
1	EDADRV_ERR_SYSTEM	External system error.
103	EDADRV_INVALID_STMT_ID	Invalid statement handle.

EDADRV_DropStmt

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

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

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

Syntax

```
RETCODE EDADRV_DropStmt(stmthdl)
```

Input variables

The following table describes the input variables.

Type	Argument	Description
EDA_T_STMT_HDL	*stmthdl*	Statement handle.

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.
1	EDADRV_ERR_SYSTEM	External system error.
103	EDADRV_INVALID_STMT_ID	Invalid statement handle.

EDADRV_FetchStmt

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

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

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

Syntax

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

Input variables

The following table describes the input variables.

Type	Variable	Description
EDA_T_STMT_HDL	*stmthdl*	The statement handle.
int	*direction*	The fetch direction. Valid values are: 0 – Fetch Forward
int	*numrows*	The number of rows to fetch.

Output variables

The following table describes the output variables.

Type	Variable	Description
int *	*rowsfetched*	The number of the row retrieved.
EDA_T_RESULT *	*result*	The result string.

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.
1	EDADRV_ERR_SYSTEM	External system error.
101	EDADRV_ERR_MEMORY	Internal memory allocation error.
103	EDADRV_INVALID_STMT_ID	Invalid cursor handle.
104	EDADRV_INVALID_FETCH_DIR	Invalid fetch direction.

EDADRV_Perform

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

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. The driver designer may choose to either Prepare and Execute or Execute Direct on the external database side.

Syntax

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

Input variables

The following table describes the input variables.

Type	Variable	Description
EDA_T_CONN_HDL	*connhdl*	The connection handle.
EDA_T_STMT	*stmt*	The statement content.

Output variables

The following table describes the output variables.

Type	Variable	Description
int *	*numrows*	The number of rows retrieved or affected.
EDA_T_RESULT *	*result*	The result string.

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.
1	EDADRV_ERR_SYSTEM	External system error.
2	EDADRV_SYSERR_OBJ_EXIST	Object already exists.
101	EDADRV_ERR_MEMORY	Internal memory allocation error.
102	EDADRV_INVALID_CONN_ID	Invalid connection handle.
106	EDADRV_TOO_MANY_DATA	Too much data fetched.
112	EDADRV_INVALID_DATATYPE	Invalid data type

Return code	Error	Description
113	EDADRV_TOO_MANY_OUT_PARAM	Too many output parameters.

EDADRV_GetEDAAttr

EDA calls the EDADRV_GetEDAAttr function to communicate an EDA attribute value to the driver.

The EDADRV_GetEDAAttr function receives an attribute name – value pair.

Syntax

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

Input variables

The following table describes the input variables.

Type	Variable	Description
int	*attribute_type*	The name of the EDA driver attribute.
EDA_T_SYMBOL	*value*	The value of the EDA driver attribute.

Valid values for the EDA driver attribute are:

Type	Attribute Name	Description
EDADRV_T_STR	EDADRV_ATTR_SYSNAME	At this release, the only valid values are UniData.
EDADRV_T_STR	EDADRV_ATTR_VERSION	Valid values are 7.1 or 7.2.
EDADRV_T_INT	EDADRV_ATTR_RM	The ASCII character representing a record mark (RM), such as ^255.
EDADRV_T_INT	EDADRV_ATTR_FM	The ASCII character representing a field mark (FM), such as ^254.
EDADRV_T_INT	EDADRV_ATTR_VM	The ASCII character representing a value mark (VM), such as ^253.
EDADRV_T_INT	EDADRV_ATTR_SM	The ASCII character representing a subvalue mark (SM), such as ^252.
EDADRV_T_INT	EDADRV_ATTR_TM	The ASCII character representing a text mark (TM), such as ^251.
EDADRV_T_INT	EDADRV_NULLVAL	The ASCII character representing the null value.

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.
105	EDADRV_INVALID_DRV_ATTR	Invalid driver attribute.

EDADRV_GetErrmsg

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

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

Syntax

```
RETCODE EDADRV_GetErrmsg(errmsg)
```

Output variables

The following table describes the output variables.

Type	Variable	Description
EDA_T_RESULT *	*errmsg*	The error message string.

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.
1	EDADRV_ERR_SYSTEM	External system error.
101	EDADRV_ERR_MEMORY	Internal memory allocation error.

EDADRV_Cleanup

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

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

Syntax

```
RETCODE EDADRV_Cleanup
```

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.

EDADRV_FreeResult

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

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

Syntax

```
RETCODE EDADRV_FreeResult(buf)
```

Input variables

The following table describes the input variables.

Type	Variable	Description
EDA_T_RESULT	*buf*	The result buffer

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.

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.

Type	Variable	Description
EDA_T_CONN_HDL	*connhdl*	The connection handle.
int	*infotype*	The information type. Valid values are: ▪ EDADRV_DBMS_NAME – The name of the database ▪ EDADRV_DBMS_VERSION – The database version. ▪ EDADRV_SERVER_NAME – The name of the instance on the external database. ▪ EDADRV_DATABASE_NAME – The name of the database.

Output variables

The following table describes the output variables.

Type	Variable	Description
EDA_T_RESULT *	*dbinfo*	The database information.

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.
1	EDADRV_ERR_SYSTEM	External system error.
101	EDADRV_ERR_MEMORY	Internal memory allocation error.
102	EDADRV_INVALID_CONN_ID	Invalid connection handle.
115	EDADRV_INVALID_INFOTYPE	Invalid information type.

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.

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

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

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

Syntax

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

Input variables

The following table describes the input variables.

Variable	Description
infotype	The information type. Valid values are: ▪ EDADRV_BLOB_FIELD – The data type for the BLOB field. ▪ EDADRV_RENAME_TABLE – The rename table statement. ▪ EDADRV_RENAME_INDEX – The rename index statement. ▪ EDADRV_DRIVER_INFO – returns the driver information, including the EDA driver API version, the EDA driver version, the EDA driver supplier name, the EDA driver creation date, the EDA driver target database name, and the EDA driver target database version. Each value is separated by "\0."
parameters	The parameter array.

Output variables

The following table describes the output variables.

Type	Variable	Description
EDA_T_RESULT *	*dbinfo*	The database information.

Return codes

The following table describes the return codes.

Return code	Error	Description
0	N/A	Successful.
101	EDADRV_ERR_MEMORY	Internal memory allocation error.
115	EDADRV_INVALID_INFOTYPE	Invalid information type.

The EDA driver header file

```
#ifndef edadrv_public_H_DEFINED
#define edadrv_public_H_DEFINED
/*
* edadrv_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 */
/*************** Datatype of Driver Symbols *****************/
typedef struct EDA_T_SYMBOL {
intsymtype;/* Symbol type, defined below */
union {
T_FUNC func;/* Pointer to driver functions */
char * strparam;/* Pointer to driver string parameters */
int intparam;/* Pointer to driver integer parameters */
} sym; /* The symbol */
char * description;/* Description of the symbol */
} EDA_T_SYMBOL;
/*********** Symbol Type Values ****************/
#define EDAT_FUNC0
#define EDAT_INT1
#define EDAT_STRING2
/*********** Datatype of EDA String ***************/
typedef struct EDA_T_STRING {
int len;/* length of data content */
int buflen;/* length of buffer */
char * buf;/* buffer pointer */
} EDA_T_STRING;/* for data transfering between driver API */
typedef EDA_T_STRING EDA_T_RESULT;
/************ Datatype of EDA Statement *************/
typedef struct EDA_T_PTYPE {
short inout;/* IN/OUT */
short datatype;/* parameter datatype */
} EDA_T_PTYPE;/* define a parameter type */
/************ Parameter IN/OUT **************/
#define EDAPARAM_IN0/* Input parameter */
#define EDAPARAM_OUT1/* Output parameter */
#define EDAPARAM_INOUT2/* In/Out Parameter */
/************ Parameter Types **************/
#define EDAPARAM_CHAR1
#define EDAPARAM_VARCHAR2
#define EDAPARAM_INTEGER3
#define EDAPARAM_LONG4
#define EDAPARAM_SHORT5
#define EDAPARAM_FLOAT6
#define EDAPARAM_DOUBLE7
#define EDAPARAM_DATE8
#define EDAPARAM_TIME9
#define EDAPARAM_TIMESTAMP10
#define EDAPARAM_CLOB11
#define EDAPARAM_BLOB12
#define EDAPARAM_NUMERIC13
#define EDAPARAM_DECIMAL14
typedef struct EDA_T_STMT {
short type;/* statement type, defined below. */
short flags;/* statement flags, defined below. */
char * stmt;/* statement buffer, stmt is null ended */
int numparam;/* number of question marks in stmt */
EDA_T_PTYPE * paramdef;/* array of parameter defs */
```

```
} EDA_T_STMT;/* EDA statement definition */
/********** Statement Types *************/
#define EDASTMT_SQL_DDL1/* SQL DDL statement */
#define EDASTMT_SQL_DML2/* SQL DML statement */
#define EDASTMT_SQL_QUERY3/* SQL QUERY statement */
#define EDASTMT_PROCEDURE 4/* Stored Procedure */
#define EDASTMT_XQUERY5/* XQUERY statement */
/********* Statement Flags ************/
#define EDASTMT_TIMEOUT 0x01
#define EDASTMT_SCROLLABLE0x02
/****** Definitionof Driver Connection Handle *******/
typedeflong EDA_T_CONN_HDL;
#define INVALID_CONN_HDL(hdl) ((hdl)<0)
#define CLEAR_CONN_HDL(hdl) ((hdl)=-1)/* Clear a handle */
/********** Definition of Driver Statement Handle **************/
typedef long EDA_T_STMT_HDL;/* Positive integer */
#define INVALID_STMT_HDL(hdl) ((hdl)<0)/* validation of handle */
#define CLEAR_STMT_HDL(hdl) ((hdl)=-1)/* Clear handle */
/*************************************************************************
* Section 2: Definition of Driver Loading API and Dynamic Symbol Array
*************************************************************************/
/******************** Driver Loading Function Name ****************/
#define EDADRV_LoadSymbols "EDA_driver_load_symbols" /* Load driver symbol
*/
extern int EDA_driver_load_symbols();
/*****************************************************************
* FUNCTION: EDADRV_LoadSymbols:Load EDA Driver Symbol Array.
* DESCRIPTION: This is the function called after a driver DLL is loaded.
* The driver should allocate the Driver Symbol Array and
* load the driver symbols into the array according to Driver Symbol
* Code. It can invoke driver initialization if needed.
* CALLS BY : Driver Manager after loading the driver DLL.
* Parameters
* int * numsymbols; OUT: The number of symbols returned.
* EDA_T_SYMBOL *symbols;OUT:The Driver Symbol Array pointer.
* RETURN : int
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*****************************************************************/
#defineEDA_SYMBOL_NUM17
/************************** Driver Symbol Codes ***********************/
#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_TRANSACTION5/* Function: End Transaction */
#define EDASYM_PREPARE_STMT6/* Function: Prepara Statement */
#define EDASYM_DROP_STMT7/* Function: Drop Statement */
#define EDASYM_EXECUTE_STMT8/* Function: Execute Statement */
#define EDASYM_CLOSE_STMT9/* Function: Close Statment */
#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 */
/************ EDASYM_DBTYPE Code: Driver Data Type ************/
#define EDA_UNKNOW 0
```

```
#define EDA_1NF 1/* Relational Database */
#define EDA_NF2 2/* Multi-valued Database */
#define EDA_XML 3/* XML Database */
#define EDA_OBJ 4/* Object Database */
/*********** EDASYM_DBFAMILY Code: Driver DBMS Family **********/
#define DBMS_OTHERS0/* Other DBMS */
#define DBMS_DB21/* DB2 family */
#define DBMS_ORACLE2/* Oracle family */
#define DBMS_MSSQL3/* Microsoft SQL Server family */
#define DBMS_U24/* U2 family */
/*************************************************************************
* Section 3: Definition of EDA API Functions
*************************************************************************/
/*********************************************************************
* FUNCTION SYMBOL: EDASYM_CONNECT
* FUNCTION : EDADRV_Connect: Make connection to external database.
* DESCRIPTION:
* CALLS BY : Connection Manager per request.
* PARAMETERS :
* EDA_T_CONN_HDL * connhdl;OUT: The connection handle.
* char * datasource;IN: The datasource name.
* char * loginname;IN: login user name.
* char * password;IN: login password.
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*********************************************************************/
/*********************************************************************
* FUNCTION SYMBOL: EDASYM_DISCONNECT
* FUNCTION : EDADRV_Disconnect: Disconnect a connection.
* DESCRIPTION:
* CALLS BY : Connection Manager per request.
* PARAMETERS :
* EDA_T_CONN_HDL connhdl;IN: the connection handle.
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*********************************************************************/
/*********************************************************************
* FUNCTION SYMBOL: EDASYM_END_TRANSACTION
* FUNCTION : EDADRV_EndTransaction, Ending a transaction on a
* connection.
* DESCRIPTION: Either Commit or Rollback a transaction.
* CALLS BY : Transaction Control Module.
* PARAMETERS :
* EDA_T_CONN_HDL connhdl;IN: the connection handle
* int trans_flag;IN: Commit or Rollback.
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*********************************************************************/
/********* Transaction flag for EDADRV_EndTransaction *********/
#define EDA_COMMIT0
#define EDA_ROLLBACK1
/*********************************************************************
* FUNCTION SYMBOL: EDASYM_PREPARE_STMT
* FUNCTION : EDADRV_PrepareStmt: Prepare a statement.
* DESCRIPTION: Driver create a statement by given connection handle
* and a statement string. Statement string may include Question Marks
```

```
* for later parameter input. A list of parameter types is specified for
* later execution. The number of auguments must match the number of
* Question Marks, otherwise it will cause errors in the execution.
* If the prepared statement is a stored procedure, the values of input
* parameters will be passed in by EDASYM_ExecuteStmt, and the results
* of output parameters will be passed out by EDASYM_FetchStmt.
* NOTE: Driver can use a pre-allocated handle to prepare the
* statement; or allocate a new handle every time according to its
* implementation.
* CALLS BY : Transaction Control Module.
* PARAMETERS :
* EDA_T_CONN_HDL connhdl;IN: connection handle.
* EDA_T_STMT_HDL *stmthdl;OUT: the statement handle.
* EDA_T_STMT stmt;IN: the statement content.
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*********************************************************************/
/*********************************************************************
* FUNCTION SYMBOL: EDASYM_EXECUTE_STMT
* FUNCTION : EDADRV_ExecuteStmt:
* DESCRIPTION: EDADRV_ExecuteStmt open the execution of a statement
* by given parameters. The statement must have been prepared and the
* number and types of parameters given here must match those in
* preparing. Parameters are separated by FM. If the statement handle is
* still open, EDADRV_ExecuteStmt will close the old one and re-open it
* with the given parameters.
* CALLS BY : Transaction Control Module
* PARAMETERS :
* EDA_T_STMT_HDL stmthdl;IN: Statement handle
* EDA_T_STRING parameters;IN: Statement parameters
* int*rowcount;OUT: number of row affected
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*********************************************************************/
/*********************************************************************
* FUNCTION SYMBOL: EDASYM_CLOSE_STMT
* FUNCTION: EDADRV_CloseStmt:Close the execution of a statement.
* DESCRIPTION: A statement is opened after calling EDADRV_ExecuteStmt
* and will keep opening even though all rows has been fetched by
* EDADRV_FetchStmt. EDADRV_CloseStmt close the execution of the
* statement.
* CALLS BY : Transaction Control Module
* PARAMETERS :
* EDA_T_STMT_HDL stmthdl; IN: the statement handler
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*********************************************************************/
/*********************************************************************
* FUNCTION SYMBOL: EDASYM_DROP_STMT
* FUNCTION : EDADRV_DropStmt: Drop a statement.
* DESCRIPTION: EDADRV_DropStmt drop a statement and make it unavailable.
* If a statement is opened EDADRV_DropStmt will close it before
* dropping it.
* CALLS BY : Transaction Control Module
* PARAMETERS :
* EDA_T_STMT_HDL stmthdl; IN: the statement handler
```

```
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*****************************************************************/
/*****************************************************************
* FUNCTION SYMBOL: EDASYM_FETCH_STMT
* FUNCTION : EDADRV_FetchStmt: Fetch an open statement
* DESCRIPTION: EDA always treat driver statement as scrollable cursor.
* The format of result differs on the driver's data type:
* EDA_1NF: Result is rows returned by SQL statement. Columns
* are separated by '\0' and rows are separated by RM.
* EDA_XML: Result is XML document.
* EDA_NF2: Result is multi-valued records separated by RM.
* EDA_OBJ: Result is self-defined.
* CALLS BY : Transaction Control Module
* PARAMETERS :
* EDA_T_STMT_HDL stmthdl;IN: statement handle.
* int direction;IN: fetch direction.
* int numrows;IN: number of rows to fetch.
* int *rowsfetched;OUT: number of rows fetched.
* EDA_T_RESULT *result;OUT: the result string.
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*****************************************************************/
/************* Fetch Direction ************/
#define EDA_FETCH_FORWARD0
#define EDA_FETCH_BACKWARD1
#define EDA_FETCH_FIRST2
#define EDA_FETCH_LAST3
/*****************************************************************
* FUNCTION SYMBOL: EDASYM_PERFORM
* FUNCTION : EDADRV_Perform: Perform a statement directly onto
* external DBMS.
* DESCRIPTION: The statement could be a DDL, DML or SQL query without
* question marks. If the statement is a stored procedure, the question
* marks in the statements must represent output parameters.
* If the statement has result returned, the format of returned
* result is same as those defined in EDA_FetchStmt.
* CALLS BY : Transaction Control Module
* PARAMETERS :
* EDA_T_CONN_HDL connhdl;IN: connection handle.
* EDA_T_STMT stmt;IN: statement content.
* int *numrows;OUT: number of rows fetched or affected.
* EDA_T_RESULT *result;OUT: the result string.
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*****************************************************************/
/*****************************************************************
* FUNCTION SYMBOL: EDASYM_GET_EDA_ATTR
* 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_SYSNAME1/* EDADRV_T_STR: UniData or UniVerse*/
#define EDADRV_ATTR_VERSION2/* EDADRV_T_STR: U2 Version string */
#define EDADRV_ATTR_RM3/* EDADRV_T_INT: U_RM */
#define EDADRV_ATTR_FM4/* EDADRV_T_INT: U_FM */
#define EDADRV_ATTR_VM5/* EDADRV_T_INT: U_VM */
#define EDADRV_ATTR_SM6/* EDADRV_T_INT: U_SM */
#define EDADRV_ATTR_TM7/* EDADRV_T_INT: U_TM */
#define EDADRV_ATTR_NULLVAL8/* EDADRV_T_INT: U_NULLVAL */
/* 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: EDASYM_GET_DBINFO
* FUNCTION : EDADRV_GetDBInfo: Get database information.
* DESCRIPTION: Get general information about the DBMS taht the
* application is currently connected to.
* CALLED BY: Driver Manager
* PARAMETERS :
* EDA_T_CONN_HDL connhdl;IN: connection handle.
* intinfotype;IN: information type
* EDA_T_RESULT *dbinfo;OUT: the DB info.
* RETURN :
```

```
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*****************************************************************/
/************* Infotypes in EDADRV_GetDBInfo ************/
#define EDADRV_DBMS_NAME1/* DBMS name */
#define EDADRV_DBMS_VERSION2/* DBMS version */
#define EDADRV_SERVER_NAME3/* DB2 instance */
#define EDADRV_DATABASE_NAME4/* Database name */
/*****************************************************************
* FUNCTION SYMBOL: EDASYM_GET_SPECIALINFO
* FUNCTION : EDADRV_GetSpecialInfo: Get special information
* for external database or driver.
* DESCRIPTION : Get special information about the DBMS that the
* application is currently connected to, like rename
* table, rename index, blob data type. Also get the
* information for this driver.
* CALLED BY : Driver Manager
* PARAMETERS :
* int infotype; IN: information type
* EDA_T_STRING parameters; IN: parameter array
* EDA_T_RESULT *spinfo; OUT: the special info.
* RETURN :
* 0: Succeeded;
* errcode if failed.
* GLOBAL :
*****************************************************************/
/************* Infotypes in EDADRV_GetSpecialInfo ************/
#defineEDADRV_BLOB_FIELD1/* data type for BLOB field */
#defineEDADRV_RENAME_TABLE2/* get rename table statement */
#defineEDADRV_RENAME_INDEX3/* get rename index statement */
#defineEDADRV_DRIVER_INFO4/* EDA driver info. */
/* More to be defined */
/************************ Driver Error Codes ************************/
#define EDADRV_ERR_SYSTEM1/* General External system error */
#define EDADRV_SYSERR_OBJ_EXIST2/* Object existing */
#define EDADRV_ERR_MEMORY101/* Internal memory allocation error */
#define EDADRV_INVALID_CONN_ID102/* Invalid connection handle */
#define EDADRV_INVALID_STMT_ID103/* Invalid statement handle */
#define EDADRV_INVALID_FETCH_DIR 104/* Invalid fetch direction */
#define EDADRV_INVALID_EDA_ATTR105/* Invalid EDA attribute */
#define EDADRV_TOO_MANY_DATA106/* Too many data fetched */
#define EDADRV_GET_UDTHOME_ERROR 107/* Get UDTHOME error */
#define EDADRV_CONFIG_NOT_EXIST 108/* edaconfig file not exist */
#define EDADRV_OPEN_CONFIG_ERROR 109/* Open edaconfig error */
#define EDADRV_DB2INSTANCE_NOT_SET 110/* DB2INSTANCE not set */
#defineEDADRV_INVALID_PARAM_TYPE 111/* Invalid parameter type */
#defineEDADRV_INVALID_DATATYPE 112/* Invalid data type */
#defineEDADRV_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

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

ALTER.EDAMAP

Beginning at UniData 8.1.0, an enhancement has been made to EDA that allows users to alter or change existing schemas without having to reload the existing tables. The `ALTER.EDAMAP` command allows users to add an attribute, such as a column, to the EDA Schema that will append the new attribute to the table without the table having to reload.

Syntax

`ALTER.EDAMAP``{`*`eda_schema`*`}{EDA.FILE` *`eda_file`*`}`

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
eda_schema	Specifies the name of the EDA schema to verify.
eda_file	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 UniData file name in the schema that UniData verifies.

EDA.CONNECT

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

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

Syntax

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

Parameters

Parameter	Description
datasource	The name of the data source to which you are connecting. The data source must exist in the EDA_DATASOURCE file.

Parameter	Description
WITH *logon_name,password*	The logon name on the external data source. If you do not specify *logon_name*, U2 searches the EDA_DATASOURCE file for a qualified user. If you specify *logon_name* without *password*, U2 searches the Connection Password file and connects with *logon_name* and that password. If you specify both *logon_name* and *password*, U2 uses both the make the connection.

EDA.CONVERT

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

If the U2 file you are converting is an EDA file, the conversion process removes the file and creates the new EDA file. Hash files and multilevel hash files can be converted to EDA files.

If the file exists, but is not an EDA file, the conversion process renames the file as *`filename`*`.edasave` and creates the new EDA file.

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

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

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 |
RENAME]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
eda_schema	Specifies the name of the EDA schema to use for the conversion. The schema resides either in the _&EDAMAP_& or _&EDAXMAP_& file.
eda_file	Specifies the name of the EDA file from which U2 extracts the EDA schema. If you specify FILE.NAME *target_file*, U2 uses the schema to convert *target_file*, U2 remaps *eda_file*.
DEFAULT.MAP	Specifies only to map the primary key (@ID) when converting a U2 file to EDA.
data_source	Specifies the data source name to use for the conversion.
primary_table	Specifies the name of the primary table, containing singlevalued attributes, to use for the conversion. If you also specify *name_space*, U2 uses it as the external schema name for the target external table/view set.
target_file	Specifies the name of the U2 file to convert. If you also specify *eda_schema*, *target_file* overrides the name of the U2 file contained in *eda_schema*. If you specify *eda_file*, U2 extracts the EDA schema from *eda_file* and uses it to convert *target_file* to EDA.

Parameter	Description
FORCE	Specifies that all existing external tables, views, indexes and user-defined functions are dropped prior to remapping the file.
VERBOSE	Displays the Data Definition Language (DDL) used in the conversion process.
RENAME	Allows the EDA conversion process to complete. When the EDA.CONVERT command is used on a database file where the **I-Type** field is working with the same file on Windows, the conversion process cannot complete due to the same file being in use. To avoid this, you must exit the current session to release the file and restart a U2 session. After the file is released, you can run EDA.CONVERT with the RENAME parameter to finish the EDA conversion process.

EDA.DISCONNECT

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

Syntax

`EDA.DISCONNECT` *`datasource`*

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

EDA.EXCEPTION

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

For more information about this file, see The EDA_EXCEPTION file, on page 84.

Syntax

```
EDA.EXCEPTION [ON | OFF]
```

By default, `EDA.EXCEPTION` is set to ON.

EDA.LOG

Use the `EDA.LOG` command to write a text log file to `C:\U2\ud82\udtemp`. The format of the log file is `EDA_`*`EDATEST`*`_nnnn` where *EDATEST* is the account name and *nnnn* is the pid.

Syntax

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

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

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

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

EDA.VERSION

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

Syntax

EDA.VERSION *datasource*

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

The EDA.VERSION command returns the following information:

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

LIST.EDAMAP

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

Syntax

LIST.EDAMAP{[XMAP] *eda_schema* | EDA.FILE [DICT] *eda_file* | DEFAULT.MAP}
[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.

Parameter	Description
eda_schema	Specifies the name of the EDA schema to display.
eda_file	Specifies the name of the EDA file whose schema is to be extracted and displayed. If you specify FILE.NAME *target_file, target_name* replaces the U2 file name in the schema U2 displays.
DEFAULT.MAP	Specifies to only display the primary key (@ID), irrespective of the attributes actually mapped of the schema you specify.
data_source	Specifies the data source name to use when displaying the schema.
primary_table	Specifies the name of the primary table, containing only singlevalued attributes, to use when displaying the schema. If you also specify *name_space*, U2 uses it for Name Space (external Schema Name) in the display.
target_file	Specifies the name of the U2 file to use when displaying the schema.
XMAP	Specifies to display the EDA schema in XML format.
OBJECT.TREE	Specifies to display the logical tree structure of the external table and view.
DDL	Specifies to display the Data Definition Language (DDL) statements used in the conversion process.

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.

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

SELECT.EDA.NONCONFORMING

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

Syntax

```
SELECT.EDA.NONCONFORMING filename
```

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

Example

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

```
:SELECT.EDA.NONCONFORMING TAPES
1 records selected to list 0.
```

```
>SAVE.LIST TAPES.NONCONFORMING
1 key(s) saved to 1 record(s).
:GET.LIST TAPES.NONCONFORMING
1 records retrieved to list 0.
LIST TAPES ALL 14:21:28 Oct 06 2006 1
TAPES B3333
Tape Name Love Story
Retail Charge 2.50
Copies Owned 2
Rented 2
Times Rented 19
Tape Cost Highly Recommended
Actors Ali McGraw
Ryan O'Neil
Director
Type of Video R
TJ
1 record listed
```

In this example, the Tape Cost field contains a string, when the dictionary attribute is defined as a numeric field.

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.

Parameter	Description
eda_schema	Specifies the name of the EDA schema to verify.
eda_file	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 U2 file name in the schema U2 verifies.
DEFAULT.MAP	Specifies to only verify the primary key (@ID) mapping, irrespective of the attributes actually mapped of the schema you specify.
data_source	Specifies the data source name to use when verifying the schema.
primary_table	Specifies the name of the primary table, containing only singlevalued attributes, to use when verifying the schema. If you also specify *name_space*, U2 uses it as the external schema name.
target_file	Specifies the name of the U2 file to use when verifying the schema.
METADATA	Connects to the external database and verifies the metadata on that database.

Chapter 6: EDA exception handling

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

Note: We recommend that you verify data and correct any exceptions prior to converting your data to an external database. For more information about data verification, see Verifying EDA schemas, on page 37.

You may choose to avoid certain external database errors being returned to your application by using the NONCONFORMING RECORD flag. For more information about this flag, see Defining options, on page 24.

Following are the EDA exceptions that can occur:

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

When U2 detects an exception, the following events occur:

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

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

The EDA_EXCEPTION file

The `EDA_EXCEPTION` file is a multilevel file, with each subfile relating to one EDA data source. The name of the subfile is `EDA_`*`datasource`*. The `EDA_EXCEPTION` file resides in `/udthome/sys` on UniData for UNIX and `\udthome\sys` on UniData for Windows platforms. Enter `EDA.EXCEPTION ON` to utilize the `EDA_EXCEPTION` file.

U2 records exceptions occurring during the conversion process or an INSERT, UPDATE, or DELETE operation in the `EDA_EXCEPTION` file.

Note: The command, `EDA.EXCEPTION` uses a period, while the file uses an underscore: `EDA_EXCEPTION`. The following example shows this difference.

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

>CT VOC EDA_EXCEPTION
    EDA_EXCEPTION
0001 File - EDA Exception File.
0002 C:\U2\ud82\EDA_EXCEPTION
0003 C:\U2\ud82\D_EDA_EXCEPTION
>
```

The following table describes each attribute of the EDA_EXCEPTION file.

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

Example

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

```
01 CT EDA_EXCEPTION,EDA_SQLS 4480-1470758635-1
4480-1470758635-1
0001 C:\U2\UV\HS.SALES
0002 H
0003 C:\U2\UV\HS.SALES\H
0004 DEN-VM-T14\Administrator
0005 17754
0006 36235
0007 EDA SQL Server Driver: [Microsoft][SQL Server Native Client 11.0][SQL Serv
er]Conversion failed when converting date and/or time from character strin
g.
0008 CONVERT
0009
0010
```

```
0011
0012
0013
0014 44
0015 Ms.
0016 Jill
0017 Kahn
0018 Fast Copy Center
0019 12 School St.
Press any key to continue...
0020
0021 Boston
0022 MA
0023 01103
0024 (617)555-7396
0025 C3000
0026 800311
0027 16500
0028 8410DATE
0029 8439
0030 600
0031 8414
0032 8564
0033 8439
0034
0035
```

INMAT

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

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

Chapter 7: EDA Replication

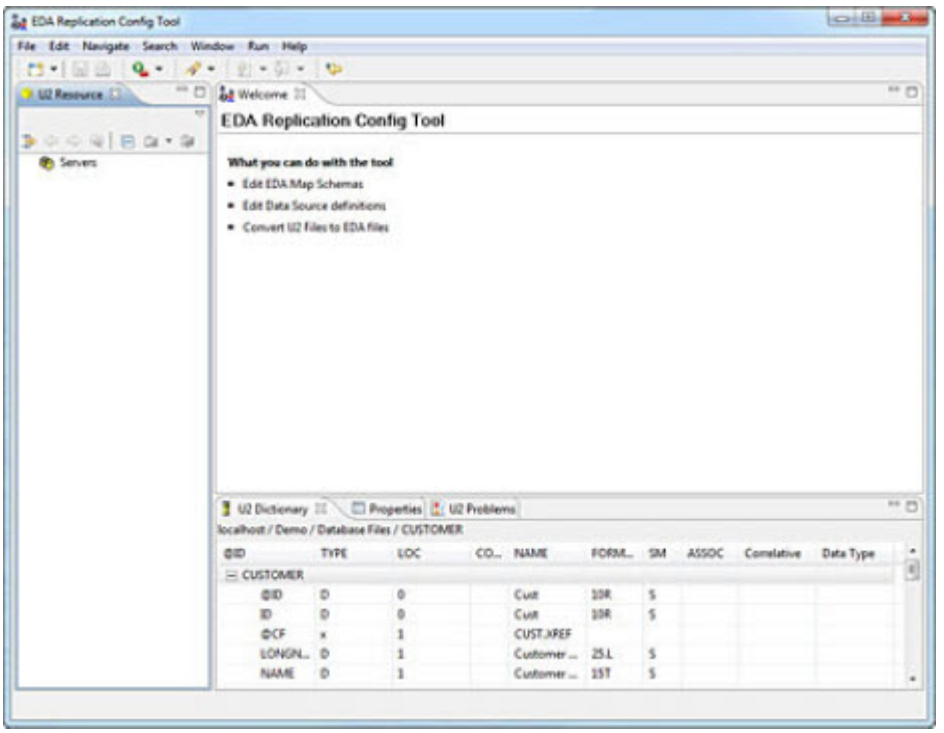
EDA Replication is useful if you want to maintain an account from which you can create reports. This increases your options for keeping your applications available. Now you can replicate your data to an SQL database in addition to keeping your data safely stored in U2. When you store data in U2, it is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

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

Note: You must connect using SQL Server authentication to use replication.

Setting up a server

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



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

- Creating a new U2 server connection, on page 13
- Connecting to the U2 server, on page 15

Defining a data source

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

To define a new data source, connect to your U2 server, right-click **Data Sources**, then click **New EDA Data Source**. The Create a New EDA Data Source dialog box appears.

In the **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.

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

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

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

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

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

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

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

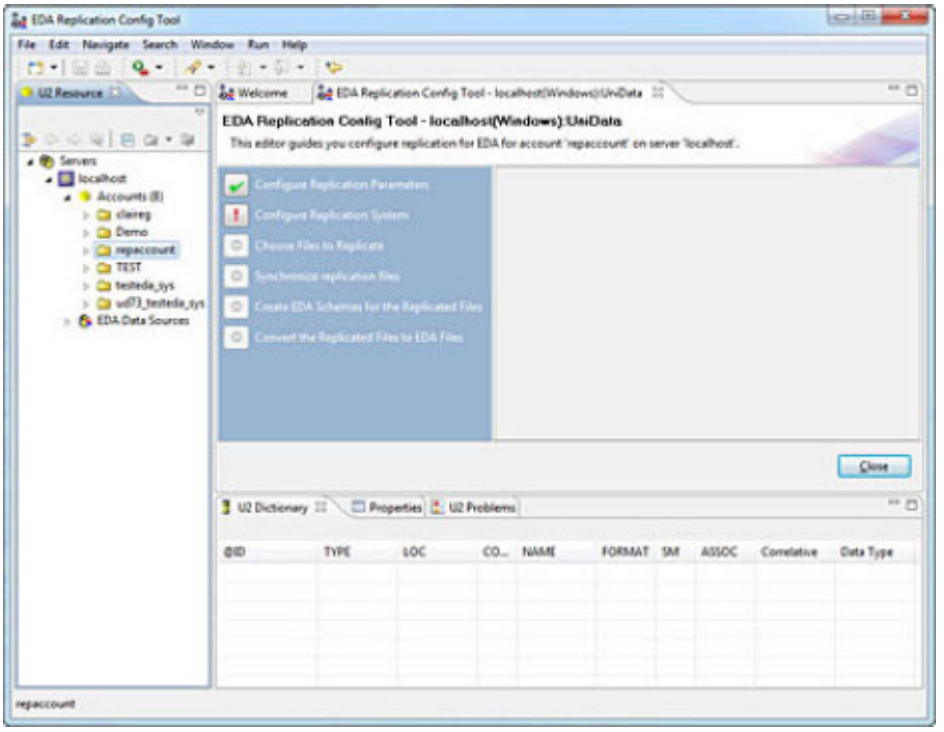
Note: If you do not use UniBasic transactions, each U2 database operation, such as a READ or WRITE, corresponds to an external transaction.

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

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

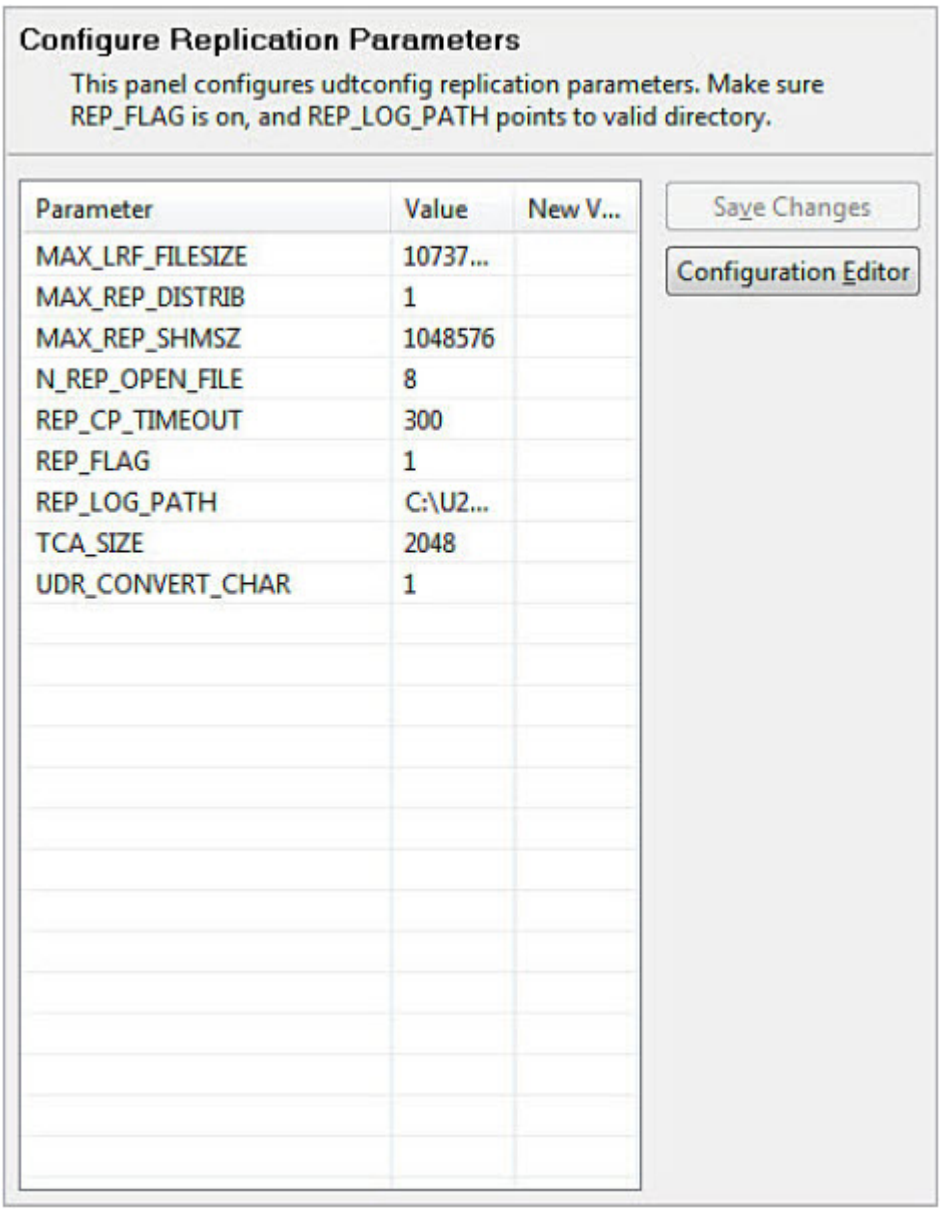
Defining EDA Replication parameters

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



Configuring replication parameters

Click **Configure Replication Parameters**. The **Configure Replication Parameters** panel appears, as shown in the following example:



To change the value of a configuration parameter, click the **New Value** column of the parameter you want to change, then enter the new value for the parameter.

MAX_LRF_FILESIZE

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

MAX_REP_DISTRIB

Reserved for internal use.

MAX_REP_SHMSZ

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

N_REP_OPEN_FILE

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

REP_CP_TIMEOUT

Specifies the cm daemon timeout interval for replication at checkpoint. The default value is 200 seconds. If this value is set to 0 the cm daemon will not time out.

REP_FLAG (replication flag)

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

Value	Description
0 (zero)	The Data Replication System is off.
Any positive integer	The Data Replication System is on.

REP_LOG_PATH

The full path to the location of the replog file.

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.

UDR_CONVERT_CHAR

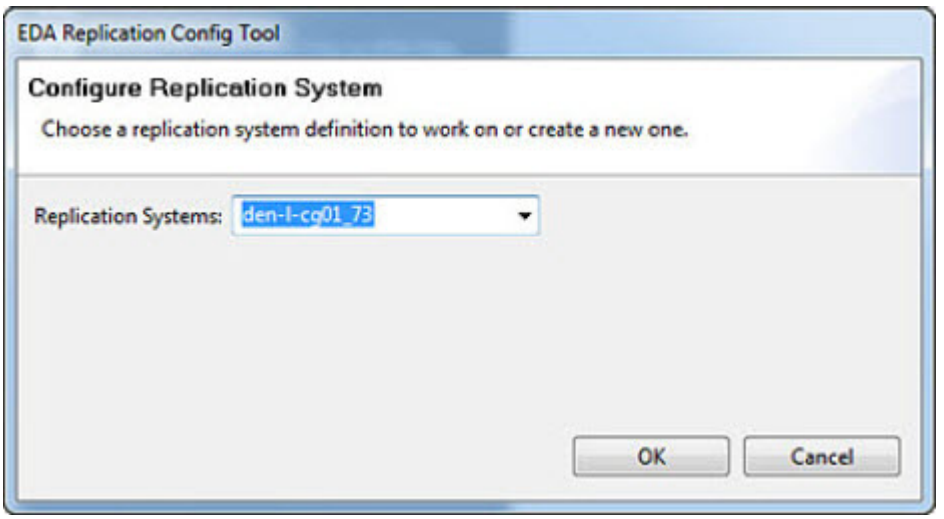
When this value is set to 1, if the publishing system and the subscribing system have a different I18N group, UniData converts marks and SQLNULL marks to those on the local machine on the data passed between the two systems. The default value is 0.

After you make your desired changes, click **Save Changes.**

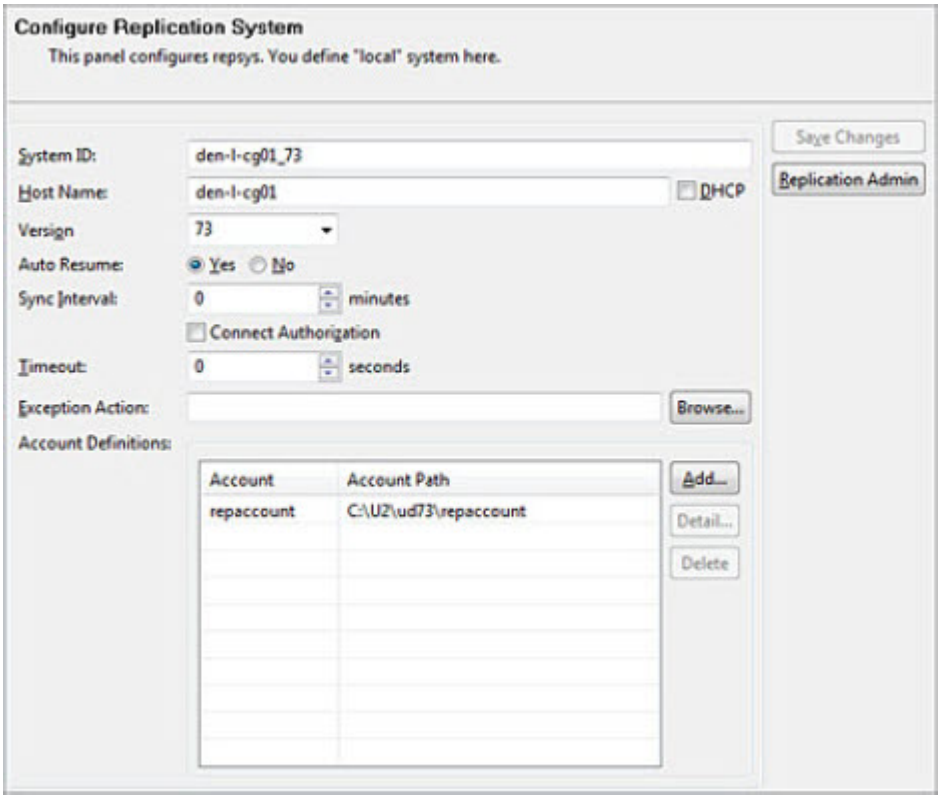
Configuring the replication system

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

1. Click **Configure Replication System**.
   The Configure Replication System dialog box opens, as shown in the following example:



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



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

4. In the **Host Name** field, enter the host name of the replication system location. A system can have only one host name.
5. In the **Version** field, select the version of U2 on the system location. The version number must be 60 or higher.
6. Select the **DHCP** check box if you want to specify that the local system has a dynamic IP address.

   If you define a local system as a DHCP system, Data Replication automatically sends the current IP address in the SYNC request to the server.
7. Next to **Auto Resume**, select **Yes** or **No**.

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

   Sync Interval defines the time interval, in minutes, in which the replication system automatically synchronizes replication.

   Data Replication automatically synchronizes subscribing systems with their publisher every period defined by sync_interval. A sync_interval of 0 indicates a manual synchronization system, where the system does not automatically synchronize the systems.

   Sync Interval applies only to those subscribing groups that have deferred replication. It does not apply to publishing groups.
9. To verify the subscribing system, select the **Connect Authorization** check box.

   Data Replication performs an authorization check when a SYNC request is received from the subscribing system.

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

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

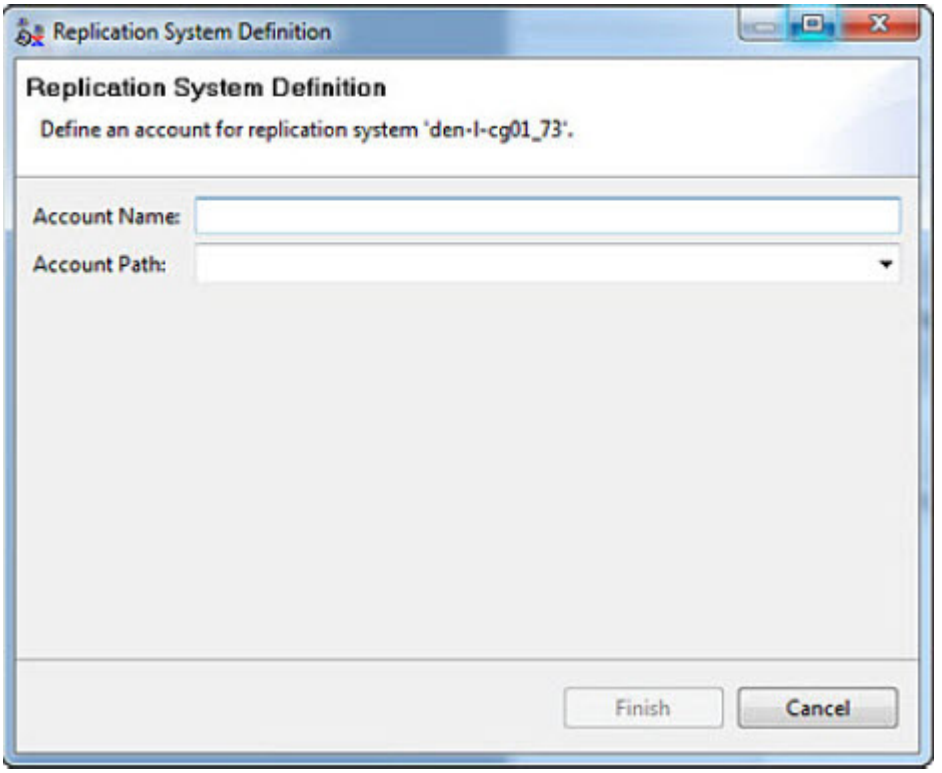
    The publishing server sends a packet to the subscribing system approximately every 4 seconds when replication is idle. The subscribing server then sends a packet back to the publishing system. If the subscribing system location has time out defined, the `publistener` process counts the time that has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

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

    We recommend that you not set the value of the TIMEOUT phrase to less than 2 minutes.
11. Next to **Exception Action**, click **Browse** to define the full path to the exception action.

    The replication exception action is a shell script on UNIX platforms, or a batch program on Windows platforms. For example, if you define a replication exception action as `UDRepExceptionAction.sh` in the `/usr/ud82` directory, browse to that directory.
12. The account definition is automatically populated with the account you previously defined. To define a different account to which to replica EDA data, click **Add** in the **Account Definitions** area.

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



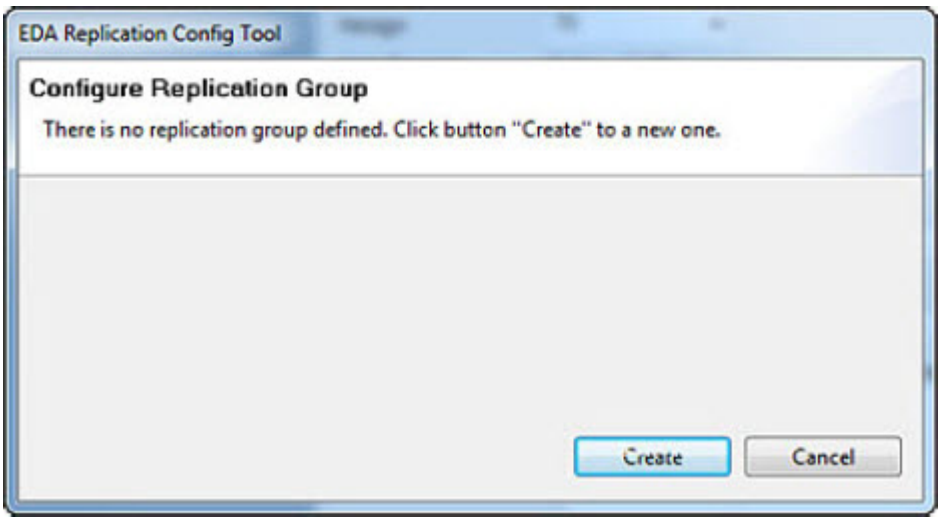
a. In the **Account Name** field, enter the name of the account.
b. In the **Account Path** field, select the path to the account.
c. Click **Finish** to save the definitions.

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

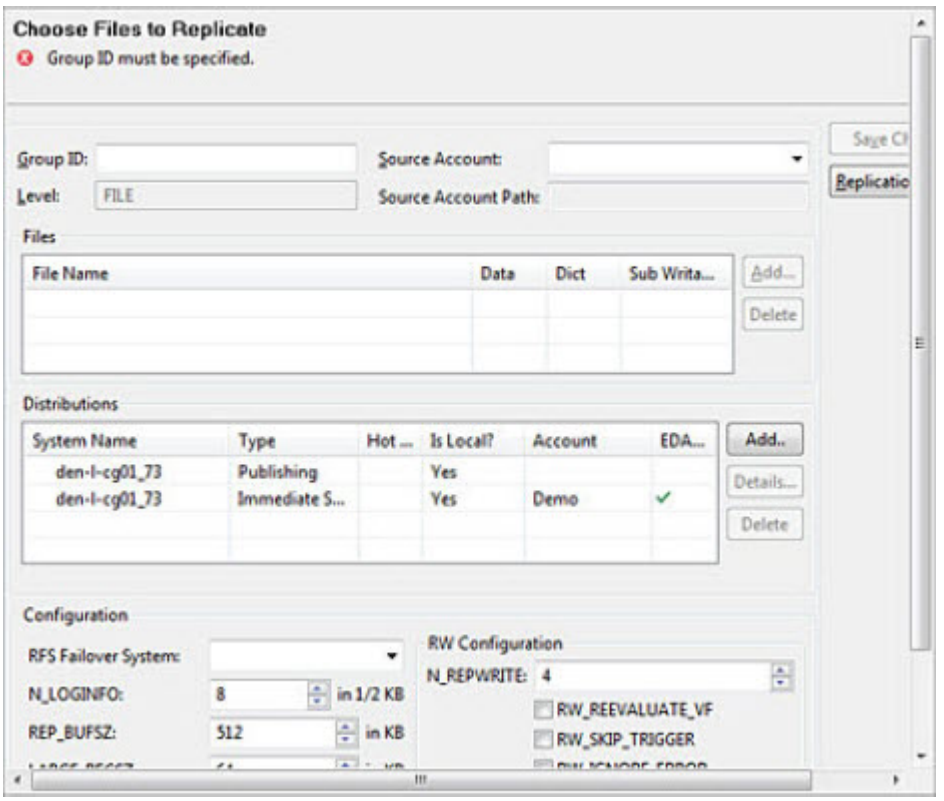
Choosing files to replicate

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

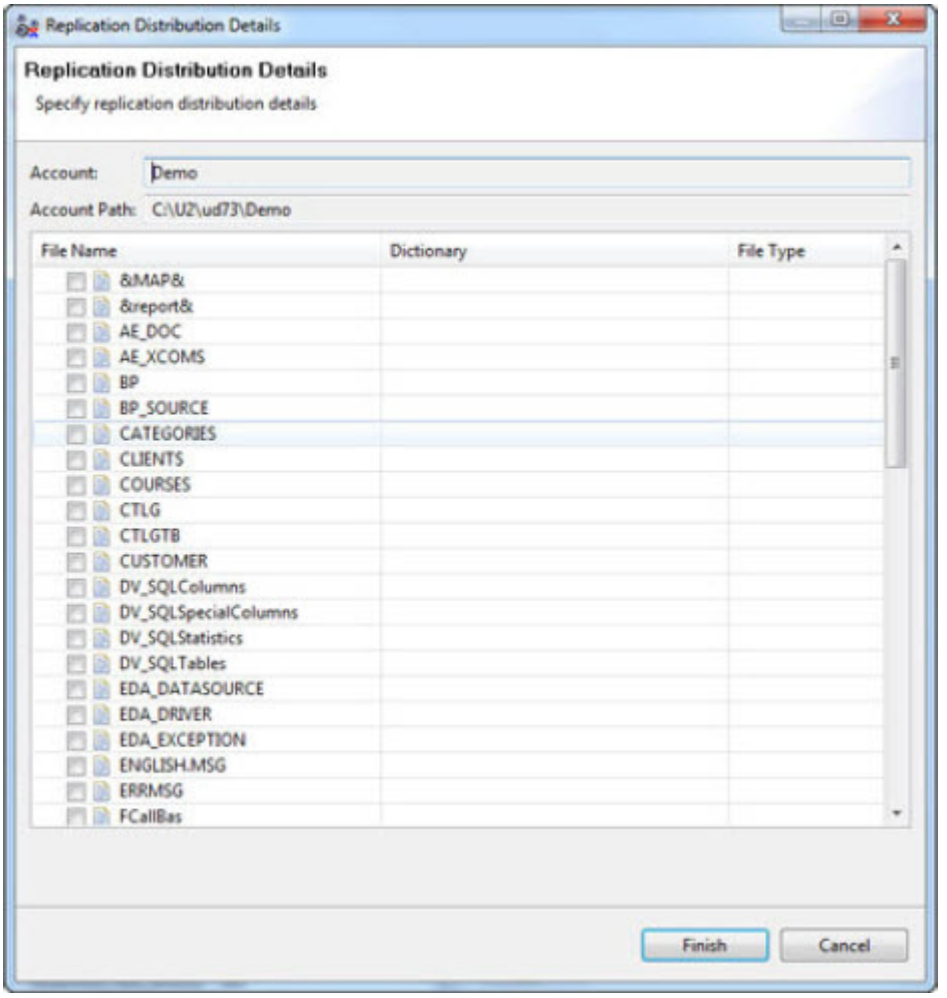
1. Click **Choose Files to Replicate**.
   If you have not previously defined a group, the following message appears:



2. Click **Create** to define a replication group.
   The Choose Files to Replicate dialog box appears, as shown in the following example:



   a. In the **Group ID** field, enter a unique name for the subscribing group.
   b. In the **Level** field, select the level of replication. You can only select FILE for EDA Replication.
   c. In the **Files** area, click **Add** to select the files you want to publish.
      A dialog box similar to the following example appears:



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

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

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

   The Replication Distribution Details dialog box appears.

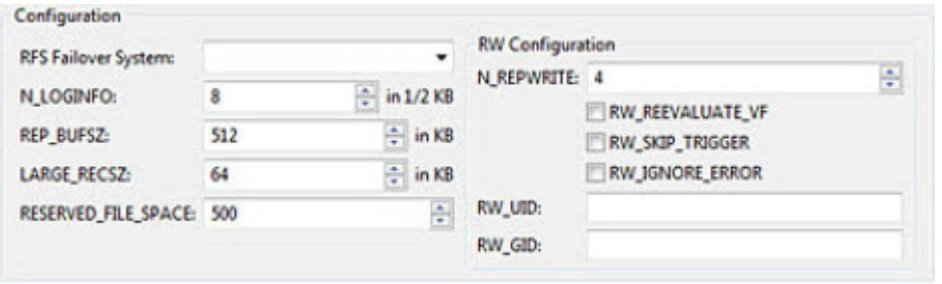
   Select the local system. Select the Replication mode you want to use. For information about types of Data Replication, see the *U2 Data Replication User Guide*.

   Click **Finish**.

e. If you want this publishing group to automatically failover to a standby system, select the standby system in the **RFS Failover System** field.

f. Set the configuration parameters.

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



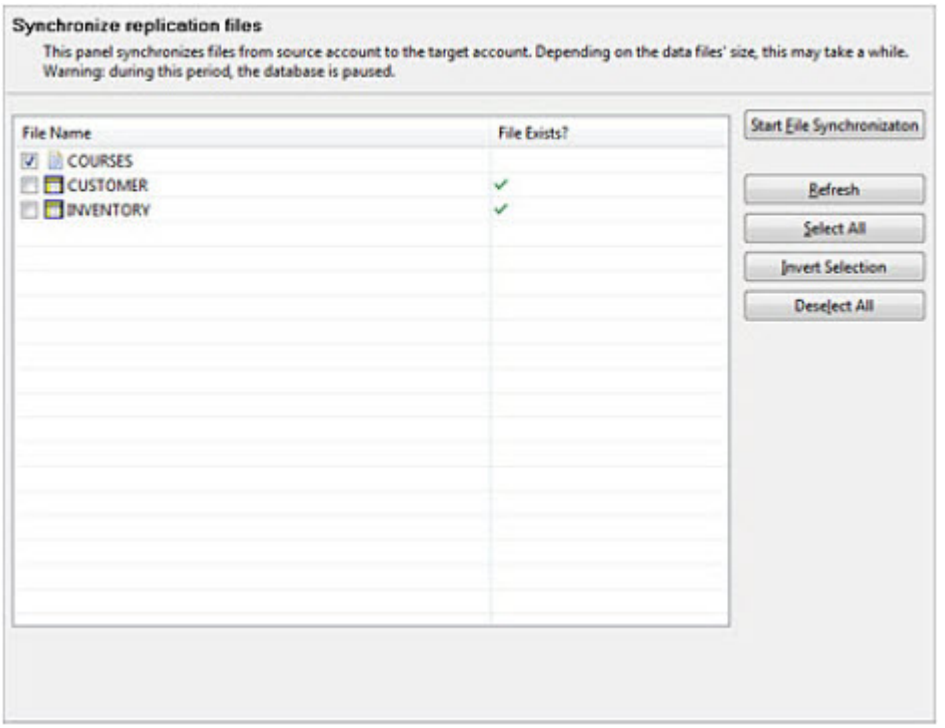
For information about these parameters, see the *U2 Data Replication User Guide*.

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

Synchronizing replication files

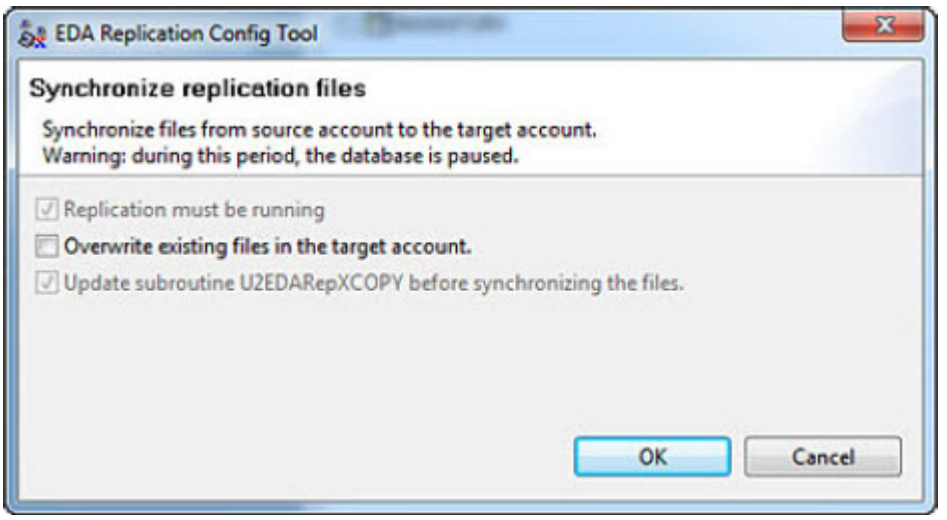
To synchronize replication files, perform the following steps.

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



2. Select the files from the source account that you want to synchronize with the target account, then click **Start File Synchronization**.

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



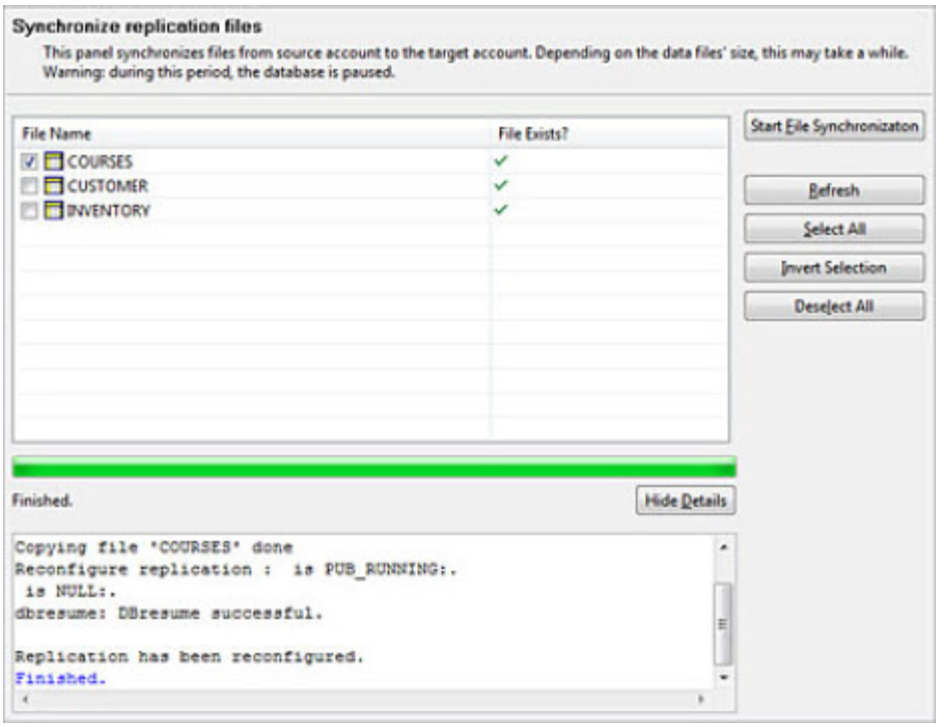
   If you want to overwrite the existing file, select **Overwrite existing files in the target account**.

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

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

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

   When the synchronization is complete, U2 displays a message similar to the following example:

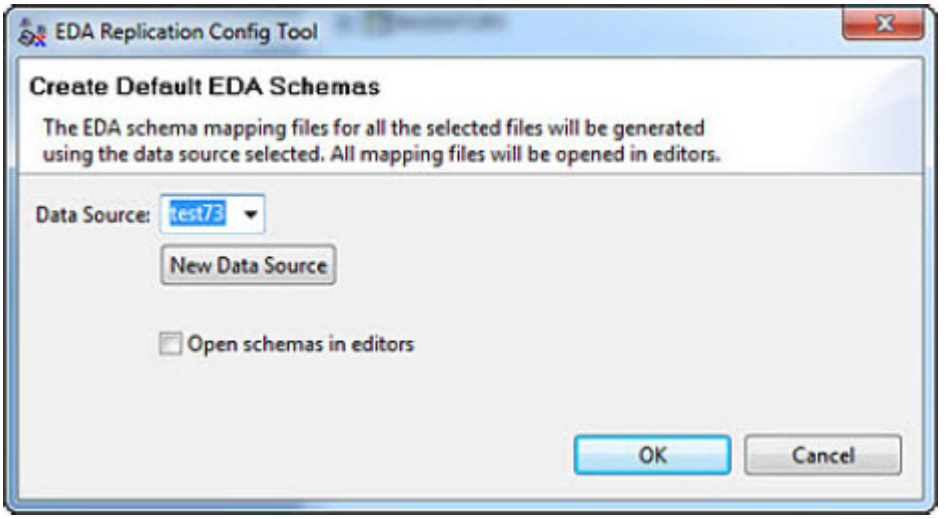


Creating EDA schemas for replicated files

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

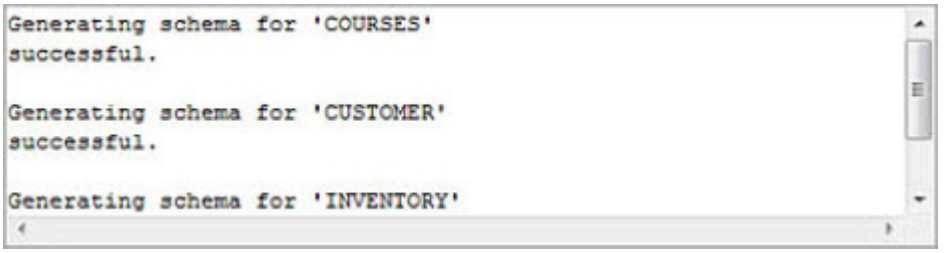
Creating default EDA schemas

To create a default EDA schema, click **Create Default EDA Schemas**. The following dialog box appears:



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

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



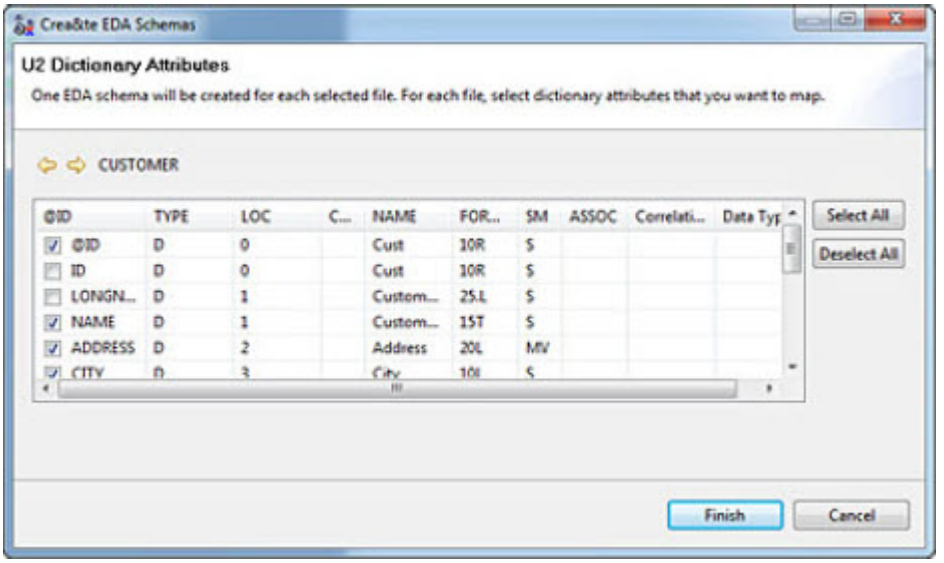
Creating EDA schemas

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

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

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

The following example illustrates the U2 Dictionary Attributes dialog box:

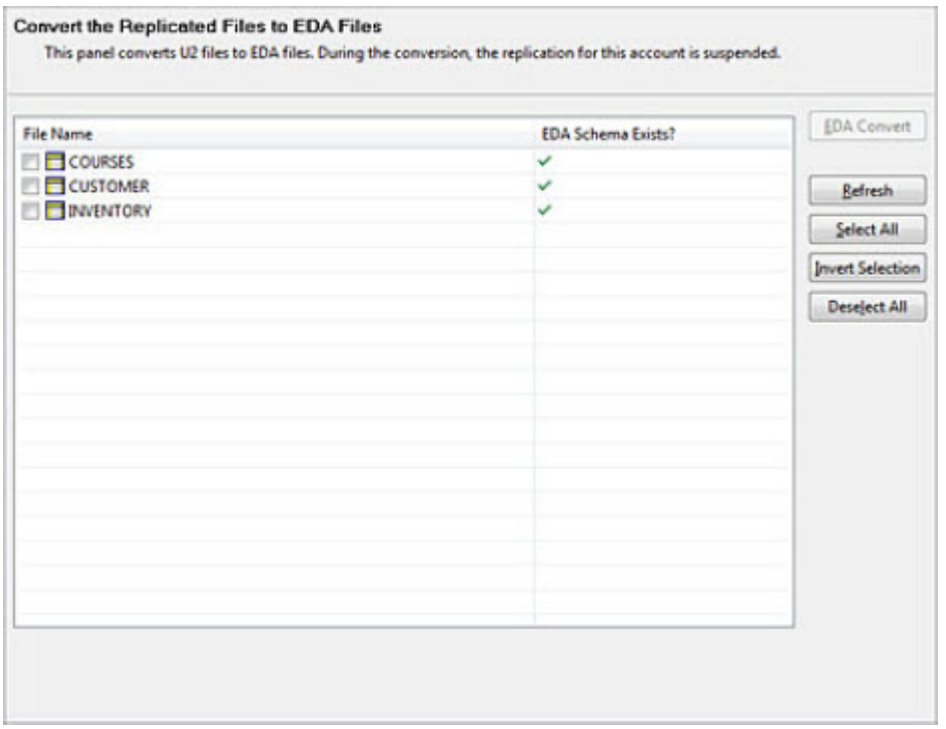


When you have finished selecting the dictionary attributes for which you want to create schemas, click **Finish**.

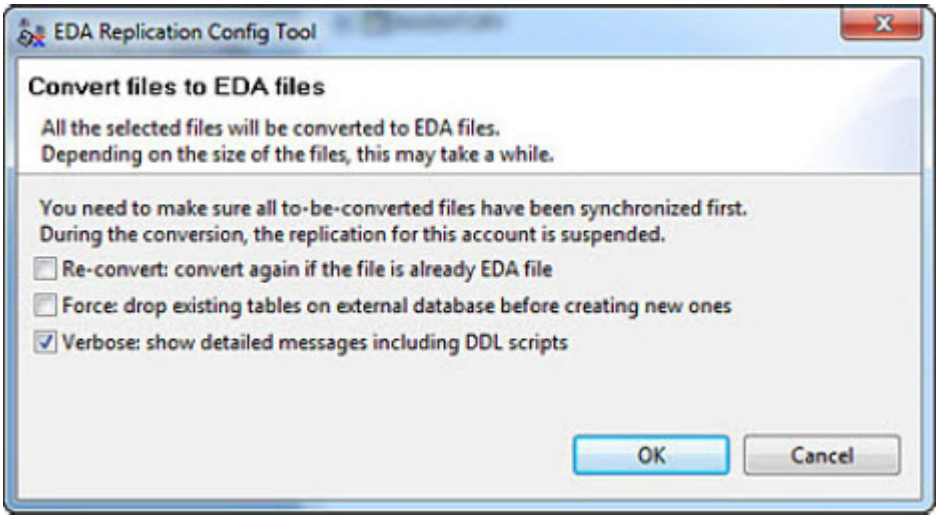
U2 creates the schema files and generates informational messages.

Converting replicated files to EDA files

To convert replicated files to EDA files, click **Convert the Replicated Files to EDA Files**. A dialog box similar to the following example appears:



Select the files you want to convert to EDA. Make sure you have synchronized the files before converting them to EDA. Click **EDA Convert**. The following dialog box appears:



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

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

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

Chapter 8: EDA best practices

UniData provides many options when mapping data from the UniData database to the EDA database. You can choose multiple multivalued and multi-subvalued fields, and virtual fields. Different mapping schemes can result in performance degradation and nonconforming data. This chapter describes guidelines to achieve better EDA performance and fewer errors.

Map selected fields

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

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

Avoid multiple multivalued associations

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

If you need to map multiple associations, use the WHOLE RECORD option to increase performance and avoid external database errors. See Defining options, on page 24 for more information.

Avoid restrictive data types

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

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

```
SELECT * FROM TEST WHERE field1 = "ABC";
```

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

Data types for record IDs

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

Use care when mapping record IDs that have a data type of INTEGER, DATE, or TIME. Because U2 converts these types of data, using OCONV to map the data to the external database and ICONV to use the data in U2, different results may occur. For example, if you have two records with the record IDs of “1” and “1.0” in the U2 database, and you use the TIME data type when mapping to the external database, only one record will be mapped, since each of these record IDs converts to 00:00:10.

RECORD_BLOB

When you specify NONCONFORMING or WHOLE_RECORD, U2 stores unmapped fields or the whole record in a record blob on the external database. Set the size of the record blob to the largest record size in the U2 file to avoid insert failures.

The following table describes the default sizes for RECORD_BLOB.

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

Updating an EDA file from the external database

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

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

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

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

- When inserting a new row, always set NONCONFORMING_FLAG to 0 for that row.
- When updating an existing row, if the NONCONFORMING_FLAG is 0, you may proceed with the update, but do not update a row which has the NONCONFORMING_FLAG set to 1. If you need to update a row with the NONCONFORMING_FLAG set to 1, perform a clean up of the values in this record before attempting an update. See Defining options, on page 24 for more information.