OpenText™ Structured Data Manager

Developers Guide

Version : 26.1

PDF Generated on : February 18, 2026

Table of Contents

1. Developers Guide

This part provides an introduction to the basics of database archiving.

Part one includes:

1.1. Introduction to structured data management

Structured data management is the act of moving or retiring data from an active database to an archive data store. An active database is typically an online transaction processing (OLTP) database and is also sometimes referred to as a production database. An archive data store is another database (sometimes called an archive database) or a set of files (CSV or XML). Moving means copying the specified parts of the active database to the archive data store and deleting it from the active database.

Optimization is desirable for many reasons, but two of the most common ones are:

- **To retain data for the long term while keeping it accessible for purposes of corporate governance, eDiscovery, and/or legal hold.** Database to file movement enables you to move or copy large amounts of data from a database to structured files (XML or comma separated values). XML files provide an open, standards-based format that can be opened long after the original application has been retired. As such, it provides an excellent method of storing data for long term retention. Comma separated values (CSV) files provide a simple format that many applications can import.
- **To reduce the footprint of your active database.** After databases and their associated applications have been in use for some time, they often become very large, which slows performance and increases the need for additional, expensive hardware. Archiving older, infrequently accessed data from the active database enables you to move eligible data to cheaper storage while also reorganizing the active database. This reduction in footprint tends to improve the performance of the active database.

This section includes:

For a complete conceptual introduction to Structured Data Manager, see *OpenText™ Structured Data Manager Concepts Guide*.

1.1.1. Before you begin

Before you begin performing the tasks in this guide, you should:

1. Review the *OpenText™ Structured Data ManagerConcepts Guide* to become familiar with the software and how you plan to use it.
2. Install Structured Data Manager according to the instructions in the *OpenText™ Structured Data Manager Installation Guide*.
3. Go through *OpenText™ Structured Data Manager Tutorial*. The tutorial enables you to get hands on with the product quickly and exposes you to many of the most commonly used features.

1.1.2. Designer overview

Most of the work associated with developing an archive is performed in Designer. Designer is a powerful graphical development environment used to:

- model data
- apply rules
- design cartridges
- design business flows that employ cartridges and implement additional logic
- preview models and cartridges for testing purposes
- deploy cartridges and business flows

The Designer module contained within Structured Data Manager drastically improves the productivity of archive developers, who no longer need to spend hours writing and debugging complex SQL. They simply drag and drop in Designer's editor to include tables and relate them to other tables. Once the model is defined, developers can point and click to define rules on tables as desired.

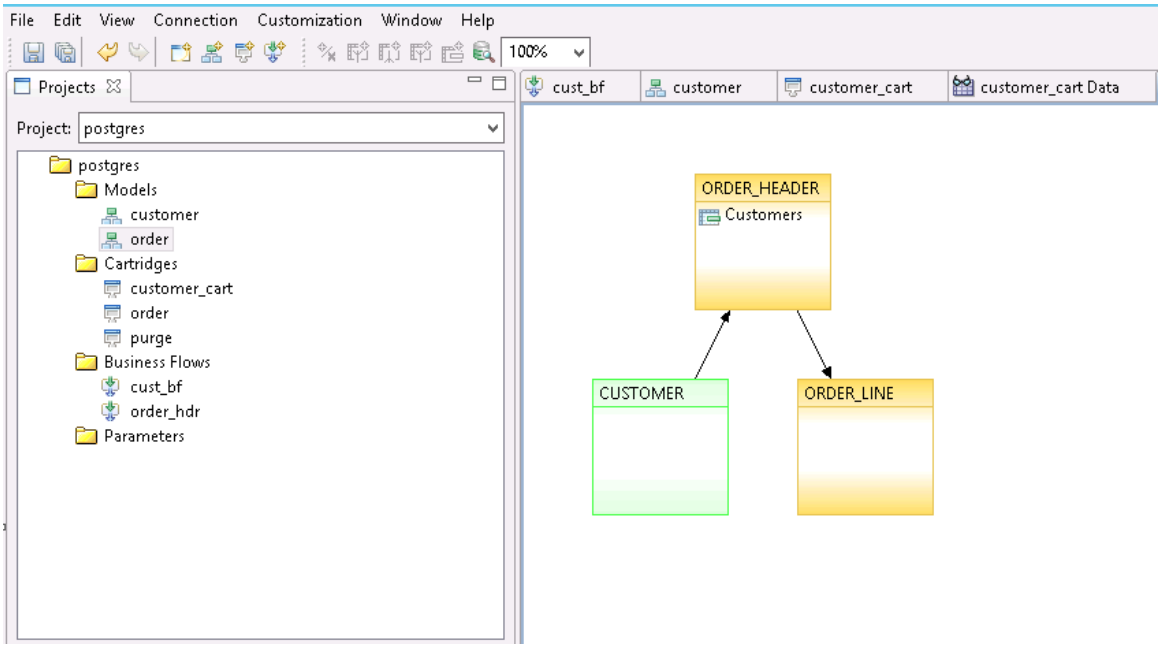
Navigation and interaction with the various components within Designer is consistent, and once familiar with the patterns in one portion of Designer, you should be able to work with any part of Designer.

This section includes:

- Main window
- Projects and connections

Main window

The main window is the primary window you see and interact with while using Designer.



Legend	Name	Description
A	Project Navigator	Shows the files contained within the project.
B	Database Navigator	Shows the content of the database or local cache to which the project is connected.
C	Editor	Workspace used to define model, cartridge, and business flow components, to view preview data, and to view database table data.

Projects and connections

To build an archive solution in Designer, all work is performed within the context of a project. A finished project contains all the metadata definitions needed to archive your data.

Projects help you organize your work for a specific goal.

You can define multiple connections, to databases and to local caches. Each project is associated with one connection.

For more information, see Working with projects and connections.

1.2. Advanced Tutorials

This part provides advanced tutorials that build on the basic tutorial described in the *OpenText™ Structured Data Manager Tutorial*. It includes:

- Access to archived data
- Search for Databases
- Advanced data masking
- Eligibility analytics
- Database to database archive and reload

1.2.1. Access to archived data

After you retire or archive your application data, users may still need to access it, albeit less frequently than when it was in your production system. Structured Data Manager provides a number of mechanisms to access archived data, including AQS and database upload/reload. For lightweight query access to data, data access cartridges provide many advantages. By quickly adding data access rules and defining master-detail records, you can provide users with basic query access to retired data wherever it resides (file or database locations).

This section contains a tutorial that describes how to build a data access cartridge to query retired data stored in files.

This section includes:

1.2.1.1. Before you begin

This advanced tutorial assumes that you are already familiar with the basic concepts and functionality of Structured Data Manager. If not, you should review both the *OpenText™ Structured Data Manager Tutorial* and *OpenText™ Structured Data Manager Concepts Guide* before proceeding with this advanced tutorial.

For the purposes of this tutorial, you can use the same project you created in the *OpenText™ Structured Data Manager Tutorial*. If you did not build that tutorial yourself, you can obtain the solution project as follows:

1. Launch Designer.
2. If you are not prompted to create a new project, select **File > New Project**. Or, you can click the **New Project** icon.
3. In the Name field, type `data_access_tutorial` as the name of your new project.
4. For Database, if you already created a connection to the database with the DEMARC schema, choose that connection from the pull-down list. Otherwise, click **New** to set up such a database connection. If you need to install the DEMARC schema, see *OpenText™ Structured Data Manager Tutorial.*
5. After filling out the New Project dialog box, click **OK**.
6. Select **File > Import**. The Import Existing Project dialog box appears.
7. Browse to the location of the tutorial solution project. On MS Windows, you can find it in `<install dir>\obt\demo\project`. On Unix, you can find it in `<install dir>/obt/demo/project`.
8. Select `tutorial_soln_<db_type>.hdp`, where `<db_type>` is your database type, for example, `oracle` or `sqlserver`.
9. Click **Open**. You should now have a complete, working version of the tutorial solution project, which you can modify as directed in this tutorial.

1.2.1.2. Identify the data

For the purposes of this example, assume that you have spoken with your business users and determined that they require access to archived sales orders based on the following criteria:

- Sales representative. Sales managers want to be able to see all orders for a particular sales representative.
- Customer. Account owners want to be able to orders from a particular customer.
- Product. Product owners want to be able to view orders of a particular product.

Since your business users are not highly technical, you want to provide them basic read access to this data in their Web browsers without the need to resort to SQL. The best way to satisfy this requirement is to build and deploy data access cartridges, and provide your users with a URL to run that cartridge to view the data.

Data access cartridges require a data model. The first step is to identify or create a model that captures the data your users require. In this case, you were archiving the data based on the Orders model from the *OpenText™ Structured Data Manager Tutorial* and can simply modify that model for data access purposes.

If you want to learn how the data model for the tutorial was built, see the *OpenText™ Structured Data Manager Tutorial*.

Note

The model you use for a data access cartridge cannot contain chaining tables.

1.2.1.3. Add data access rules

In most cases, a data model that was designed for archiving requires some modification to optimize it for data access. In this scenario, you need to add parameters and rules that meet the criteria set by your business users. In the *OpenText™ Structured Data Manager Tutorial*, you already added a rule and parameter for selecting orders based on sales representative. Now, you will add the rules and parameters for selecting orders by customer and product.

To add rules for data access to the model

1. Open the tutorial project you created or follow the instructions in Before you begin to obtain and import the appropriate .hdp file for your database.
2. In Designer, open the **Orders** model.
3. Right click CUSTOMER and choose **Add Rule**.
4. Enter `Orders by Customer ID` for the **Name**.
5. Click the **Properties** tab.
6. For Usage, check only **Data Access**.
7. Click the **SQL** tab.
8. For WHERE, enter the following clause:

```
:customer_id = 0
or
"CUSTOMER"."CUSTOMERID" = :customer_id
```

9. Click **OK**.
10. Repeat step 3 through step 9 to create a rule called `Orders by Product ID` on the PRODUCT table with the following WHERE clause:

```
:product_id = 0
or
"PRODUCT"."PRODUCTID" = :product_id
```

Note

Configuration parameters are not supported in rules for data access cartridges. If you reference a configuration parameter from a data access cartridge rule, the cartridge deployment will fail.

1.2.1.4. Create a data access cartridge

With your model ready, you can now create a data access cartridge that leverages your model and the new rules.

To create a data access cartridge

1. Go to **File > New Cartridge**.
2. In the New Cartridge dialog box, type `Orders_Advanced_DA` for the Name.
3. Select **Data Access** from the Type list
4. Choose the **Orders** model if it is not already selected.
5. Click **OK**. The Data Access Cartridge editor displays. Notice the following on the Overview page:

   Your model is **Orders**.

   - **Allow Data Access** is checked by default. This option indicates that you want the data access cartridge to appear under Data Access in the Web Console when you deploy it. Users can run the cartridge from there interactively and see the query results. In cases where you do not want users to run the cartridge directly in this fashion, you would uncheck this option to prevent it.
   - **Allow Free Text Search** is also checked by default. This option indicates that you want to be able to use the data access cartridge to render search results. See Search for Databases.

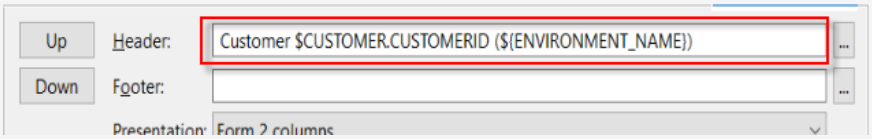
     For the purposes of this tutorial, leave this option checked.
   - **Folder(s)** is empty. This option allows you to organize your archive into named folders, separated by semicolons. For the purposes of this tutorial, leave this field empty.
6. Click **Master Template** at the bottom of the editor. The master template determines what will be displayed to the user for the master record.
7. In the upper left, select the check boxes next to the following columns:
   - ORDER_HEADER.ORDERID
   - CUSTOMER.LASTNAME
   - CUSTOMER.FIRSTNAME
   - STATUS.NAME
   - SALESREP.SALESREPID
   - SALESREP.NAME
8. With the STATUS.NAME still selected, change the **Header** on the right to read `Status`.
9. Click **Detail Template** at the bottom of the editor. The detail template determines what will be displayed to the users when they drill down on an order from the master.

Tip

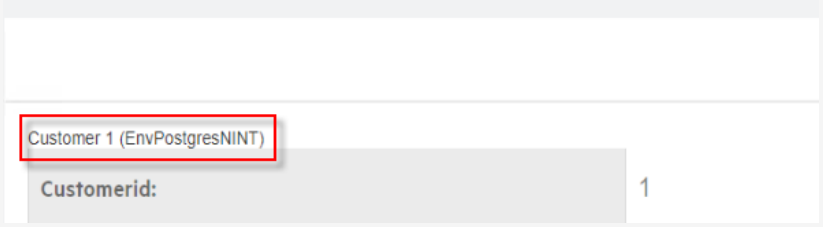
You can customize the DAC detail template. To do this, you need to export the default DAC detail template, modify as per your requirement and import it back. For the detailed steps, refer *Create data access cartridges* section in OpenText™ Structured Data Manager *Tutorial*.

Note

Sometimes it is useful to display the environment to which the DAC is associated. To do so, in DAC, you can use system parameter **ENVIRONMENT_NAME**. Following image gives an idea on the usage of **ENVIRONMENT_NAME** parameter in **Header** of the **Detail Template** tab:



The image below gives an idea on the display of environment in Web Console for the **Header** specified as above in detail template:



10. In the upper left, select the ORDER_LINE table.
11. Select the check boxes next to all of these columns:
    - ORDER_LINE.ORDERID
    - ORDER_LINE.QUANTITY
    - ORDER_LINE.PRICE
    - ORDER_LINE.DISCOUNT
    - PRODUCT.NAME
    - PRODUCT.PRICE
12. Save your work.
13. Click the **Data Masking** tab. The Edit Data Masking page opens and you can apply data masks to particular columns as necessary to protect privacy or hide confidential information, such as Social Security Numbers and credit card numbers. See Apply data masks for data access cartridge.
14. Click **Preview** to confirm that the cartridge is working as expected.
15. Enter a value between 1 and 50 for the `salesrep` parameter in the Parameter Values dialog. The rows for that salesrepid should then appear in black in the preview. When you are satisfied that your cartridge is working correctly, continue to the next step.
16. Deploy the cartridge as you would any other. Follow the instructions in Generating and Deploying. Select the following data locations on the Target page of the Deployment Assistant to union both your source and retired data in the data access cartridge results:

    An AQS location that you created in the Web Console. See the *OpenText™ Structured Data Manager Tutorial* for more information.

    OLTP_DB
17. Go to Web Console.

> **Note**
>
> If you have multi-byte characters in your database, you need to update the `pdf.font` property in `outerbay.properties` file, to include the necessary fonts to display the characters. This is located in the home directory under config file. If you want to specify your own font for multi-byte characters, you need to update `pdf.additional.font.paths` property in `outerbay.properties` file.
>
> For example, **pdf.font=STSong-Light-H** or **pdf.additional.font.paths=C:\\Windows\\Fonts\\msjh.ttc;Identity_H;true**.
>
> For more information about the home directory, see Manage the Home Directory.

18. From the home page of the Web Console, click **Data Access** to see a list of deployed data access cartridges. The Orders_Advanced_DA cartridge should appear in the list.
19. Click **Orders_Advanced_DA** to run it.
20. After you confirm that the deployed cartridge is running as expected, you can let your business users access it. See the *OpenText™ Structured Data Manager Runtime Guide* for more information about creating users and giving them privileges.

Apply data masks for data access cartridge

To create a data mask

1. In the cartridge editor, select the **Data Masking** tab.

   The Edit Data Masking page opens.

2. If you need to apply a mask to a column, expand the table node that contains the column, for example, CUSTOMER.
3. Select the table column, for example, CREDITCARD.
4. Click and select the **Action** type as Mask or Unmask.
5. Choose a value from **Masking Functions** or, if you are using a mask you defined yourself, refer to Advanced data masking.

   For information on pre-built masks, see section Pre-built masks.

   Before you choose your data mask, consider all of the following:

   - You must ensure that the data mask you apply corresponds to the data type of the column. For example, if the column is a numeric value, you must apply a numeric mask to it or use a function to convert the data to a data type that is compatible with the mask.
   - Based on the **Action** type selected, the masking column generates the naming convention as **MASKED_COLUMN-NAME** or **UNMASKED_COLUMN-NAME** .
   - The masking columns are implicitly Groovy columns and have Groovy tt that are editable.

You can view the formatted values as preview in the Master template or Detailed template only for non-DPP (SecureData) masking column functions. The formatted values for custom masking functions can be seen only after you deploy and run the Data Access Cartridge on the WebConsole.

1.2.2. Search for Databases

Structured Data Manager can help you to provide searching for all of your structured data wherever it is stored. By creating and running an indexing cartridge against a model, you can enable a search engine, such as OpenText™ Knowledge Discovery (IDOL), to find and retrieve database records wherever they may be located (source or archive database).

This section contains a tutorial that describes how to build an indexing cartridge to search database records.

For more information about Knowledge Discovery (IDOL), how to configure it, and how to use it, see your Knowledge Discovery (IDOL) documentation.

This section includes:

1.2.2.1. Before you begin

This advanced tutorial assumes that you are already familiar with the basic concepts and functionality of Structured Data Manager. If not, you should review both the *OpenText™ Structured Data Manager Tutorial* and *OpenText™ Structured Data Manager Concepts Guide* before proceeding with this advanced tutorial.

For the purposes of this tutorial, you can use the same project you created in the *OpenText™ Structured Data Manager Tutorial*. If you did not build that tutorial yourself, you can obtain the solution project as follows:

1. Launch Designer.
2. If you are not prompted to create a new project, select **File > New Project**. Or, you can click the New Project icon.
3. In the Name field, type `index_search_tutorial` as the name of your new project.
4. For Database, if you already created a connection to the database with the DEMARC schema, choose that connection from the pull-down list.

   Otherwise, click **New** to set up such a database connection. If you need to install the DEMARC schema, see *OpenText™ Structured Data Manager Tutorial.*
5. After filling out the New Project dialog box, click **OK**.
6. Select **File > Import**. The Import Existing Project dialog box appears.
7. Browse to the location of the tutorial solution project. On MS Windows, you can find it in `<install dir>\obt\demo\project`. On Unix, you can find it in `<install dir>/obt/demo/project`.
8. Select `tutorial_soln_<db_type>.hdp`, where `<db_type>` is your database type, for example, `oracle` or `sqlserver`.
9. Click **Open**. You should now have a complete, working version of the tutorial solution project, which you can modify as directed in this tutorial.

1.2.2.2. Modeling the data

By creating a model, you tell Structured Data Manager about the tables you want to index as well as their relationships to one another. You can then reference this model from the indexing cartridge, which performs the actual indexing of data.

In this case, you are indexing the data based on the Orders model from the *OpenText™ Structured Data Manager Tutorial*. If you want to learn how the data model for the tutorial was built, see *OpenText™ Structured Data Manager Tutorial*.

1.2.2.3. Add indexing rules

You may want to add specific rules for indexing, but they may differ from the rules you use when archiving or reloading. Hence, you can mark which rules you want to use for indexing cartridges. If a rule is not flagged for indexing, it is ignored by any indexing cartridge using that model.

For example, when you are indexing a live, production database, you probably need to perform indexing at regular intervals to account for newly added records. To ensure that you do not unnecessarily re-index records, you could add an indexing rule that only selects records added in the time since the last indexing job run.

In the *OpenText™ Structured Data Manager Tutorial*, you already added a rule that puts a time bound around the selected records. This rule does not exactly match what you might do for an indexing rule, but it will suffice for illustrative purposes. Flag this rule for use as an indexing rule.

To mark a rule for use in indexing cartridges

1. Open the tutorial project you created or follow the instructions in Before you begin to obtain and import the appropriate .hdp file for your database.
2. In Designer, open the **Orders** model.
3. Right click the `Orders Shipped N Months Ago` rule in the ORDER_HEADER table and choose **Edit Rule**.
4. Click the **Properties** tab.
5. Under Usage, check **Indexing**.
6. Click **OK**.

1.2.2.4. Create a data access cartridge

Search uses data access cartridges to retrieve and display the data for any search hits on database records. Hence, when you create an indexing cartridge, it often makes sense to prepare an associated data access cartridge.

> **Note**
>
> Data access cartridges can be used independently of indexing cartridges. See Access to Archived Data. Furthermore, you can run an indexing cartridge without a data access cartridge, but, in this case, you may receive errors when a search returns results that include database records but cannot find a data access cartridge to render them.

To create a data access cartridge

1. Go to **File > New Cartridge**.
2. In the New Cartridge dialog box, type `Orders_Index_DA` for the Name.
3. Select **Data Access** from the Type list
4. Choose the **Orders** model if it is not already selected.
5. Click **OK**. The Data Access Cartridge editor displays. Notice the following on the Overview page:
   - Your model is **Orders**.
   - **Allow Data Access** is checked by default. See Access to archived data for more information.

     For the purposes of this tutorial, leave this option checked.
   - **Allow Free Text Search** is also checked by default. This option indicates that search can invoke the cartridge for rendering its results.

     This option must remain checked for this tutorial.
6. For **Selection Rules**, only leave **Orders by Customer Name** and **Orders by Product Name** checked. Uncheck all other rules.
7. Click **Master Template** at the bottom of the editor. The master template determines what will be displayed to the user for the master record.

   > **Tip**
   >
   > When called from search, the data access cartridge bypasses the master template and uses the detail template to render the referenced records. However, a master template is required when running the data access cartridge independently.
8. In the upper left, select the check boxes next to the following columns:
   - ORDER_HEADER.ORDERID
   - ORDER_HEADER.ORDERDATE
   - SALESREP.NAME
9. To add a new pseudo column populated by a Groovy script, select ORDER_HEADER.ORDERID and click **Add**.
10. Enter `Customer` for the **Name**.
11. Enter the following for tt:

```
import org.apache.commons.lang.StringEscapeUtils

"""

<a href="mailto:${CUSTOMER.EMAIL}">

${StringEscapeUtils.escapeHtml(CUSTOMER.FIRSTNAME)}
${StringEscapeUtils.escapeHtml(CUSTOMER.LASTNAME)}

</a>
```

```
"""
```

12. Click **OK**.
13. With the new Customer column selected, check Allow HTML formatting to resolve the HTML tt you entered for the Customer column.

> **Note**
>
> By default, the cartridge applies `escapeHTML` to the text you enter for tt. Otherwise, special characters could inadvertently cause incorrect rendering. For example, if a string contained <B> and HTML formatting was on by default, everything that follows would become bold.
>
> Even worse, a stray angle bracket could cause data to disappear. For example, if a string contained < rather than &lt, then the data after it would be interpreted as an HTML instruction and not appear in the output at all.

14. Click **Detail Template** at the bottom of the editor. The detail template determines what will be displayed to the users when they drill down on an order from the master.
15. In the upper left, select the ORDER_HEADER table.
16. Update the **Header** field to read `Order`.
17. Check **Border** and **Indent**.
18. Select the check boxes next to all of these columns:
    - ORDER_HEADER.ORDERID
    - ORDER_HEADER.ORDERDATE
    - ORDER_HEADER.SHIPDATE
19. Add a new pseudo column below ORDER_HEADER.ORDERID populated by a Groovy script, named `Customer`, with the following tt:

```
import org.apache.commons.lang.StringEscapeUtils

"""

<a href="mailto:${CUSTOMER.EMAIL}">

${StringEscapeUtils.escapeHtml(CUSTOMER.FIRSTNAME)}
${StringEscapeUtils.escapeHtml(CUSTOMER.LASTNAME)}

</a>

"""
```

20. Add a new pseudo column below ORDER_HEADER.SHIPDATE populated by a Groovy script, named Total, with the following tt:

```
'$' + ORDER_HEADER.TOTAL
```

21. Add a new pseudo column below Total populated by a Groovy script, named Discount, with the following tt:

```
List<String> priceValues = ORDER_LINE.PRICE

List<Integer> prices = priceValues.collect{it as Integer}

int undiscountedTotal = prices.sum()

int paidPrice = ORDER_HEADER.TOTAL as int

int discount = undiscountedTotal - paidPrice

return '$' + discount
```

22. Check **Allow HTML formatting** for each of the newly created columns.
23. In the upper left, select the ORDER_LINE table.

24. Update the Header field to read Details.
25. Check **Border** and **Indent**.
26. Select the check boxes next to all of these columns:

    Tip

    To quickly find the column, type QUA in Filter.

    - ORDER_LINE.QUANTITY
    - ORDER_LINE.PRICE
    - PRODUCT.NOTE
27. Add a new pseudo column above ORDER_LINE.ORDERLINEID populated by a Groovy script, named Product, with the following tt:

    ```
    """
    <a href="http://www.google.com/search?q=${PRODUCT.NAME}&btnI"
    target="_blank">
    ${PRODUCT.NAME}
    </a>
    """
    ```
28. With the new column selected, check **Allow HTML formatting**.
29. Click the **Data Source** tab.
30. For the CUSTOMER, PRODUCT, SALESREP tables, change the Data Source to **Source**.
31. Deploy the cartridge as you would any other. Follow the instructions in Generating and deploying. Select OLTP_DB on the Target page of the Deployment Assistant.

    Tip

    For indexing and data access cartridges, deploying to a non-intrusive environment is preferred because it does not require write access to the database.

    Note

    In this case, records are only stored in the source database, but, if some records were archived, you could choose multiple targets from which the data access cartridge could union the query results.

1.2.2.5. Create an indexing cartridge

With your model and data access cartridge now ready, you can create an indexing cartridge that leverages them.

To create an indexing cartridge

1. Go to **File > New Cartridge**.
2. In the New Cartridge dialog box, type `Orders_Index_Search` for the Name.
3. Select **Indexing** from the Type list
4. Choose the **Orders** model if it is not already selected.
5. Click **OK**. The Indexing Cartridge editor displays. Notice that your model is Orders and that only the `Orders Shipped N Months Ago` rule is selected because it was the rule we marked for indexing. All other rules are ignored by this cartridge.

   If using Knowledge Discovery (IDOL) and its database is the same as the one specified in your Web Console settings, you need not specify anything for **Knowledge Discovery (IDOL) Database**. Otherwise, enter the database name here. See Run the indexing cartridge.
6. For **Data Access**, select **Orders_Index_DA**, which you created in Create a data access cartridge. Knowledge Discovery (IDOL) uses a data access cartridge to retrieve and display the data for any hits from an Knowledge Discovery (IDOL) search.
7. Click the **Document Title** tab. When the user performs a search, the tt you enter on this tab determines what to display as the first line of any search hits on the indexed data from your database.

   Tip

   Document Title and Document Content both expect Groovy tt as their input.
8. For tt, enter the following:

```
"""
Order ${ORDER_HEADER.ORDERID} for ${ORDER_HEADER.CUSTOMER.FIRSTNAME}
${ORDER_HEADER.CUSTOMER.LASTNAME}
"""
```

9. Click the **Document Content** tab. When the user performs an Knowledge Discovery (IDOL) search, the tt you enter on this tab determines the more detailed content to display following the title for any search hits on the indexed data from your database.
10. For tt, enter the following:

```
"""
Products:
${ORDER_HEADER.ORDER_LINE.PRODUCT.collect {" ${it.NAME}\n"}.sum()}
Stores:
${ORDER_HEADER.ORDER_LINE.ORDER_LINE_DIST.STORE.collect {" ${it.NAME}\n"}.sum()}
"""
```

11. Click the **Indexed Columns** tab, where you can identify the columns you want to index. You should take care when selecting which columns to index. For example, it is not recommended to index columns that contain sensitive data, such as credit card numbers or phone numbers or Social Security numbers.

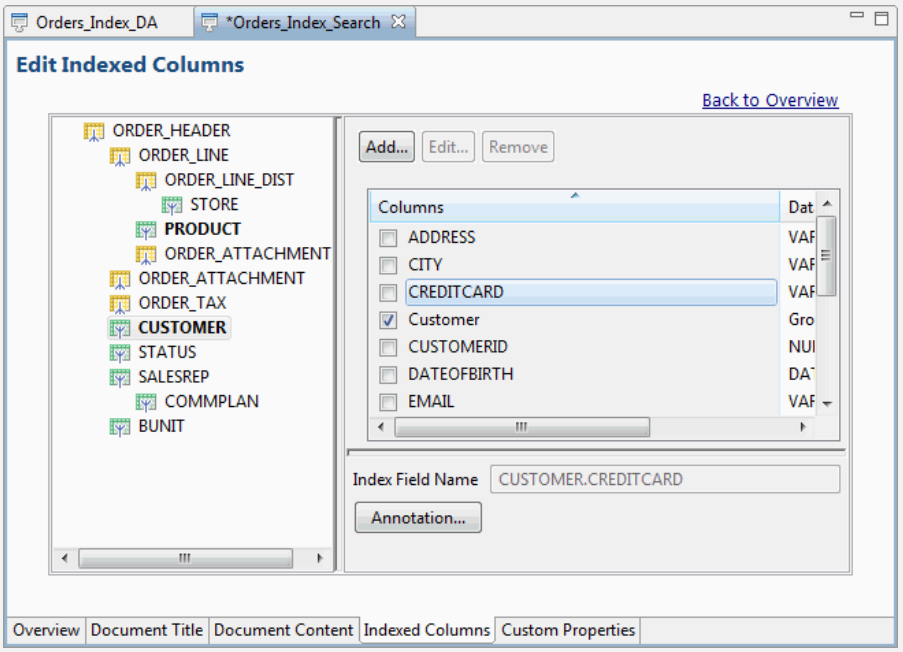
    

    Note

    When using Knowledge Discovery (IDOL), indexed columns invoke parametric search.
12. Select the **PRODUCT** table and check the **NAME** column.
13. Change the **Index Field Name** to `Product`.

14. Select the **CUSTOMER** table.
15. With the CREDITCARD column selected, click **Add** to create a column populated by Groovy.
16. Enter `Customer` for the **Name**.
17. Enter the following for tt:

```
"FIRSTNAME + ' ' + LASTNAME"
```



18. Deploy the indexing cartridge as you would any other. Follow the instructions in Generating and Deploying.

> **Tip**
>
> For both indexing and data access cartridges, deploying to a non-intrusive environment is preferred because it does not require write access to the database.

1.2.2.6. Run the indexing cartridge

After you have deployed the cartridges, you can run the indexing cartridge to enable users to perform searches that include your structured data.
From Structured Data Manager 7.53 onwards, you can send the index to **Content Manager** by setting the destination location as **CM**. This destination location can be CM or a file system or any other location type where a file can be written.

> **Tip**
>
> Instead of running the indexing cartridge by itself, you can associate it with a database to file cartridge by choosing it from the Indexing list of values in the database to file cartridge editor. When you run the database to file cartridge, the indexing cartridge will run, too.

1. Go to the **Web Console > Settings**.
2. Click **Indexing Server**. You need to specify the connection details for the server running Knowledge Discovery (IDOL).
3. Enter the valid details for your Knowledge Discovery (IDOL) server, including the protocol you want to use to communicate with it. See the *Knowledge Discovery (IDOL)Administration Guide* for more information about HTTPS and Knowledge Discovery (IDOL).

   > **Note**
   >
   > The Knowledge Discovery (IDOL) database name that you enter should already exist on the specified Knowledge Discovery (IDOL) server. If it does not, the indexing jobs will run successfully but the data sent to Knowledge Discovery (IDOL)is ignored.

   > **Tip**
   >
   > If you enter a Server Host of 999.999.999.999, the indexing job will fail and the .idx file will be available in `<install_dir>/temp/environment_name>/*.idx`.

4. If you have configured TEA encryption, you can enable it here by checking **Enabled** and entering the key in the provided field.

   TEA encryption is configured in `AutonomyIDOLServer.cfg`. For example:

   ```
   ACIEncryption]
   CommsEncryptionTEAKeys=1111,2222,3333,4444
   CommsAllowUnencrypted=false
   CommsEncryptionType=TEA
   ```

5. Optionally, you can change the Security by Database setting as well:

   Allow users with data access privileges to query any IDOL database means that any user who has the data access privilege granted to them can also query any Knowledge Discovery (IDOL) database. See the Web Console online help for more information on privileges.

   Grant users IDOL database query access means that you must explicitly grant users query access to Knowledge Discovery (IDOL) databases. If you choose this option, when you select **Apply**, a new link appears called **Set security by database**. Following the link takes you to a page where you can manage query access by Knowledge Discovery (IDOL) database and user.

6. Click **Apply**. When it says the settings are saved, you are ready to run your cartridge.
7. Click **Launch**. You should see your indexing business flow, `Orders_IDOL_Search_BF`.
8. Click **Orders_IDOL_Search_BF** to run it.
9. Click **Run**. It may take a few minutes for your business flow to run.
10. Go to the Web Console home page by clicking the logo in the upper left corner.
11. Click **Free Text search**.

12. Search for a customer or product name, for example, `Varian`. A set of results should now appear. Notice how the results display according to what you specified in Document Title of your indexing cartridge.
13. Click one of the results. This action causes your data access cartridge to retrieve and display the record from the database. Notice how the record displays according to what you specified in the Detail Template of your data access cartridge.
14. For other methods of employing Knowledge Discovery (IDOL) search, see your Knowledge Discovery (IDOL) documentation.
15. If the destination location is set as **CM**, then the **IDX** file is sent to **CM** and the index is available under the preview or rendition tab.

1.2.3. Advanced data masking

Data masks enable you to obfuscate column values in your archived data as a means of protecting private, confidential information such as credit card or social security numbers. Structured Data Manager provides pre-built masks for data that most commonly requires protection. In addition, you can create your own custom masks for data that you want to mask.

Before creating and applying a data mask, consider the following:

- The type of data masking you choose for archiving has implications for reloading data. If you choose a type of masking that cannot be reversed, then that data cannot be reloaded. You need to consider whether you expect to reverse masking upon reload when you choose the masking type. See Data masking and reload/undo in Apply data masks.
- Masking of primary keys is supported for copy jobs (in database to file cartridge) in non-intrusive environments only. If you choose to mask a primary key, you should always choose a reversible mask. Otherwise, when you reload the data into a database, your primary key values are lost and your table may no longer behave as expected.
- You must ensure that the data mask you apply corresponds to the data type of the column. For example, if the column is a numeric value, you must apply a numeric mask to it or use a function to convert the data to a data type that is compatible with the mask.
- If you apply a mask on a primary or foreign key, then you should also apply the same mask on a referenced primary key to ensure database integrity.
- By default, the SDM offers a limited list of out-of-the-box masking functions. In addition to this, you can create your own masking functions.
- If you have Voltage SecureData product then you can integrate SDM with it and generate SecureData masking functions.

See Working with Cartridges for detailed descriptions of all the available data masks.

This section contains a tutorial that walks you through an example of how to use data masking to protect your data.

This chapter includes:

1.2.3.1. Before you begin

This advanced tutorial assumes that you are already familiar with the basic concepts and functionality of Structured Data Manager. If not, you should review both the *OpenText™ Structured Data Manager Tutorial* and *OpenText™ Structured Data Manager Concepts Guide* before proceeding with this advanced tutorial.

For the purposes of this tutorial, you can use the same project you created in the *OpenText™ Structured Data Manager Tutorial*. If you did not build that tutorial yourself, you can obtain the solution project as follows:

1. Launch Designer.
2. If you are not prompted to create a new project, select **File > New Project**. Or, you can click the **New Project** icon.
3. In the **Name** field, enter `masking_tutorial` as the name of your new project.
4. For **Database**, if you already created a connection to a database with the DEMARC schema, choose that connection from the list.

   Otherwise, click **New** to set up such a database connection. If you need to install the DEMARC schema, see the *OpenText™ Structured Data ManagerTutorial.*
5. After filling out the New Project dialog box, click **OK**.
6. Select **File > Import**.

   The Import Existing Project dialog box appears.
7. Browse to the location of the tutorial solution project.

   On MS Windows, it is located in `<install dir>\obt\demo\project`.

   On Unix, it is located in `<install dir>/obt/demo/project`.
8. Select `tutorial_soln_<db_type>.hdp`, where `<db_type>` is your database type, for example, `oracle` or `sqlserver`.
9. Click **Open**.

   You should now have a complete, working version of the tutorial solution project, which you can modify using the instructions in this chapter.

1.2.3.2. Apply pre-built data masks

In the basic tutorial, you masked the SOCIAL and CREDITCARD columns of the CUSTOMER TABLE. However, with escalating concerns about confidentiality of customer data, that might not be considered sufficient. You might also want to mask columns such as LASTNAME, PHONE, ADDRESS, and EMAIL to further protect the identities of your customers.

Masking all these columns requires masks that can handle strings. For this purpose, Structured Data Manager includes pre-built string masks.

To mask data using a pre-built string mask

1. Go to the cartridge editor for the **Orders_D2F** cartridge.
2. Click the **Data Masking** tab.
3. Select the **CUSTOMER** table on the left and expand it.

   At a minimum, you need to mask last names, addresses, phone numbers, email addresses, credit card numbers, and Social Security Numbers.
4. Select the **LASTNAME** column.
5. In the **Mask Type** list of values, choose **String mask: random length**.

   This option generate strings of random lengths using the character X (or whatever character you specify).
6. For **Max Length**, enter `30`. This specifies that the random string length will not exceed 30 characters.
    1. Using the information in Table 1, specify masks for the other columns in the CUSTOMER table.

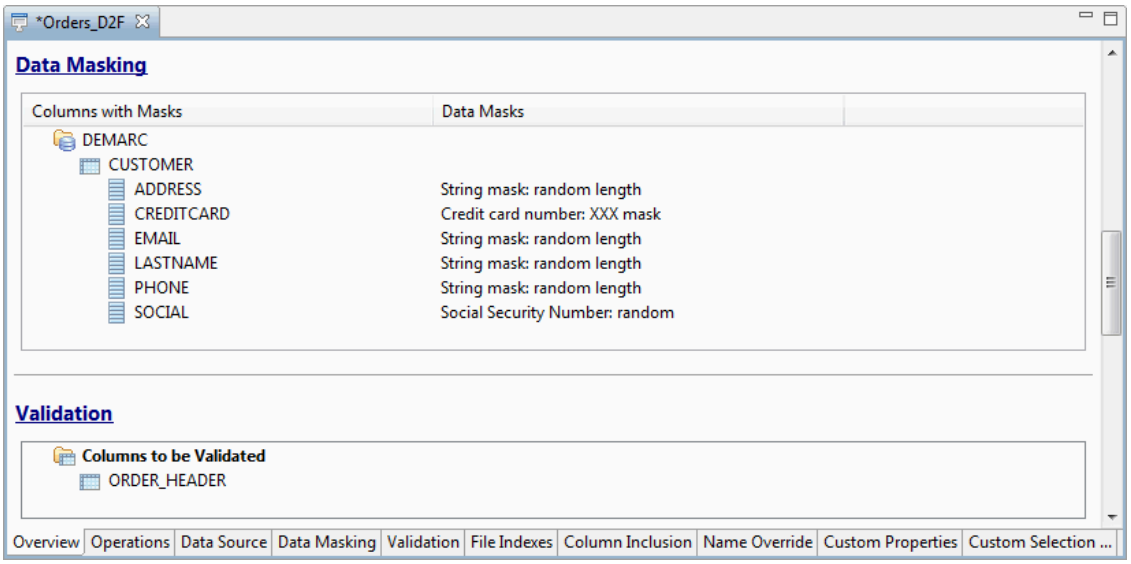
       Tip

       You must ensure that the mask you choose for any column matches the data type of that column. For example, if the column is VARCHAR2(80), use a string mask with a length that is within 80 characters. You can double-click the table in the Database Navigator in the lower left pane of the Designer window to check the characteristics of the columns.

       Column masking information

Column name	Mast
PHONE	String mask: random length Max. Length: 20
EMAIL	String mask: random length Max. Length: 40
ADDRESS	String mask: random length Max. Length: 60

7. Click the **Overview** tab.
8. Scroll down to the Data Masking section to view a summary of the mask.


*Orders_D2F
Data Masking
Columns with Masks
Data Masks
DEMARC
CUSTOMER
ADDRESS
String mask: random length
CREDITCARD
Credit card number: XXX mask
EMAIL
String mask: random length
LASTNAME
String mask: random length
PHONE
String mask: random length
SOCIAL
Social Security Number: random
Validation
Columns to be Validated
ORDER_HEADER
Overview
Operations
Data Source
Data Masking
Validation
File Indexes
Column Inclusion
Name Override
Custom Properties
Custom Selection ...

1.2.3.3. Apply string map masks to data

Random string masks work well for obfuscating data, but you cannot undo these random string masks upon reload or upload. In other words, when you put the data back into a database, the masked value is retained rather than the real value.

For cases where you must be able to undo the masking upon reload or upload, you can create a string map. String map data masks look up a given string in a map and apply the mapping value. When the data is reloaded or uploaded, Structured Data Manager can find the original value in the map and put that value rather than masked value into the database.

String map masks are reversible; hence, the mapping must be one-to-one and complete.

1.2.3.4. Apply numeric map masks to data

Numeric data masks are applicable to numeric columns (such as NUMERIC, INT, REAL, FLOAT, DECIMALS, etc.). Just as with string map data masks, numeric map data masks look up a given number in a map and apply the mapping value. When the data is reloaded or uploaded, Structured Data Manager can find the original value in the map and put that value rather than masked value into the database.

Numeric map masks are reversible; hence, the mapping must be one-to-one and complete.

1.2.3.5. Apply string and numeric map masks

String and numeric map masks require a mapping file. You need to provide absolute path of the file in mapping file parameter.

To apply a string or numeric mask

1. Create a mapping file that defines a map using the syntax: `key=value` per line, where `key` is mapped to value by this map mask. If a key or value contains spaces, you must enclose them in them in quotation marks (for example, `Jack="Male Customer 1"`). In this case, we are masking the FIRSTNAME column in the CUSTOMER table:

   ```
   Jack=MaleCustomer1
   Paul=MaleCustomer2
   Phillip=MaleCustomer3
   Nadine=FemaleCustomer4
   Audrey=FemaleCustomer5
   ...
   ```

2. Save the mapping file to the `obt\extensions\runtime\masking` directory (or any folder accessible to a web console user).
3. In the cartridge editor, click the **Data Masking** tab.

   The Edit Data Masking dialog opens.
4. To apply the string map or numeric map mask to a column, expand the node that contains the table you want to mask.

   For example, CUSTOMER.
5. Select the table column, FIRSTNAME.
6. For **Mask Type**, select either String map or Numeric map.

   In the Mapping file field, type the full path name of the mapping file you created in step 1. In this case, the numeric mapping file name is `StringNonintrusive.txt`

   > **Note**
   >
   > If the mapping file is located in the default folder (obt\extensions\runtime\masking), then you can simply specify the file name (in this case, `StringNonintrusive.txt`) in the Map table name field. However, if the mapping file is located in a different folder, for example, `C:\Users\dba user\Documents\DataMasking\mapping`, then you must specify the full path, which in this case would be: `C:\Users\dba user\Documents\DataMasking\mapping`.
7. Deploy the cartridge and launch the business flow.
8. Confirm the mask was applied by viewing the results in the home directory under `archivedata`:

```
<CUSTOMER>
<CUSTOMERID>1</CUSTOMERID>
<LASTNAME>Jones</LASTNAME>
<FIRSTNAME>MaleCustomer1</FIRSTNAME>
<PHONE>111-222-3333</PHONE>
<EMAIL>JonesJ202@msn.com</EMAIL>
<ADDRESS>123 Main</ADDRESS>
<CITY>Aberdeen</CITY>
<STATE>CA</STATE>
<ZIP>10001</ZIP>
<GENDER>Male</GENDER>
<DATEOFBIRTH>1984-02-12T00:00:00.000000000-08:00</DATEOFBIRTH>
<CREDITCARD>4588347116299261</CREDITCARD> <SOCIAL>897-01-2221</SOCIAL>
<STATUSID>14</STATUSID>
</CUSTOMER>
...
```

1.2.3.6. Create custom data masks

In situations where a pre-built mask simply cannot satisfy your business requirements, or you require specific, conditional logic to apply a data mask, custom data masking is available. The custom masking functions you want to introduce may or may not require user inputs at design time. If there is no user input required then you can use a simple approach of creating Groovy script as below. There are certain limitations in this simple approach such as function name is same as the file name given by you. If you have function parameters or need better control on function names, id etc. then you can use the elaborate approach described in Appendix B: Custom Data Masks.

To create a custom mask using a Groovy script

1. Create a Groovy script that includes the following signature:

   `Object mask(Object param);`
   Mandatory. Use to apply a data mask data during archive.

   `Object revert_mask(Object param);`
   Optional. Use to recover masked data and restore the original values during upload and reload. If you wish to restore data to its original state, then you must provide this function. Otherwise, masked values from the archive are copied as-is during upload and reload.

   Because data types are mapped internally and vary amongst the supported database platforms, it is recommended that you include input parameters for the respective data types. For example, if the column being masked is of data type Decimal, Integer or SmallInt, then the object that is passed to these functions is of the type java.math.BigDecimal.

   Additionally, the SDM passes the following context information to custom groovy functions. This helps you to change the data of a given column based on the other column values.

   - `tableName`: Name of the table on which the `mask/revert_mask` method is applied.
   - `colName`: Name of the column for the field on which the `mask/revert_mask` method is applied.
   - `row`: Details of the row on which `mask/revert_mask` method is applied. It is a LinkedHashMap of key-value pair of Column Name and DBParam object.

   For more information, refer to PrefixLastName.groovy for a sample tt on the use of context variables.

   Note

   Ensure that you do not use any other methods other than toString of DBParam class to fetch the value of the given column as SDM might change the DBParam class in future.

   Note

   Review the sample Groovy scripts in <install_dir>/obt/extensions/runtime/masking for additional guidance.

2. After verifying that the function compiles successfully, copy the Groovy script into the folder or any sub-folder you wish to have under:

   `${obt.dir.extensions}/runtime/masking`

   where

   `${obt.dir.extensions}` is defined in `directories.properties`, for example:

   `obt.dir.extensions=${OBT_HOME}/extensions`

   Suppose that your Groovy script name is `MyMask.groovy` and your OBT_HOME is `C:\SDM\SDM760\obt`. Copy `MyMask.groovy` to `C:\SDM\SDM760\extensions\runtime\masking`. Then add the name of the custom mask (`MyMask.groovy`) on the Data Masking tab of the cartridge and apply it to columns.

3. In Designer, click the Data Masking tab in the cartridge and then click **Refresh** button. You can also use **Import** button on **Data Masking** tab in the cartridge to copy to the `${obt.dir.extensions}/runtime/masking` folder. In this case you don't need to use **Refresh** button.

4. The deployment process will automatically pick up the unmasking function based on the masks that were applied during archiving. You need not specify the unmasking function in the cartridge.

   If you provided an unmasking function as described in step 1, then upload jobs automatically pick up the unmasking function and apply it during upload when the parameter Unmask data on upload is set to true. See the *OpenText™ Structured Data Manager Runtime Guide* for additional details about the Unmask data on upload parameter.

   See the following table for data type mappings from databases to non-intrusive environments and vice versa.

Column data type mappings

SQL-based column type	Java-based column type
VARCHAR For Oracle, this data type is VARCHAR2. , NVARCHAR, CHAR, NCHAR	java.lang.String
VARBINARY, BINARY	java.lang.Byte[]
BIGINT, NUMERIC For Oracle, this data type is NUMBER. , DECIMAL	java.math.BigDecimal
INTEGER, SMALLINT, TINYINT	java.lang.Integer
REAL, DOUBLE	java.lang.Double
FLOAT	java.lang.Float
BIT	java.lang.Short
TIME	java.sql.Time
DATE	java.sql.Date
TIMESTAMP	java.sql.Timestamp

1.2.3.7. Extract binary zero in character strings

Oracle and DB2 databases creates junk characters when there are binary zeros (CHR(0)) in character strings. Columns containing these character strings can be extracted using Structured Data Manager, but this fails when AQS cache is refreshed.

Below is the example of a table with binary zero in the data:

TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	COLUMN_SIZE
TESTZEROBIN	A	1	CHAR	10
TESTZEROBIN	B	1	CHAR	10
TESTZEROBIN	C	1	CHAR	10

In dbvisualizer, the data appears as :

*	A	B	C
1	a□□□□	b□	□□□□
2	a□□□□	b□	□□□□
3	a□□□□	b□	□□□□

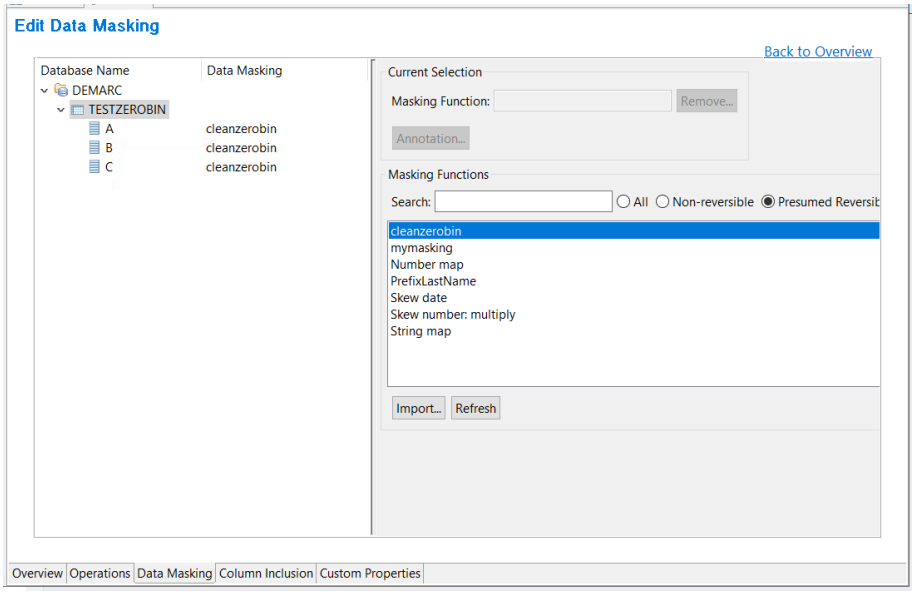
In order to remove the junk characters, create SDM masking function.

To create SDM masking function

1. In the SDM home directory `${home}\extensions\runtime\masking`, create a Groovy file **cleanzerobin.groovy**. Add the following content:

```
Object mask(Object _in) {
 return _in.replaceAll("[\\x00-\\x00]"," ");
}
Object revert_mask(Object _in){
 return _in;
}
```

2. Use **Refresh** button on **Data Masking** tab to make this function available to your cartridge.
3. Apply **cleanzerobin** masking function to remove the unnecessary columns.



4. When you run the business flow with AQS cache enabled, all the binary zeros are replaced with space.

   In dbvisualizer the data in AQS cache appears as:

*	A	B	C	JOB_RUN_ID
1	a	b		375
2	a	b		375
3	a	b		375

1.2.4. Eligibility analytics

When you run a business flow with one or more model-based cartridges, you may find it helpful to review the eligibility analytics. Eligibility analytics show you whether a record was selected for movement and, for those records not selected, which rule caused them to be excluded.

This section contains a tutorial that walks you through an example of how to enable eligibility analytics on an existing cartridge and business flow.

This section includes:

1.2.4.1. Before you begin

This advanced tutorial assumes that you are already familiar with the basic concepts and functionality of Structured Data Manager. If not, you should review both *OpenText™ Structured Data Manager Tutorial* and *OpenText™ Structured Data Manager Concepts Guide* before proceeding with this advanced tutorial.

For the purposes of this tutorial, you can use the same project you created in the *tutorial*. If you did not build this tutorial yourself, you can obtain the tutorial solution project as follows:

1. Launch Designer.
2. If you are not prompted to create a new project, select **File > New Project**. Or, you can click the New Project icon.
3. In the **Name** field, enter `EA_tutorial` as the name of your new project.
4. For Database, if you already created a connection to the database with DEMARC schema, choose that connection from the list.

   Otherwise, click **New** to set up such a database connection. If you need to install the DEMARC schema, see the *OpenText™ Structured Data Manager Tutorial.*
5. After filling out the New Project dialog box, click **OK**.
6. Select **File > Import**.

   The Import Existing Project dialog box appears.
7. Browse to the location of the tutorial solution project.

   On MS Windows, it is located in `<install dir>\obt\demo\project`.

   On Unix, it is located in `<install dir>/obt/demo/project`.
8. Select `tutorial_soln_<db_type>.hdp`, where `<db_type>` is your database type, for example, `oracle` or `sqlserver`.
9. Click **Open**.

   You should now have a complete, working version of the tutorial solution project, which you can modify using the instructions in this chapter.

1.2.4.2. Enable eligibility analytics

When you use preview in Designer and select **Eligibility Analytics for all rules**, eligibility analytics information displays along with the data in the preview. At runtime, you can also generate eligibility analytics and review them just before running the business flow against your production database.

Eligibility analytics is especially useful when you archive for the first time so that you can view the number of rows that are moved. Eligibility analytics can also help when you want to review the reasons why specific data has been excluded. For example, when you enable eligibility analytics on the Product Not Recalled rule, after you archive you can review the number of orders that were not picked up for the archive due to this particular rule.

Of course, if a large percentage of your data is excluded because it is ineligible, then eligibility analytics can seem somewhat inefficient. OpenText recommends that you use eligibility analytics when you anticipate only a small percentage (from 1% to 5%) of your data to be ineligible.

Note

Eligibility analytics are not supported in non-intrusive environments. See the *OpenText™ Structured Data Manager Runtime Guide* for more details about managing non-intrusive environments.

To enable eligibility analytics

1. Open the tutorial project you created or follow the instructions in Before You Begin to obtain and import the appropriate .hdp file for your database.
2. In Designer, open the **Orders** model by double-clicking it in the Project Navigator.
3. Double-click the **Orders Shipped N Months Ago** rule on the ORDER_HEADER table.
4. Click the **Properties** tab.
5. If it is not already checked, select **Eligibility Analytics**, then click **OK**.
6. Repeat step 3 through step 5 for the **Product Not Recalled** rule on the PRODUCT table and the **Order Closed** rule on the STATUS table.

After enabling eligibility analytics, you can access the results for each business flow (using the Web Console) to help you decide whether or not to proceed with an archive operation, to guide you as you debug a cartridge, or to confirm that the cartridge behaves as expected.

1.2.4.3. Add a pause to a business flow

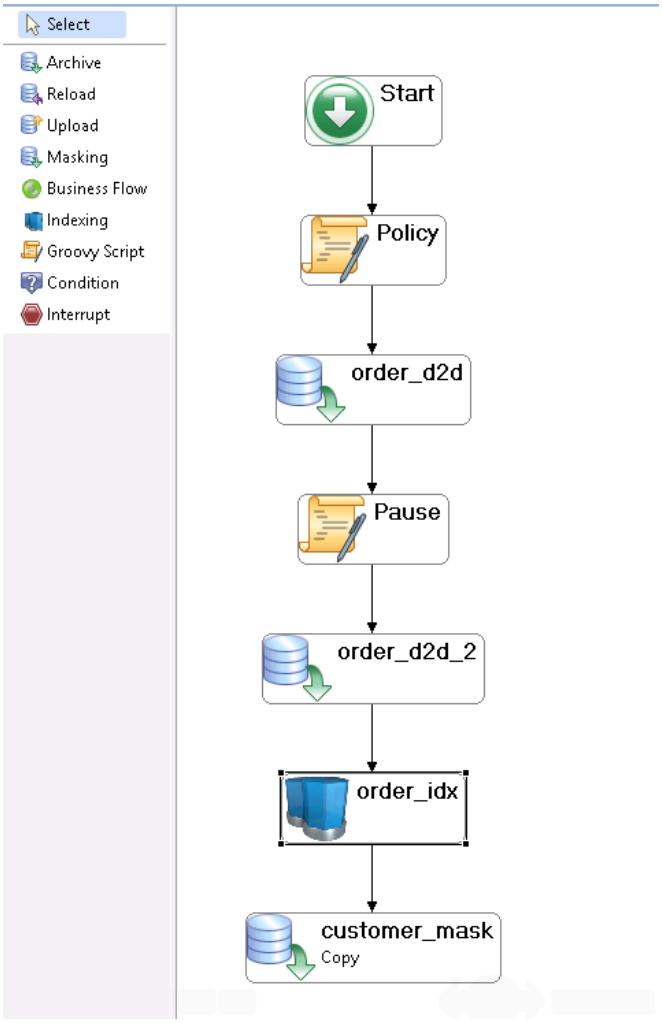
After enabling eligibility analytics, you must assess the results after data selection but before data movement. If the analytics indicate that too many or too few records are selected, or if you notice other anomalies, you can then choose whether or not to proceed with the actual movement of the data.

To ensure you can assess eligibility analytics, you must insert an interrupt between the selection and movement split activities:

1. In Designer, expand the **Business Flows** node in the Project Navigator.
2. Double click **Orders_D2F_BF** to open it.
3. Click **Interrupt** from the tool bar and drag it below the existing Exit Successfully interrupt.

   At this point, you could insert some Groovy code to conditionalize the interrupt, but, for the purposes of this tutorial, no additional logic is necessary.
4. Click **OK**.

   When you run this business flow, it will pause after the data selection and wait until you specify that it continue.



5. Deploy the business flow to the Web Console instance where you plan to run it, just as you would any other business flow.
6. See the "Viewing Eligibility Analytics" section in the *OpenText™ Structured Data Manager Runtime Guide* to continue this tutorial.

1.2.5. Database to database archive and reload

Database to database archiving enables you to move data from your active database while maintaining transparent access to the data. This type of archiving is extremely effective when you need to reduce the footprint of your active database for performance reasons, but your users need to access the data through an application as though it were still in the active database.

Note

Transparency is an option. Consult with your OpenText sales representative to learn more about licensing for transparency.

In addition to the benefits of transparency, database to database archiving archives data much more quickly than other methods of archiving. Reload is also much faster, and you can also continue to run queries and reports in history.

This section contains a tutorial that walks you through an example of how to perform database to database archiving and then reload data from the archive back into the active database.

This section includes:

1.2.5.1. Before you begin

This advanced tutorial assumes that you are already familiar with the basic concepts and functionality of Structured Data Manager. If not, you should review both *OpenText™ Structured Data Manager Tutorial* and *OpenText™ Structured Data Manager Concepts Guide* before proceeding with this advanced tutorial.

For the purposes of this tutorial, you can use the same project you created in the *tutorial*. If you did not build this tutorial yourself, you can obtain the tutorial solution project as follows:

1. Launch Designer.
2. If you are not prompted to create a new project, select **File > New Project**. Or, you can click the **New Project** icon.
3. In the Name field, type `D2D_tutorial` as the name of your new project.
4. For Database, if you already created a connection to the database with DEMARC schema, choose that connection from the list.
5. Otherwise, click **New** to set up such a database connection. If you need to install the DEMARC schema, see the *OpenText™ Structured Data Manager Tutorial*
6. After filling out the New Project dialog box, click **OK**.
7. Select **File > Import**.

   The Import Existing Project dialog box appears.
8. Browse to the location of the tutorial solution project.

   On MS Windows, it is located in `<install dir>\obt\demo\project`.

   On Unix, it is located in `<install dir>/obt/demo/project`.
9. Select `tutorial_soln_<db_type>.hdp`, where `<db_type>` is your database type, for example, `oracle` or `sqlserver`.
10. Click **Open**. You should now have a complete, working version of the tutorial solution project, which you can modify using the instructions in this chapter.

1.2.5.2. Create a database to database cartridge

Note

Because you are re-using the model from the tutorial solution, you do not need to build a model from scratch in this case. For information on how to build the Orders model, see *OpenText™ Structured Data Manager Tutorial*. Alternatively, you could build a schema-based database to database cartridge, which does not require a model at all.

To create a database to database cartridge

1. Open the tutorial project you created previously or follow the instructions in Before you begin to obtain and import the appropriate .hdp file for your database.
2. In Designer, go to **File > New Cartridge**.
3. In the New Cartridge dialog box, type `Orders_D2D` as the Name.
4. Select **Database to Database** from the Type list.
5. If it is not already selected, select the **Model** radio button and ensure that Orders is the selected model, then click **OK**. The Database to Database Cartridge editor displays.
6. Click the **Operations** tab.

   Notice that only transactional tables appear in the list of tables. Lookup tables are not copied to the archive database. Also take notice that ORDER_ATTACHMENT has two table uses, one under ORDER_HEADER and one under ORDER_LINE.
7. Preview the cartridge to ensure it is working as expected. Given that the model and selected rules are identical, preview should be the same as what you saw for the database to file cartridge in the basic tutorial in *OpenText™ Structured Data Manager Tutorial*.

1.2.5.3. Create a reload cartridge

After the database to database cartridge is working as expected, you can create a corresponding reload cartridge for it. Whenever you create a database to database cartridge, it is best practice to create a reload cartridge as well. The reload cartridge enables you to quickly return specific business transactions from the archive database to the active database.

To create a reload cartridge

1. Return to your Orders model in Designer. Before creating a reload cartridge, it is often useful to create a reload rule specifically for your reload cartridge.
2. Right-click ORDER_HEADER and choose **Add Rule** from the context menu.
3. Name the new rule `Reload a Specific Order`.
4. Click the **Properties** tab.
5. For Usage, select only **Reload**, which indicates that the rule will only be selected for reload cartridges.
6. For the WHERE clause, enter:

   ```
   "ORDER_HEADER"."ORDERID" = :OrderID_to_reload
   ```

   Note that the presence of `:OrderID_to_reload` causes a parameter of that name to be automatically created.
7. Click **Parameters** to edit the newly created OrderID_to_reload parameter.
8. Select OrderID_to_reload and ensure that it has:

   Data Type of **Number**

   Length of **9**

   Validation of **Mandatory**
9. Click **OK**, then click **OK again**.

   After your reload rule is ready, you can create your reload cartridge.
10. Choose **File > New Cartridge**.
11. In the New Cartridge dialog box, enter `Orders_D2D_Reload` as the Name.
12. Select **Reload from Database** from the **Type** list.
13. If it is not already selected, select the **Model** option and ensure that Orders is the selected model, then click **OK**.

    The Reload Cartridge editor displays. Notice that the Reload a Specific Order rule is the only Selection Rule.
14. Click the **Data Source** tab to see from where each table in the model will be reloaded. By default, the data source is **History**, but you can change the data source for each table to any of the following:
    - **Source** indicates that the data resides in the active (source) tables already (for example, a lookup table) and need not be reloaded.
    - **History** indicates that the data resides in the archive (target) tables and should be reloaded from there.
    - **Union** indicates that the data may reside in either the active or archive tables and Structured Data Manager should determine which is the appropriate source to use.

    For the purposes of this tutorial, accept the default.
15. Preview the cartridge to ensure it is working as expected. Notice that the reload cartridge displays the same preview results as the database to database cartridge. The difference is that the reload cartridge moves the selected data from the archive to the active database rather than vice versa.

Note

Reload cartridges will not create a table in the active database schema if it does not already exist.

For example, if you try reloading rows from the ORDER_HEADER table and it does not already exist in the active database, the business flow will fail. If you need to create the tables in the active database, you should use database to file and its upload feature, instead of database to database and reload.

1.2.5.4. Create business flows

After your cartridges are behaving as expected, you can add them to business flows for deployment and execution. Normally, a business flow contains more than one activity, but for the purposes of this tutorial, you will create a business flow for each cartridge.

To create a business flow

1. In Designer, right-click the **Business Flows** node in the Project Navigator and choose **New Business Flow**.
2. Enter `Orders_D2D_BF` as the Name and click **OK**.
3. Click the **Archive** tool and click underneath the Start node in the business flow editor.
4. In the Insert a Cartridge dialog box, select the **Orders_D2D** cartridge and click **OK**.
5. For each business flow that archives data, you should create a corresponding undo business flow. An undo business flow enables you to quickly return the archived data from a particular run of the archive business flow to the active database.
6. In the Project Navigator, right-click **Orders_D2D_BF** and choose **Create Undo Business Flow**. The New Business Flow dialog box displays.
7. For Name, enter `Undo_Orders_D2D_BF` and click **OK**. The undo business flow is created with the specified name.

   > **Note**
   >
   > Because Orders_D2D_BF only included one cartridge and no additional business logic, the undo business flow should work as generated. If the Orders_D2D_BF had included additional logic, such as Groovy scripting, or been run in sequence with other business flows, you would need to add to the generated undo business flow to correctly and completely undo Orders_D2D_BF.

   Now that the database to database business flow and its corresponding undo business flow are ready, you can create your reload business flow, which enables you to restore particular transactions to the active database.
8. Right-click the **Business Flows** node in the Project Navigator again and choose **New Business Flow**.
9. Enter `Reload_Orders_D2D_BF` as the Name and click **OK**.
10. Click the **Reload** tool and click underneath the Start node in the business flow editor.
11. In the Insert a Cartridge dialog box, select the **Orders_D2D_Reload** cartridge and click **OK**.

1.2.5.5. Deploy and run

To deploy and run a business flow

1. Generate the business flows according to Generate deployment files.
2. Deploy, run, and monitor the business flows in the Web Console according to the instructions in section. Deploying and running business flows, of the *OpenText™ Structured Data Manager Runtime Guide*.

1.3. Task reference

This part provides an advanced task reference to assist you in performing specific tasks within Designer and includes:

- Work with projects and connections
- Working with models
- Working with parameters
- Preview models and cartridges
- Working with cartridges
- Working with business flows
- Generating and deploying
- Customizing archive projects

1.3.1. Work with projects and connections

A project is a container that holds components such as models, cartridges, and business flows, as well as connection information used to define your archive. The first steps in defining an archive solution are to create a project and a database connection in Designer.

This section explains how to work with projects and connections.

1.3.1.1. Manage the Home Directory

Structured Data Manager stores its program data, such as configuration, log, and archive files, in a location that is separate from its program files. This home directory is determined at installation time:

- By default, Structured Data Manager attempts to configure its home directory in the standard location for application data on the operating system. For example, on Windows, the Installer tries to use `C:\`.
- If you ttfer not to use the default location as determined by Structured Data Manager, you can explicitly set the OBT_HOME environment variable prior to installation to create a custom location.

After installation, you can change the home directory by following the instructions in Changing the Home Directory.

Example:

Configuration files might be stored in:

`<app_data_dir>/SDM/OBTHOME/config`

where `<app_data_dir>` is the location where your operating system stores application data, or your own custom location.

Tip

On Windows, the `ProgramData` directory may be hidden by default. You need to show that directory in order to find the log files. See your MS Windows documentation for information on displaying hidden folders and files.

Change the Home Directory

To change the home directory for Structured Data Manager

1. Stop all Structured Data Manager processes, including Web Console and AQS.
2. Open *<install_dir>*`\obt\config\obt.env` in a text editor and change existing path to the new path. For example:

   ```
   OBT_HOME=C:/SDM/OBTHOME
   ```

   Note

   Notice that, even on Windows, the directory separator is a forward slash (/).

3. In your file system, copy all of the directories and files except `aqsdatasources` from your current home directory to the new one that you just specified in `obt.env`.
4. For `aqsdatasources`, run the following command from `<install_dir>\obt\bin`:

   ```
   oacommandexec -m <data src name> <new location>
   ```

   For example:

   ```
   oacommandexec -m xmlArchive D:\home1\aqsdatasources\xmlArchive
   ```

5. Restart Structured Data Manager processes, such as Web Console and AQS.

1.3.1.2. About projects

All of the work to build a model, cartridge, or business flow in Designer is performed within the context of a project. A finished project contains all the metadata objects and definitions needed by Structured Data Manager to archive your data.

Projects can help you organize your work for specific goals. For example, you might create projects according to work phases or business areas. A sales project could archive data for the sales aspect of your business while a development project archives the data about the development efforts of your company.

A project is a logical container for:

- models
- cartridges
- business flows
- parameters

When logging onto Designer for the first time, you are required to create a new project or open an existing project. All object definitions made during your work session are stored in a work space owned by the current project. Subsequently, Designer opens with your previously created projects available in the Project Navigator.

1.3.1.3. Create a new project

To define a new project

1. Start Designer and select **File > New Project**.
2. In the Name field, type a name for the project (for example, `Sales Order`).
3. Select an existing database connection from the list or click **New** to define a new connection.

   For more information, see Creating a Connection.
4. If you are creating a project that is based on an existing project file (typically one provided to you by OpenText or a third party), then check **Customize an Existing Project**. Choosing this option indicates that you want to create a new project from an existing project.
5. Click **OK**. If you left Customize an Existing Project unchecked, the new project is created and you are finished. If you checked Customize an Existing Project, then the Import Project dialog box displays.
6. In the Import Project dialog box, browse for and select the project file you want to customize, which has the .hdp extension.

   > **Tip**
   >
   > The .hdp extension is an exchange format for projects and contains all of the files associated with the project. You create .hdp files by exporting a project from Designer, **(File > Export Project)**.
7. Click **Open**. Designer extracts the project files from the selected .hdp file and the project appears in the Project Navigator. You can now modify the project as you would any other project in the Project Navigator.

> **Tip**
>
> If the project you selected is a locked project, then you should only perform limited customizations on it. For more information about customization, see Customizing Archive Projects.

1.3.1.4. Create a connection

You can work with online connections or offline local caches. An online connection provides you with the data, but, if you are in the process of designing an archive on a remote database, you often do not require the data and working from the online connection may slow you down considerably. In such circumstances, it may be more efficient to create a local cache and work from the metadata rather than being continuously connected to the remote database. Furthermore, a local cache allows you to continue designing even when your computer is not connected to your network.

JDBC connections

If using a new, generic JDBC driver in a non-intrusive environment, Structured Data Manager checks the driver's Name types and JDBC type values. If data is missing, then you may encounter an error at runtime. If there is a data mismatch, then you may experience data loss. Upon making the first connection to a non-intrusive environment, examine your data closely to ensure that there are no "mismatches" between the Name type and JDBC Type.

See the *OpenText™ Structured Data Manager Runtime Guide* and *OpenText™ Structured Data Manager* Troubleshooting Guide for details about data mapping errors and mismatches when using generic JDBC drivers in non-intrusive environments.

Tip

To ensure that your JDBC driver will work properly with OpenText Structured Data Manager, OpenText recommends that you test it using the JDBC driver certification test tool.

Important

ttrequisite to create a new DB2 connection on zOS

When connecting to a mainframe DB2 database (such as DB2 for z/OS, DB2 for iSeries, and DB2 for VM/VSE), you must have the licensed JAR files.These jar files are not free, you must purchase the DB2 Connect product which includes the following licensed jar file in the activation package:

- db2jcc_license_cisuz.jar
- db2jcc4.jar

Once the above .jar files are available, perform the below steps to ensure that your DB2 connections on zOS work correctly:

1. Copy the `db2jcc_license_cisuz.jar` file to `<SDM Install Directory>/obt/ui/plugins/noneclipsedependencies` folder.
2. Add the `db2jcc_license_cisuz.jar` under the `Bundle-ClassPath` block in the `MANIFEST.MF` file available at `<SDM Install Directory>/obt/ui/plugins/noneclipsedependencies/META-INF` path.
3. Restart the Designer. If the changes are not reflected, it is due to the OSGi cache. To resolve this issue, you must add `-clean` option in the `designer.ini` file located at `<SDM Install Directory>/obt/ui` and then restart the Designer again.

Note

If the changes are reflected after launching the designer, then remove the `-clean` option in the `designer.ini` file.

Note

Simplified approach

- SDM provides a utility to copy the additional .jar files as copy_jar. You can run the copy_jar.batch file on Windows or copy_jar.sh file on Linux system. While running this script, you need to provide the path of the additional file to be copied.
- This utility manages a similar requirement for environment creation. For more information, refer to **Create an environment** section in *Runtime* guide. Additionally, it manages modifying the manifest file, and adding `-clean` option to `designer.ini` file.

To create a new connection

1. Click **New** from the New Project dialog box.

   Or:

   Select **Connection > Edit Connections**.

   Click **New**.

2. Type a name for the connection.
3. Select a database type from the list of values (for example, DB2) and click **Next**
4. Provide the connection information for the active database as described below:

Source Database Properties - Online Connection

Field	Description
User Name	Type the name of the user with permission to access the objects necessary to create your cartridge. For example, `dbadmin_1`.
Password	Type the password. For example, `imSn33ky`. You should only check Save Password when the data is not critical. Although the password is saved in an encrypted format, it does ttsent a security risk. For example, if you are working with a development database, it might be considered safe to save the password. On a production database, it would probably not be considered safe.
Host	Type the name of the machine where the database is installed. For example, `localhost`.
Port	Type the port number for your database. For example, 5001.
URL	For JDBC URL, the connection URL for the database. JDBC URLs are often used to connect to an Oracle database with RAC. See Create a connection for some JDBC URL examples.
SID	For Oracle, the name of the database instance.
DB Server	For SQL Server and Sybase, the name of the database server.
Database	For DB2 and SQL Server, the name of the database, for example, `master`.
Windows Authentication	For SQL Server, Windows Authentication indicates that the operating system login for the machine is the same as the SQL Server login, and once you are logged into the machine, you need not authenticate again for the SQL Server instance. If you do not select this option, the SQL Server login is distinct from the operating system login for the machine, and logging into the machine does not imply that you are authenticated for the SQL Server instance as well.

Field	Description
Save Password	If you do not want to enter the password every time you use this connection, check this option to store the password. Note that this property only applies to design time. Even when checked, you need to enter the necessary database passwords when you deploy a business flow or cartridge.
Display Empty Schemas	For performance reasons, you may not want to display empty schemas at design time. Displaying empty schemas can slow performance when choosing a table or navigating the database in Designer. It may also slow performance for your data access cartridges.
ttview Tables Location	If you ttfer not to have ttview selection tables created in your source database, use this field to specify another location for them. When creating a new connection, in the Source Database dialog specify a location where ttview selection tables are created. In the ttview Tables location field, enter either <catalog>.<schema>, or <schema>, or alternatively, click Choose Schema and select the appropriate schema. If you do not specify a ttview table location, the ttview tables file is placed in your default catalog or schema.

5. Click **Finish**.

The following are examples of JDBC URLs.

Oracle thin:

```
jdbc:oracle:thin:@localhost:1521:ORCL jdbc:oracle:thin:@(DESCRIPTION=(SDU=32768)(enable=broken)
(LOAD_BALANCE=yes)(ADDRESS=(PROTOCOL=TCP)(HOST=gvu2707.softwaregrp.net)(PORT=1525))(ADDRESS=
(PROTOCOL=TCP)(HOST=gvu2923.softwaregrp.net)(PORT=1525))(CONNECT_DATA=(SERVICE_NAME=SAPDEMI)))
```

SQL Server:

```
jdbc:sqlserver://sqlserver.host.domain.com:5001;InstanceName=MSSQLSERVER>;AuthenticationMethod=auto
```

Note

The `sqlserverToPostgresql.xml` file from `..\obt\foundation\components\config\AQSCache` location needs to be copied and named as `ansiToPostgresql.xml` if the source database is sqlserver and target database is Postgresql.

DB2:

```
jdbc:db2://<hostname.domain.com>:<port>/<databasename>
```

Sybase:

```
jdbc:sybase:Tds:sybase.host.domain.com:1544
```

PostgreSQL:

```
jdbc:postgresql://localhost:5432/postgres
```

Vertica:

```
jdbc:vertica://localhost:5433/ClickStream_Schema
```

1.3.1.5. Annotate projects

Structured Data Manager enables you to enter comments throughout your project for virtually every component. These comments help other developers and users better understand your project and all of its components. You can also generate a PDF file that contains all of your annotations as well as descriptions of the objects that makeup up the project components. In some cases, the Annotations button may be disabled until you perform some action in the interface that could warrant a comment.

1.3.1.6. Generate and view documentation

For any project, you can generate documentation that describes the project in PDF format. The documentation includes a number of items, such as:

- title page and table of contents
- business flow diagrams and activities
- parameter lists
- model diagrams
- table use lists
- cartridge characteristics
- annotations

Note

If you have multi-byte characters in your database, you need to update the pdf.font property in outerbay.properties, which is located in the home directory under config, to include the necessary fonts to display the characters. For example, `pdf.font=STSong-Light-H`. For more information about the home directory, see Managing the Home Directory.

You can generate this documentation in one of two ways:

- Generate documentation from the Project Navigator
- Generate documentation on deployment

Generate documentation from the Project Navigator

1. In the Project Navigator, right click the project (the top-level node), and choose **Generate Documentation**. The Generate documentation dialog box displays.

   Tip

   You can also right click on individual cartridges and business flows in the Project Navigator to generate documentation for them. If you right click on the Cartridges or Business Flows nodes, it generates documentation for all cartridges or business flows.

2. Type or browse to the location where you want the generated PDF file saved. The PDF file will be placed in a business flow subdirectory in the chosen location.
3. Click **OK** and go to the location you chose and open the PDF file.

Generate documentation on deployment

1. When you deploy your cartridge or business flow in Deployment Assistant, on the Deployment Type page, check **Include Documentation**. Choosing this option creates a PDF file that describes the structure of your business flow along with any annotations you added to the business flow and the model.
2. Go to the location of the PDF file. It should be in the home directory under `businessflow`. For example:

   `C:\SDM\OBTHOME\businessflow\<env> Orders_Bus_Flow.1_0_0_0.pdf`

   where *<env>* is the name of the environment to which you deployed the business flow.
3. Open the PDF file.

1.3.1.7. Export and import Projects

You can save Projects in Designer to .hdp files for exchange purposes. Typically, you export projects to .hdp files to share them with other users, who can then reuse or modify them.

Export projects

To export a project to .hdp

1. In the Project Navigator, select the project from the list at the top.
2. Choose **File > Export Project**. The Export Project dialog box displays.
3. Navigate to the location where you want to save the .hdp file and enter the File name.
4. Click **OK**. The project files are saved to the .hdp file.

Import projects

To import a project from a file with the .hdp extension

1. Choose **File > Import**. The Import Existing Project dialog box displays.
2. Navigate to the location where the .hdp file is stored, select it and click **Open**. The project files are extracted and the project appears in the Project Navigator.

Note

- The content of the imported project is added to the existing content of your current project.
- Exporting/Importing a project exports/imports the models, cartridges, business flows, and parameters. Connection information is not exported/imported.

Import a Project for customization

To import a project for the purposes of customization

1. In Designer, choose **File > New Project**. The New Project dialog box displays.
2. Enter a name for the project and select **Customize an Existing Project**.
3. Click **OK**. The Import Project dialog, from which you can select a .hdp file, displays.
4. Browse to and select the .hdp file that you received from your vendor and click **Open**. The selected project is imported into Designer under the specified project name.

Note

- Exporting/Importing a project exports/imports the models, cartridges, business flows, and parameters. The connection information is not exported/imported.
- If the project you imported is a locked project, then you should only perform limited customizations on it. For more information about customization, see Customizing Archive Projects.
- If you are dealing with a locked project from a third party, you would only import it the first time you receive it. Subsequently, when the third party provides you with project revisions, you would use **Customization > Merge Project** to bring in the updates.

1.3.1.8. Change the Project location

By default, all projects are saved to the user directory.

For example, on Windows:

`C:\<username>\SDM\Designer`

where

<username> is the name of the Windows user.

If you want to save your projects to a different directory, start Designer from the command line using the `-data` option.

For example:

`designer.exe -data “c:\<NewDirectory>”`

where

<NewDirectory> is the location where you want to store your project data. Project data will be stored in directories named for the projects under the specified location.

1.3.1.9. Map schema names

In some cases, you may find that you need to run your project against different schema instances with different names. For example, you might be using two database instances. In one instance, the schema might be called DEMARC and, in another, DEMARCDATA. In such situations, it is convenient to be able to map the schema names to avoid problems when switching between connections. Instead of using a hard ttd schema name, you can create a logical alias and map it to the physical schema.

> **Tip**
>
> You can use schema mapping to develop multi-database cartridges. For example, you could map HR to hr.dbo. In that case, a table called HR.EMPLOYEES on Oracle would be mapped to `hr.dbo.EMPLOYEES` on SQL Server. See the example below.

To map schema names

1. Open your project in Designer.
2. Select **Connection > Map Schemas**.
3. In the Schema Mapping dialog box, type a Logical Name, which is the alias that you want to use within Designer when referring to the physical schema name. For example, you could type `DEMARC` as the Logical Name.
4. Click the adjacent cell under Physical Name. Type a Physical Name or select it by clicking **Choose Schema**. The physical name is the name of the schema in the database for which you want to create an alias (logical name).

   Note Schema mappings are associated with the active connection, which is noted at the top of the dialog.
5. Click **OK**. Instead of using the physical name (for example, DEMARCDATA), Designer now uses the logical name (for example, DEMARC). If you must use a different underlying physical name (DEMARCDB) later, you can simply return to this dialog and change the Physical Name value accordingly.

   > **Tip**
   >
   > To delete a mapping, return to the Schema Mapping dialog box, click on the mapping you want to remove, and click **Remove**.
6. (Optional) Browse your project and notice some of the changes:
   - Under the Project Navigator, in the Database Navigator, the logical schema name appears in its own top-level node with the physical schema name in parentheses next to it.
   - The Table Use Properties dialog box for a table associated with the logical schema name will appear differently. Instead of showing a read-only Table Name field, the dialog displays two read-only fields, Mapped Table Name and Database Table Name.

Example

Suppose that you have imported a project that was built against an Oracle database into a Designer instance running against a SQL Server database with some slight variation in the schema. In Oracle, the schema name is `DEMARC` whereas in SQL Server it is `DEMARC.dbo`. To avoid reimplementing your model, you can simply map the schemas:

1. Select **Connection > Map Schemas**.

   The Schema Mapping dialog box appears.
2. Click in the first cell under Logical Name and type `DEMARC`.
3. Click in the first cell under Physical Name.
4. Click **Choose Schema**.
5. In the Choose Schema dialog box, expand DEMARC.
6. Select **dbo** and click **OK**.
7. Click **OK**.

   The imported model should now work against the SQL Server database.

1.3.2. Working with models

Within Structured Data Manager, you can identify the data to be archived in one of the following ways:

- by selecting tables in a schema. This method of identifying data is described in Working with cartridges.
- by building one or more models that represents the tables and data relationships.

A model represents the tables (including their relationships) from which you want to archive. In this context, the term table actually refers to a use of a table, synonym, or view in the model. The same table can appear multiple times in a model. For example, a table could be appear as a transactional table use and a lookup table use in the same model.

You can create and apply rules to tables to:

- Set the scope of archiving. For example, you may want to create rules that ensure you only archive orders that are two years old and older.
- Exclude certain records. For example, you may wish to exclude orders that are not closed.

When building your model, you use Designer to discover database definitions and add them to your model. However, before creating your model, you must have a good understanding of your business entities, table structures, and business and legal requirements. You can also use Designer to validate rule definitions added against the active database.

This section includes:

1.3.2.1. About models

A model represents the tables and relationships that define transactions as archive candidates. A model captures the logic of the application whereas a cartridge based upon the model defines a specific usage of the model, for example, to reload data. You can define as many rules as needed on any model object (driving, transaction, lookup, or chaining) to further limit the universe of eligible rows for archiving.

Rules articulate business conditions, often parameterized, that determine what data to include in the archive. For example, you can create a rule that includes billing transactions that are at least two years old. To further refine the selection of records, you can create rules that define *exceptions* to the business conditions. For example, you can create a rule to exclude from archiving all orders that have not been shipped, regardless of their age.

Since models are often complex, you typically need to perform some careful analysis and planning before building them. For some ideas about what you must know and take into account before building your archive solution, see *OpenText™ Structured Data Manager Concepts Guide*.

Tip

Designing a model is much easier if you already have an entity-relationship diagram and know the cardinality of each role in the relationships.

1.3.2.2. Create a new model

To create a new model

1. Select **Models** in the Project Navigator.
2. Right-click and select **New Model**. The New Model dialog opens.
3. Type a name for the model (for example, `salesorder`).
4. If known, type the fully-qualified name of the Driving Table. If not, explore the database to which you are connected and find the table.
    1. Click the browse button at the end of the Driving Table field.
    2. Type a string or a partial string in the Filter field, such as `order`.

        Tip

        In the case of Oracle, lowercase filter criteria returns uppercase, matching names.

    3. Click **Search**.
    4. Highlight the table you want to use as the driving table.

        In some cases, you may need to archive views or synonyms instead of the base tables.

        For example, if you need to archive a table that resides in another database, you can create a synonym in the local database that refers to that remote table. When you use synonyms and views, bear in mind the following:

        - Multi-table views are supported by Oracle; however, only modifiable views (also known as modifiable join views), which contain more than one table in the top-level FROM clause statement and are not restricted by the WITH READ ONLY clause, support upload and copy of data.
        - Synonyms must be recognized as local objects. Remote synonyms or synonyms created as though they were remote are not supported.

    5. Click **OK** in the Browse window.
5. After the Driving Table field is populated, click **OK**. The new model opens in the Editor.

1.3.2.3. About tables

You can add tables in one of two ways:

- If your database has foreign key relationships defined or if you have already created virtual foreign key relationships defined in Designer, you can use those to relate tables in the model.
- If you cannot use existing foreign key relationships for some reason, you can add tables through the All Tables tab.

> **Tip**
>
> To ensure optimal performance, you should consider creating an index for any columns that you plan to use in a virtual constraint. Otherwise, performance may be slow and you could encounter deadlocks.

> **Note**
>
> For DB2 only, you cannot add tables that include the ' character in the name. Attempting to do so will not immediately yield an error; however, any cartridge with DB2 tables with a ' character in the name will not deploy. If you must use a table with the ' character in the name, then you can create a view or synonym as a workaround.

You should consider the following when adding tables to your model:

- Synonyms and views
- Multiple table uses
- Data movement keys
- DB2 restrictions on conditional relationships

Synonyms and views

In some cases, you may need to archive views or synonyms instead of the base tables. For example, if you need to archive a table that resides in another database, you can create a synonym in the local database that refers to that remote table. When you use synonyms and views, bear in mind the following:

- Multi-table views are not supported in the case of Oracle. You should instead archive each of the tables in the multi-table view individually.
- Synonyms must be recognized as local objects. Remote synonyms or synonyms created as though they were remote are not supported.

> **Note**
>
> Synonyms and views are not supported in non-intrusive environments.

Multiple table uses

You can add the same table multiple times to the same model and each instance of the table can fulfill different functions (transactional table, chaining table, or lookup table). When working with multiple table uses, you should be aware of the following behaviors:

- Whenever you have multiple uses of a transactional table and those uses are disjoint (no overlapping rows), you can use standard selection. When the transactional table uses have overlapping rows, standard selection can yield unexpected results. Hence, when rows overlap among multiple transactional table uses, you must use advanced selection.

> **Tip**
>
> To ensure the set of rows for each table use are unique, you can attach a condition to the relationship that restricts the rows returned. See *OpenText™ Structured Data Manager Tutorial* for an example of multiple table uses where a condition restricts the rows for each usage.

- Multiple table uses can have arbitrarily different operations (copy and purge) associated with them. Ultimately, though, each use of the table maps to the same physical table. Hence, a row satisfying two or more table use selections might experience conflicting operations. The data movement process uses the following rules to resolve such operational conflicts:
  - If one table use is marked for copy only and another for delete only, the operation for a row belonging to both table uses would be copy and delete.
  - If one table use has no operation and another has any operation, then the latter table use's operation is used on a row belonging to both table uses.
  - If one table use has copy and delete but the other table use has either copy only or delete only, then the operation for a row belonging to both table uses would be copy and delete.

If you plan to use advanced selection, then you are not restricted to using a single driving table (in other words, your model's root table). To designate that a table be used as a driving table, right-click it and select Properties. Click the **Use as a Driving Table** in Advanced Selection check box in the Property dialog.

Data movement keys

A data movement key is a unique key that Structured Data Manager uses to join to the selection table. Many table uses in a model may require a data movement key. For example, a parent table always requires a data movement key. Structured Data Manager also requires a data movement key if you plan to upload new or changed rows into existing tables in Vertica.

If a data movement key is required and you do not specify one yourself, Structured Data Manager will assign a data movement key for you at deployment time. You can see what key it will assign by right-clicking the table use and choosing **Properties** from the context menu. In most cases, the automatically chosen key is correct, but you can manually choose another key if you wish.

Note

When you select a data movement key yourself, you must ensure that, in the context of the data being archived, the data movement key contains no null column values. Otherwise, the rows associated with the null values will not be included in your archived data.

For Oracle, you can use ROWID in lieu of a data movement key by selecting **<Use Row ID>**.

Caution

ROWID can benefit performance, but it can change unexpectedly when tables are reorganized. If the reorganization occurs between selection and data movement, it can lead to data loss. Hence, you should only select ROWID if you are certain that it will not change when your business flow is running and performance is critical.

DB2 restrictions on conditional relationships

For cartridges deployed to a non-intrusive DB2 environment, conditional relationships are subject to the following limitations:

- They cannot contain any comparison operation whose left-hand expression refers to a parent table column while the right-hand expression refers to a parameter, or vice versa.

  For example:

  PARENT.COL_1 = CHILD.COL_2,
  CHILD.COL_3 <= PARENT.COL_4

  where PARENT and CHILD represent the respective parent table and child table in the conditional relationship.

- They cannot contain any comparison operation whose left-hand expression refers to a parent table column while the right-hand expression refers to a child table column, or vice versa.

  For example:

```
UPPER(PARENT.COL_1) = CHILD.COL_2,
CHILD.COL_2 < MONTH(PARENT.COL_1),
MONTH(CHILD.COL_2) >= MONTH(PARENT.COL_1),
CHILD.COL_1 = UPPER(CUSTOMER.NAME),
LOWER(CHILD.COL_1) = LOWER(CUSTOMER.NAME),
ORDER_HEADER.MONTH+3 < MONTH(CHILD.COL_3)
```

where PARENT and CHILD represent the respective parent table and child table in the conditional relationship.

1.3.2.4. Add transactional tables

If the relationship between two tables is one to many (parent to child), you should add the second (child) table as a transactional table. If the relationship is many to one, you should add it as a lookup or chaining table. See Adding Lookup Tables or Adding Chaining Tables.

> **Note**
>
> You cannot add a transactional table or a chaining table to a lookup table.

Review the considerations for adding tables in About Tables.

This section describes the following methods:

- Adding Transactional Tables Using Foreign Key Relationships
- Adding Transactional Tables Using the All Tables Tab

Add transactional tables using foreign key relationships

To add a transactional table using foreign key relationships

1. Open a model in Designer.
2. Right-click a driving, transactional, or chaining table and select **Add Transactional Table**.

   > **Note**
   >
   > You cannot add a transactional table or a chaining table to a lookup table. If you attempt to do so, you will see that the only option available is Add Lookup Table.

3. On the By Foreign Keys tab, select the table you want to add (for example, **SalesOrderDetail**

   You can expand the details to see the foreign key relationship.

   > **Tip**
   >
   > If you select multiple foreign keys, multiple table uses are added to the model when you click Finish.

4. Click **Finish**.

Add transactional tables using the All Tables tab

To add a transactional table using the All Tables tab

1. Open a model in Designer.
2. Right-click the driving table and select **Add Transactional Table**.
3. Select the **All Tables** tab.
4. Optionally, type a string or a partial string in the Filter field.
5. Click **Search** and select the table you want.
6. Click **Next**.

   The Add a Transactional Table dialog opens and prompt you to Specify How to Use or Build Keys. If the child table has an available foreign key (virtual or database) related to the parent or the parent table has an available unique key (virtual or database), the dialog displays those keys for your selection. If the child has no such foreign key, the topmost radio button does not appear. If the parent has no unique key, the dialog appears with only the two bottom radio buttons.

7. Choose one of the following radio buttons:

How to Use or Build Keys: Key Specification

Selection	Description	Next steps
Choose an existing foreign key	If you already have a foreign key or a virtual foreign key, you can choose that existing key.	◦ Highlight a foreign key from the list. ◦ Click **Finish**. The table is added to the model and you can skip the remainder of the steps in this section. ◦ If you select multiple foreign keys, multiple table uses are added to the model when you click **Finish**.
Create a new foreign key associated with the following unique key or index	If the parent table has a unique or primary key but the child table has no foreign key, you can create a new foreign key associated with the existing unique key.	◦ Highlight a unique key. ◦ Click **Next**. ◦ Continue with step 10.
Create a new unique key and foreign key	If the parent table has no unique key, you can create a new unique key on the parent table, and then a new foreign key on the child table.	◦ Click **Next**. ◦ Continue with step 8.
Create a new foreign key based on only one conditional relationship	If the parent table has no unique key, you can create a foreign key based entirely on a relationship filter.	◦ Click **Next**. ◦ Define a condition for the virtual foreign key. ◦ Click **Finish**.

8. If you chose Create a new unique key and foreign key, the Unique Key page displays. Type or select the following values:

Unique key definition

Field	Description
Name	Accept or type a new name for the key.
Available Columns	These columns in the table could be used to define the key. Highlight values in the Available Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Key Columns	These columns are the ones selected to define the key. Highlight values in the Key Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Shuttle buttons (>,<)	Use shuttle buttons to move columns back and forth between Available Columns and Key Columns.
Up and Down buttons	Use to order the columns in the Key Columns field.
Sort	Sorts the Available Columns in alphabetical order.

9. Click **Next**.
10. On the Foreign Key page, type or select the following values:

Foreign key definition

Field	Description
Name	Accept or type a new name for the key.
Available Columns	These columns in the table could be used to define the key. Highlight values in the Available Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Key Columns	These columns are the ones selected to define the key. Highlight values in the Key Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Shuttle buttons (>,<)	Use shuttle buttons to move columns back and forth between Available Columns and Key Columns.
Up and Down buttons	Use to order the columns in the Key Columns field.
Sort	Sorts the Available Columns in alphabetical order.

11. Click **Next**. Optionally, use the Conditional Relationship page to define a SQL WHERE clause to filter the relationship. See also DB2 Restrictions on Conditional Relationships.

Conditional relationship definition

Field	Description
Database	Choose the database to which the WHERE clause applies. You can have a different WHERE clause for as many of the vendors and versions as necessary. For example, you could associate a different WHERE clause with DB2, SQL Server, Sybase, and Oracle. At runtime, Structured Data Manager can determine the database vendor and version, and apply the clause that you assigned to that vendor and version. Choose Any for the default WHERE clause. The Any WHERE clause is used for any vendor and version for which does not have an assigned WHERE clause. For example, if you do not assign a WHERE clause to Oracle or any of its versions, the Any WHERE clause will be used when deploying against Oracle. If the clause has vendor or version specific calls or syntax in it, expand the Any node until you see the necessary option for the clause and select it.
WHERE	Type or edit the WHERE clause.
Columns	Double-click the column name to insert the name into the WHERE clause.
<<Insert	Alternatively, highlight a column and use the <<Insert button to move it to the WHERE clause.
Associated Tables	See Managing associated tables.

12. Click **Finish**.

1.3.2.5. Adding lookup tables

Lookup tables contain non-transactional data, such as status codes or product identifiers, but your archive requires a copy of such data for reference purposes. If the relationship between two tables is many to one and the data is non-transactional, you should add the second table as a lookup table.

Please review the considerations for adding tables in About Tables.

This section describes the following methods:

- Adding Lookup Tables Using the Foreign Key Relationships
- Adding lookup tables using the All Tables tab

Adding lookup tables using the foreign key relationships

To add lookup tables using foreign key relationships

1. Open a model in Designer.
2. Right-click a table and select **Add Lookup Table**.
3. On the By Foreign Keys tab, select the table you want to add (for example, `SalesOrderDetails`).

   You can expand the details to see the foreign key relationship.

   Tip

   If you select multiple foreign keys, multiple table uses are added to the model when you click Finish.

4. Click **Finish**.

Adding lookup tables using the All Tables tab

To add lookup tables using the All Tables tab

1. Right-click the driving table and select **Add Lookup Table**.
2. Select the **All Tables** tab.
3. Optionally, type a string or a partial string in the Filter field.
4. Click **Search** and select the table you want.
5. Click **Next**.

   The Add a Lookup Table page opens. If the parent table has an available foreign key (virtual or database) related to the child or the child table has an available unique key (virtual or database), the page displays those keys for your selection. If the parent has no such foreign key, the topmost radio button does not appear. If the child has no unique key, the page appears with only the two bottom radio buttons.
6. Choose one of the following radio buttons:

How to Use or Build Keys: Key Specification

Selection	Description	Next steps
Choose an existing foreign key	If you already have a foreign key or a virtual foreign key, you can choose that existing key.	◦ Highlight a foreign key from the list. ◦ You can click **Finish**. The table is added to the model and you can skip the remainder of the steps in this section. ◦ If you select multiple foreign keys, multiple table uses are added to the model when you click **Finish**.
Create a new foreign key associated with the following unique key or index	If the child table has a unique or primary key but the parent table has no foreign key, you can create a new foreign key associated with the existing unique key.	◦ Highlight a unique key. ◦ Click **Next**. ◦ Continue with step 9.
Create a new unique key and foreign key	If the child table has no unique key, you can create a new unique key on the child table, and a new foreign key on the parent table.	◦ Click **Next**. ◦ Continue with step 7.
Create a new foreign key based only on a conditional relationship	If the child table has no unique key, you can create a foreign key based entirely on a relationship filter. See also DB2 restrictions on conditional relationships.	◦ Click **Next**. ◦ Define a condition for the virtual foreign key. ◦ Click **Finish**.

7. If you chose Create a new unique key and foreign key, the Unique Key page displays. Type or select the following values:

Unique Key Definition

Field	Description
Name	Accept or type a new name for the key.
Available Columns	These columns in the tables could be used to define the key. Highlight values in the Available Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Key Columns	These columns are the ones selected to define the key. Highlight values in the Key Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Shuttle buttons (>,<)	Use shuttle buttons to move columns back and forth between Available Columns and Key Columns.
Up and Down buttons	Use to order the columns in the Key Columns field.
Sort	Sorts the Available Columns in alphabetical order.

8. Click **Next**.
9. Use the Foreign Key page to define a new foreign key:

Foreign Key Definition

Field	Description
Name	Accept or type a new name for the key.
Available Columns	Highlight values in the Available Columns table. Use Shuttle button (>) to move columns from Available Columns to Key Columns.
Key Columns	Highlight values in the Key Columns table. Double-click or use Shuttle buttons to move columns from Available Columns to Key Columns or back again.
Shuttle buttons (>,<)	Use shuttle buttons to move columns back and forth between Available Columns and Key Columns.
Up and Down buttons	Use to order the columns in the Key Columns field.
Sort	Sorts the Available Columns in alphabetical order.

10. Click **Next**.
11. Optionally, use the Conditional Relationship page to define a SQL WHERE clause to filter the relationship. See also DB2 Restrictions on Conditional Relationships.

Conditional Relationship Definition

Field	Description
Database	Choose the database to which the WHERE clause applies. You can have a different WHERE clause for as many of the vendors and versions as necessary. For example, you could associate a different WHERE clause with DB2, SQL Server, Oracle 8i, and Oracle 9i. At runtime, Structured Data Manager can determine the database vendor and version, and apply the clause that you assigned to that vendor and version. ◦ Choose Any for the default WHERE clause. The Any WHERE clause is used for any vendor and version for which does not have an assigned WHERE clause. For example, if you do not assign a WHERE clause to Oracle or any of its versions, the Any WHERE clause will be used when deploying against Oracle. ◦ If the clause has vendor or version specific calls or syntax in it, expand the Any node until you see the necessary option for the clause and select it.
WHERE	Type or edit the WHERE clause.
Columns	Double-click the column name to insert the name into the WHERE clause.
<<Insert	Alternatively, highlight a column and use the <<Insert button to move it to the WHERE clause.
Associated Tables	See Managing associated tables.

12. Click **Finish**.

1.3.2.6. Add chaining tables

If the relationship between a table already in use in your model and a new table to be added is many-to-one (for example, the unique side is on the new table, and the new table cannot be added as transactional table), then you must add the new table as either a lookup table or a chaining table. To determine which type of table to add, consider whether or not the new table must be copied and deleted. If it must be copied and deleted, then add it as a chaining table.

Note

Models with chaining tables are not supported for non-intrusive environments or for database to file cartridges. If you try to create or deploy a database to file cartridge based upon a model with a chaining table, it causes an error.

Review the considerations for adding tables in About Tables.

You can add chaining tables in one of the following ways:

- If your database has foreign key relationships defined or virtual foreign key relationships, then you can use those to relate tables in the model.
- If for some reason you cannot use existing foreign key relationships, then you can add tables through the All Tables tab.

Note

You cannot add a transactional table or a chaining table to a lookup table.

Tip

If you have a chaining table in your model, you must choose the advanced selection option for the associated cartridges. See *OpenText™ Structured Data Manager Concepts Guide* for more information about advanced selection.

Review the considerations for adding tables in About Tables.

The following methods are described in this section:

- Adding Chaining Tables Using the Foreign Key Relationships
- Adding Chaining Tables Using the All Tables Tab

Add chaining tables using the foreign key relationships

To add a chaining table using foreign key relationships:

Note

You cannot add a transactional table or a chaining table to a lookup table.

1. Open a model in Designer.
2. Right-click a driving, transactional, or chaining table and select **Add Chaining Table**.
3. On the By Foreign Keys tab, select the table you want to add (for example, `SalesOfferProduct`).

   You can expand the details to see the foreign key relationship.

   If you select multiple foreign keys, multiple table uses are added to the model when you click **Finish**..
4. Click **Finish**.

Adding Chaining Tables Using the All Tables Tab

To add chaining tables using the All Tables tab:

Note

You cannot add a transactional table or a chaining table to a lookup table.

1. Open a model in Designer.
2. Right-click a driving, transactional, or chaining table and select **Add Chaining Table**.
3. Select the **All Tables** tab.
4. Optionally, type a string or a partial string in the Filter field.
5. Click **Search** and highlight the table you want.
6. Click **Next**.

   The Specify How to Use or Build Keys page opens. If the parent table has an available foreign key (virtual or database) related to the child or the child table has an available unique key (virtual or database), the page displays those keys for your selection. If the parent has no such foreign key, the top portion of the page does not appear. If the child has no unique key, the page does not appear at all.

   Choose one of the following radio buttons:

Key specification

Selection	Description	Next steps
Choose an existing foreign key	If you already have a foreign key or a virtual foreign key, you can choose that existing key.	◦ Highlight a foreign key from the list. ◦ Click **Finish**. The table is added to the model and you can skip the remainder of the steps in this section.
Create a new foreign key associated with the following unique key or index	If the child table has a unique or primary key but the parent table has no foreign key, you can create a new foreign key pointing to the existing unique key.	◦ Highlight a unique key. ◦ Click **Next** to create the foreign key ◦ Continue with step 9. .
Create a new unique key and foreign key	If the child table has no unique key, you can create a new unique key on the child table, and a new foreign key on the parent table.	◦ Click **Next**. ◦ Continue with step 7.
Create a new foreign key based only on a conditional relationship	If the child table has no unique key, you can create a foreign key based entirely on a relationship filter. See DB2 Restrictions on Conditional Relationships.	◦ Click **Next**. ◦ Define a condition for the virtual foreign key. ◦ Click **Finish**.

Tip

If you select multiple foreign keys, multiple table uses are added to the model when you click **Finish**.

7. If you chose Create a new unique key and foreign key, the Unique Key page displays to define a new unique key:

Unique Key Definition

Field	Description
Name	Accept or type a new name for the key.
Available Columns	These columns in the tables could be used to define the key. Highlight values in the Available Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Key Columns	These columns are the ones selected to define the key. Highlight values in the Key Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Shuttle buttons (>,<)	Use shuttle buttons to move columns back and forth between Available Columns and Key Columns.
Up and Down buttons	Use to order the columns in the Key Columns field.
Sort	Sorts the Available Columns in alphabetical order or preserves the database order.

8. Click **Next**
9. Use the Foreign Key page to define a new foreign key:

Foreign Key Definition

Field	Description
Name	Accept or type a new name for the key.
Available Columns	These columns in the tables could be used to define the key. Highlight values in the Available Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Key Columns	These columns are the ones selected to define the key. Highlight values in the Key Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Shuttle buttons (>,<)	Use shuttle buttons to move columns back and forth between Available Columns and Key Columns.
Up and Down buttons	Use to order the columns in the Key Columns field.
Sort	Sorts the Available Columns in alphabetical order.

10. Click **Next**.
11. Optionally, use the Conditional Relationship page to define a SQL WHERE clause to filter the relationship. See also DB2 Restrictions on Conditional Relationships.

Conditional Relationship Definition

Field	Description
Database	Choose the database to which the WHERE clause applies. You can have a different WHERE clause for as many of the vendors and versions as necessary. At runtime, Structured Data Manager can determine the database vendor and version, and apply the clause that you assigned to that vendor and version. Choose Any for the default WHERE clause. The Any WHERE clause is used for any vendor and version that does not have an assigned WHERE clause. For example, if you do not assign a WHERE clause to Oracle or any of its versions, the Any WHERE clause will be used when deploying against Oracle. If the clause has vendor or version specific calls or syntax in it, expand the Any node until you see the necessary option for the clause and select it.
WHERE	Type or edit the WHERE clause.
Columns	Double-click the column name to insert the name into the WHERE clause.
<<Insert	Alternatively, highlight a column and use the <<Insert button to move it to the WHERE clause.

12. Click **Finish**.

1.3.2.7. Managing associated tables

If the WHERE clause for a conditional relationship or rule refers to tables that are not included in the model, you need to include such tables as associated tables. In Designer, click Associated Tables and add these tables to the list. For tables in the associated tables list, a GRANT SELECT is automatically issued to ensure that they are accessible to the cartridge when it is deployed.

See DB2 Restrictions on Conditional Relationships.

1. From the Rule dialog box or Conditional Relationship page, click **Associated Tables**. The Associated Tables List dialog displays.
2. You can populate the list in one of two ways:
    1. Click **Automatic**. Designer discovers the tables referenced in your WHERE clause that are outside the model and adds them to the list.

       or

    1. Click **Add**. The Add Tables dialog box appears.
    2. Scan or search the tree to find the tables, select them and click **OK**

       Tip

       To remove tables from the Associated Tables List dialog, select the table and click **Remove**.
3. Click **OK**.

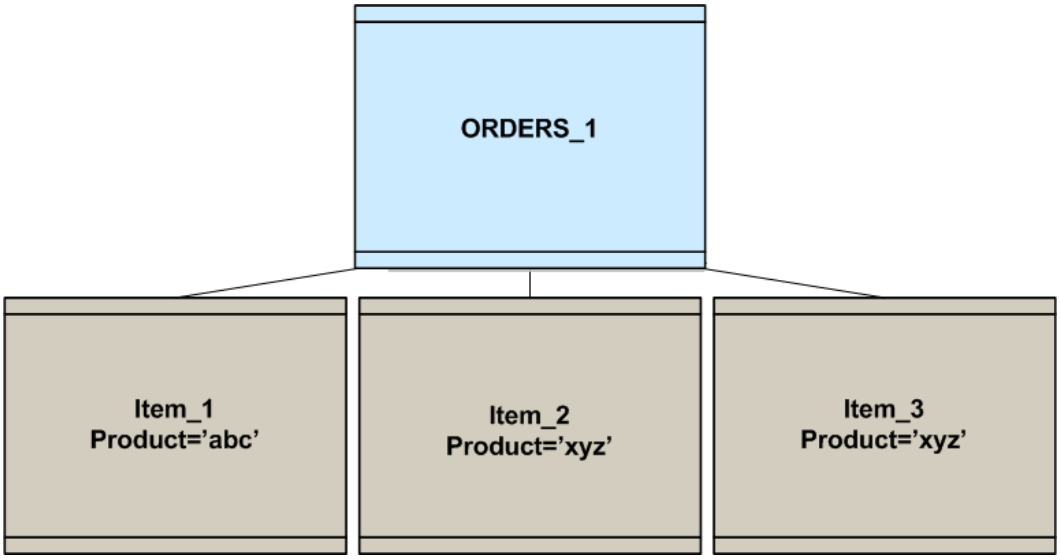
1.3.2.8. About rules

A rule is a condition defined by a SQL WHERE clause that determines whether or not an object is eligible for archiving or data access. Typically, you define rules in the model and then select the rule for use in a cartridge. In the case of a schema-based cartridge, where you have no model, a rule may be both defined and used in the cartridge.

When you create the rule, you specify whether it applies to archive, reload or data access cartridges. You may also specify that a rule only be used for Designer ttview and ignored during the normal execution of the cartridge in production. For example, you may want to limit the scope of data during development to speed performance for frequent ttview operations. When you move the business flow to production, though, you will want to automatically drop that development limitation on the data.

Rules can be defined as exclusive or inclusive. With exclusive rules, all rows in the model instance must meet the defined criteria. With inclusive rules, only one row must meet the criteria. Default rules are Exclusive. For example, for the model instance ORDERS_1 in Rule behavior in models: exclusive vs. inclusive, you could define a rule to find orders with items that are equal to abc:

Rule behavior in models: exclusive vs. inclusive



Rule Types

Rule type	Effect
Exclusive	The entire model instance ORDERS_1 is excluded because Item_2 and Item_3 do not meet the criteria to be archived. All item rows must have a product of 'abc'.
Inclusive	The entire model instance ORDERS_1 is included because at least one item (Item_1) meets the criteria to be archived.
Default	This option is Exclusive. For more information see the Exclusive rule type.

Database differences and parameters in rules

You must bear in mind that rules are just SQL strings that Structured Data Manager pushes down to the database, where parameters are converted to bind variables. The SQL with parameters resolved must be valid for the database running it.

For example, suppose that you need to add two months to a date in your rule. Different databases may exttss this function differently.

In Oracle, you would add two months as follows:

```
add_months("ORDER_HEADER".SHIPDATE,2)
```

By contrast, in PostgreSQL, you would specify:

```
"ORDER_HEADER".SHIPDATE + interval '2 months'
```

Notice how PostgreSQL requires single quotes around `2 months`. You need to consider this difference as you attempt to parameterize your rule. Your first thought might be to simply make the number 2 a parameter. This approach works in Oracle:

```
Oracle: add_months("ORDER_HEADER".SHIPDATE, :Min_Months_to_Retain)
```

However, in PostgreSQL, you run into a problem:

```
Postgres: ("ORDER_HEADER"."SHIPDATE"+ interval ':Min_Months_to_Retain months')
```

Because the parameter, `Min_Months_to_Retain`, is within single quotes, the database does not treat it as a variable and does not resolve its value. In addition, when the cartridge using the model is dropped into a business flow, it is not recognized as in-scope.

To avoid this issue, you need to use the cast function:

```
("ORDER_HEADER"."SHIPDATE" + cast(:Min_Months|| ' months' as interval))
```

1.3.2.9. Add rules

To add a rule to a model

1. Open a model.
2. Right-click a table and select **Add Rule**. The Rule dialog opens.
3. Type or select values for the following fields:

Rule Definition

Field	Description
Name	Type a name for the rule.
Usage	Choose from the list of usage options: • **Archive and Reload** indicates that the rule applies to both the archiving and reloading operations. • **Archive Only** indicates that the rule applies only when archiving. • **Data Access** indicates that the rule applies only to data access cartridges, which query and display data for users in the Web Console. • **Reload Only** indicates that the rule applies only when reloading. • **Preview Only** indicates that the rule applies only when previewing data in Designer. This type of rule is useful for limiting the data returned to improve performance during development.
Type	Choose from the list of options: • **Exclusive** indicates that, for any parent row to be eligible, all of its child rows must meet the defined criteria. • **Inclusive** indicates that, for any parent row to be eligible, only one of its child rows must meet the defined criteria. Inclusive rules must be associated with a driving table. • **Default** indicates that is Exclusive rule. For more information see the Exclusive rule type.
Customization	Choose from the list of options: • **Optional** • **Mandatory** • **Recommended** See Guidelines for Customizations for more information about customization.

Field	Description
Eligibility Analytics	Choose whether you want eligibility analytics to be enabled. Because eligibility analytics can impact the performance of a cartridge or business flow, you can choose to enable or disable them on a per rule basis. By default, eligibility analytics are turned off and you must explicitly enable them.Generally, it is recommended that you enable eligibility analytics on rules that will exclude a fairly small percentage of the data (for example, between 1% and 5%). Eligibility analytics are not supported in non-intrusive environments.
Database	Choose the database type to which the WHERE clause applies. You can have a different WHERE clause for as many of the vendors and versions as necessary. For example, you could associate a different WHERE clause with DB2, SQL Server, Oracle 8i, and Oracle 9i to account for differences in function calls. At runtime, Structured Data Manager can determine the database vendor and version, and apply the clause that you assigned to that vendor and version. Choose **Any** for the default WHERE clause. The Any WHERE clause is used for any vendor and version which does not have an assigned WHERE clause. For example, if you do not assign a WHERE clause to Oracle or any of its versions, the Any WHERE clause will be used when deploying against Oracle. If the clause has vendor or version specific calls or syntax in it, expand the Any node until you see the necessary option for the clause and select it.
WHERE	Type or edit the WHERE clause. If you use subqueries, you should fully qualify column names to ensure that they are distinct from the columns in the outer query.
Parameters and Columns	Highlight a parameter or column and use the <<Insert button to move it to the WHERE clause.
<<Insert	Use to move a highlighted parameter or column to the WHERE clause.
Parameters	Click Parameters to manage your parameter definitions (add or edit parameters). See Working with Parameters for more information.
Annotation	Optionally, click Annotation to enter a description or comment for the rule.

Field	Description
Validate	Click Validate to verify that the syntax of your WHERE clause is correct. Note that validation does not ensure that your rule will behave as you expect. Rather, it simply confirms that it meets the syntax requirements of a WHERE clause for the database you are using.
Associated Tables	See Managing Associated Tables.

4. Optionally, click **Parameters** to create or edit parameters. See Working with Parameters for more information about parameters.
5. Click **OK**.

1.3.2.10. Working with virtual unique and foreign keys

A virtual unique key is the equivalent of a database unique key. A virtual foreign key is similar to a database foreign key, but may also define the condition under which the relation is valid. Virtual unique and foreign keys are created in Designer when you add tables to a model, or from the Virtual Constraints dialog box. After you create the virtual unique or foreign key, it becomes referenceable and available when you add tables in Designer. Virtual unique and foreign keys are not created in the database, and are not deployed to the database.

When working with virtual unique and foreign keys, you should be aware of the following restrictions:

Because virtual unique and foreign keys are not validated, ensure that the ones you select are correctly defined.

To ensure optimal performance, you should consider creating an index for all columns that you plan to use as part of a virtual unique or foreign key. Otherwise, your archiving performance may be extremely slow. A non-unique backing index or a unique index that covers some of the columns in the key may be sufficient to ensure good performance.

This section includes:

Add a virtual unique key

To add a virtual unique key

1. Open a model.
2. Right-click a table and select **Virtual Constraints**. The Virtual Constraints dialog box opens.
3. Click **Add Unique Key**. The Virtual Unique Key dialog box opens.
4. Type or select the following:

Virtual unique key definition

Field	Description
Name	The default name of the key. You can accept it or type a new name.
Available Columns	These columns in the tables could be used to define the key. Highlight values in the Available Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Key Columns	These columns are the ones selected to define the key. Highlight values in the Key Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Shuttle buttons (>,<)	Use Shuttle buttons to move columns back and forth between Available Columns and Key Columns.
Up and Down buttons	Use to order the columns in the Key Columns list.
Sort	Displays the Available Columns in alphabetical order.

5. Click **OK**.
6. Click **Close** to close the Virtual Constraints dialog box.

Add a virtual foreign key

To add a virtual foreign key

1. Open a model.
2. Right-click a table and select **Virtual Constraints**. The Virtual Constraints dialog box opens.
3. Click **Add Foreign Key**. The Add Virtual Foreign Key dialog box opens.
4. Type a string or a partial string in the Filter field to choose a table from which you will select a database or virtual unique key that corresponds to the virtual foreign key you are creating. Note that, if no unique key exists, the wizard walks you through creating one.
5. Click **Search** and highlight the table you want.
6. Click **Next**.

   The Specify How to Use or Build Keys page opens. If the chosen table has no unique keys (database or virtual), the page appears with only the two bottom radio buttons.
7. Choose one of the following radio buttons:

Key specification

Selection	Description	Next Steps
Create a new foreign key associated with the following unique key or index	This option creates a new foreign key using existing unique key.	Highlight an existing unique key to be used with the new foreign key. Click **Next**. Continue with step 10.
Create a new unique key and foreign key	This option creates a new unique key and then a new foreign key that uses the just created unique key.	Click **Next**. Continue with step 8.
Create a new foreign key based only on a conditional relationship	If the chosen table has no unique key, you can create a foreign key based entirely on a relationship filter.	Click **Next**. Define a condition for the virtual foreign key. Click **Finish**.

8. Use the Unique Key page to define a new unique key:

Unique key definition

Field	Description
Name	The default name of the key. You can accept it or type a new name.
Available Columns	These columns in the tables could be used to define the key. Highlight values in the Available Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Key Columns	These columns are the ones selected to define the key. Highlight values in the Key Columns table. Use the shuttle buttons to move columns back and forth from the Available Columns and Key Columns lists as necessary.
Shuttle buttons (>,<)	Use Shuttle buttons to move columns back and forth between Available Columns and Key Columns.
Up and Down buttons	Use to order the columns in the Key Columns field.
Sort	Sorts the Available Columns in alphabetical order.

9. Click **Next**.

10. Use the Foreign Key page to define a new foreign key:

Foreign key definition

Field	Description
Name	Accept or type a new name for the key.
Available Columns	Highlight values in the Available Columns table. Use Shuttle button (>) to move columns from Available Columns to Key Columns.
Key Columns	Highlight values in the Key Columns table. Double-click or use Shuttle buttons to move columns from Available Columns to Key Columns or back again.
Shuttle buttons (>,<)	Use Shuttle buttons to move columns back and forth between Available Columns and Key Columns.
Up and Down buttons	Use to order the columns in the Key Columns field.
Sort	Sorts the Available Columns in alphabetical order.

11. Click **Next**.
12. Optionally, use the Conditional Relationship page to define a SQL clause to filter the relationship.

Conditional relationship definition

Field	Description
Database	Choose the database to which the WHERE clause applies. You can have a different WHERE clause for as many of the vendors and versions as necessary. For example, you could associate a different WHERE clause with DB2, SQL Server, Oracle 8i, and Oracle 9i o account for differences in function calls. At runtime, Structured Data Manager can determine the database vendor and version, and apply the clause that you assigned to that vendor and version. Choose **Any** for the default WHERE clause. The Any WHERE clause is used for any vendor and version for which does not have an assigned WHERE clause. For example, if you do not assign a WHERE clause to Oracle or any of its versions, the Any WHERE clause will be used when deploying against Oracle. If the clause has vendor or version specific calls or syntax in it, expand the Any node until you see the necessary option for the clause and select it.
WHERE	Type or edit the WHERE clause.
Columns	Double-click the column name to insert the name into the WHERE clause.
<<Insert	Alternatively, highlight a column and use the <<Insert button to move it to the WHERE clause.
Associated Tables	See See "Managing associated tables".

13. Click **Finish**.
14. Click **Close** to close the Virtual Constraints dialog box.

Edit a virtual unique key or virtual foreign key

To edit a virtual unique or virtual foreign key

1. Open a model.
2. Right-click a table and select **Virtual Constraints**. The Virtual Constraints dialog box opens.

   Tip

   You can also right-click a line in the model and choose Edit Virtual Foreign Key.

3. Select a virtual constraint from the list and click **Edit**.
4. Edit the virtual unique key or virtual foreign key as required and click **OK**.
5. Click **Close** to close the Virtual Constraints dialog box.

Delete a virtual key

To delete a virtual key

1. Open a model.
2. Right-click a table and select **Virtual Constraints**. The Virtual Constraints dialog box opens.

   Tip

   You can also right-click a line in the model and choose Replace Virtual Foreign Key Connection, if you want to simply change the selected foreign key for the relationship.

3. Select a virtual constraint from the list and click **Remove**.

1.3.2.11. Modify object properties

Object properties allow you to change the name of an object in the model. For example, you can rename a database table to something that is more meaningful to you for the creation of your cartridge.

This feature also allows you to set aliases and add annotations to the object.

1. Right-click a table use in the model and select **Properties**.
2. Modify the properties as needed.

Object properties

Property	Description
Table Name	The name of the table in the database. If you have mapped schemas through **Connection > Map Schemas**, you will see Mapped Table Name and Database Table Name instead of just Table Name.
Shape Name	The name you want displayed in the diagram. This name has no impact on the cartridge. It is for display purposes in the diagram only.
Table Alias	The alias to be used in rule WHERE clauses. (Changing the name will invalidate any existing rules that reference the previous alias.)
Data Movement Key	This key is used to join to the selection tables. You can select: **[Default]** to auto select a key at deployment. This is also the default value. the key you want to use as the Data Movement Key. for Oracle, **<Use Row ID>**.

1.3.2.12. Locate a table

This feature helps you locate model tables in the database navigator. If your database has many tables, you can use this feature to quickly find a table so that you can browse its details.

1. Right-click a table in the model editor or under the Model node in the Project Navigator and select **Locate Table**.

   The table is selected in the database navigator.

1.3.2.13. Drag and drop a table

This feature allows you to drag a table from the database navigator and attach it to any existing table in the model.

Note

You cannot add a transactional table or a chaining table to a lookup table.

To drag and drop a table

1. Highlight a table in the database navigator panel.
2. Drag the table over to the Model editor.
3. Drop the table on any existing table.
4. Select one of the following:
    - add *<table_name>* as a transactional table
    - add *<table_name>* as a lookup table
    - add *<table_name>* as a chaining table

From this point, the wizard behaves much the way it is described in the following sections, except it does not display the Choose Table dialog because the parent and child are already known to it.

- Add transactional tables
- Adding lookup tables
- Add chaining tables

1.3.3. Working with parameters

In Structured Data Manager, you can use parameters in your rules, thereby enabling users who run the jobs to change the values at runtime. In addition, you can create parameters that can be given values dynamically at runtime by code or at configuration time by administrators.

This section explains how to work with parameters:

- About parameters
- Create parameters
- Validate parameters
- Create lists of values for parameters
- Referencing Parameters
- Manage model compatibility

1.3.3.1. About parameters

In Structured Data Manager, there are three types of parameters:

- Runtime parameters have their values set at runtime by the user running the job. Runtime parameters tend to be best for operational values that tend to change with each execution of a job. For example, if your archive is based on a specified cutoff date, you most likely need to update that date every time you run the job.
- Configuration parameters have their values set by an administrator (someone who has repository privileges from Console) through the administrator interface. Typically, this type of parameter represents values that should be changed very infrequently, perhaps only at deployment time. A retention period defined by the legal department is a good example of a value that should not change very often and that only administrators would be allowed to change. You can access configuration parameters from the Web Console, **Business Flow Management > Tasks > System Parameters > Business Flows**.
- Dynamic parameters have their values set once by a Groovy script that runs at deployment time to obtain a value. These parameters are evaluated in the background at runtime whereas runtime parameters prompt for values in the Web Console or have them supplied on the command line. For example, this type of parameter can supply the type or version of a database or application, which can be obtained programmatically at deployment time.

Note

After a job starts running in the Web Console, all three types of parameters are treated the same way. Their values can be updated during the run but that will not be preserved after the run.

Regardless of a parameter's type, it can be referenced from several places within your archive project:

- WHERE clauses within your rules
- Groovy scripts in runtime and configuration parameter validations
- Groovy scripts in business flows
- SQL for runtime and configuration parameter list of values
- Interrupts in your business flows
- Custom selection programs in your cartridges
- Manage Compatibilities dialog box for the model (dynamic parameters only)

1.3.3.2. Create parameters

You can create parameters beforehand and then reference them, or, using SQL, you can create them dynamically as you reference them.

This section includes:

- Create or edit parameters from the Project Navigator
- Create or edit parameters using the Parameters button
- Create parameters automatically
- Validate parameters

Create or edit parameters from the Project Navigator

To create or edit parameters using Project Navigator

1. In Designer, from the Project Navigator, right-click the Parameters node and choose New Parameter.
2. Type or select values for the following Parameter definition properties:

Field	Description
Name	Enter a name for the parameter. In choosing parameter names, you must avoid reserved names. See Validating Parameters for a list of reserved names.
Parameter Type	Type of the parameter, choose one from the list of options: ◦ Runtime parameters get their values from the user at runtime. ◦ Configuration parameters are assigned values by the administrator and cannot be changed by the user at runtime. ◦ Dynamic parameters get their values from Groovy scripts. If you choose this type, the rest of the properties in the dialog are replaced by Data Type and a Code area, where you can enter your Groovy script.
Label	Label that appears on the Console Job Launcher page.
Data Type	Type of the parameter's value (Date, Number, or String). Dynamic parameters can only return string values.
Length	Maximum length allowed for the selected data type.
Default	Type a default parameter value.
Validation	For runtime and configuration parameters only, the type of validation you want to use for the parameter's values. Choose from the list of options: ◦ **None** ◦ **Groovy** means that you will supply a Groovy script to validate the parameter's value. ◦ **Mandatory** means that a parameter value must be supplied in order for the job to run. See Validating Parameters for more information.

Field	Description
List of Values	For runtime and configuration parameters only, the kind of list of values you want to use. Choose from the list of options: ◦ **None** ◦ **Static** means that you can use the adjacent browse button to create a hard-coded list of values for the parameter. ◦ **SQL** means that a SQL SELECT statement populates the list of values for the parameter. See Creating Lists of Values for Parameters.
Code (dynamic parameters only)	This area only appears when you choose a Parameter Type of Dynamic. Enter a Groovy script that populates the parameter's string value. The code must return a string value. You can use the parameter for compatibility checking. See Managing Model Compatibility for more information about compatibility.

3. Click **OK**.

Create or edit parameters using the Parameters button

To create or edit parameters using the Parameters button

1. In Designer, from either the Rule or Groovy Script dialogs, click **Parameters**.
2. To create a new parameter, click **Add**. To edit an existing parameter, click it in the list on the left of the dialog.
3. Type or select values for the fields as described in step 2.

   In choosing parameter names, you must avoid reserved names. See Validating Parameters for a list of reserved names.
4. Click **OK**.
5. To reference the parameter, see Referencing Parameters.

Tip

To delete a parameter, right-click it in the Project Navigator under the Parameters node and choose Delete. When you delete a parameter, ensure that you also delete all references to it throughout your project.

Create parameters automatically

To create parameters automatically

1. In Designer, referencing a parameter in SQL that does not already exist causes a new parameter to be automatically created with default properties. See Referencing Parameters for information on how to reference parameters.
2. After you reference the new parameter from SQL, go to the Project Navigator and expand the Parameters node to find the automatically created parameter.

> **Tip**
>
> Some dialog boxes, such as the Rule dialog box, contain a Parameters button, which you can click to see the parameters as well. If you're not in such a dialog, you need to accept the dialogs and go to the Project Navigator.

3. Double-click the automatically created parameter. Confirm that its properties are appropriate for your usage. See step 2 for more information about parameter definition properties.
4. Click **OK**.

1.3.3.3. Validate parameters

For runtime and configuration parameters, you can choose whether and how to validate the values entered by the user. By validating parameters, you ensure that the user enters a valid value before attempting to run the business flow that references the parameter. If the user enters an invalid value, the job will not run. If you choose not to validate parameter values, then an invalid value is not discovered until the job fails.

To validate parameters

1. Go to the Parameter Definition dialog box by right-clicking the parameter in the Project Navigator and choosing **Edit Parameter**.
2. In the **Validation** field, choose one of the following values:
    1. **None** means that you want no validation performed on parameter values.
    2. **Mandatory** means that a value must be entered for the parameter in order for the business flow to run. If a value is not entered, then the business flow will not run. In general, if you choose Mandatory, it is good practice to also enter a default value for the parameter. That way, if the user does not enter a value, the business flow will run with the default value rather than fail.
    3. **Groovy** means that you are providing a Groovy script to check the value. If you choose this option, you must also click the adjacent browse button to create a Groovy script. If the parameter fails validation, the script must return a short error message that describes the reason for the validation failure. If the parameter passes validation, the script must return nothing, or a null or empty string. An example Groovy validation script might look something like the following:

        Example

        ```
        import java.util.Date
         import groovy.time.TimeCategory;
         use([TimeCategory]) {
         Date now = new Date()
         Date maxDate = now - retention.months
         if (cutoff_dt > maxDate){
         return "Date must be older than ${retention} months."
         }
         }
        ```

        where `cutoff_dt` is a runtime parameter and `retention` a configuration parameter.

1.3.3.4. Create lists of values for parameters

Lists of values for runtime and configuration parameters provide another method for reducing the likelihood of user input errors and restricting the possible values the user may enter. You can create a fixed list of values by manual entering values and labels. Alternatively, if the possible values are stored in a database table somewhere, you can create a dynamic list of values by using a SQL SELECT statement to populate the list. The latter option tends to be the most efficient method because it ensures that the list of values will be automatically kept in sync with the database. You do not have to worry about updating the list every time the possible values change.

Tip

Lists of values do not apply when you run your business flow from the command line. To restrict the values entered from the command line, you should use validation Groovy scripts. See Validating Parameters.

Create a static list of values

To create a static list of values

1. Go to the Parameter Definition dialog box by right-clicking the parameter in the Project Navigator and choosing **Edit Parameter**.
2. In the List of Values field, choose **Static**.
3. Click the Browse button to the right of List of Values.
4. For Id, enter a valid value for the parameter.
5. For Label, enter the label that you want the user to see in the list. For example, suppose the parameter values are the start and close dates of quarters in your fiscal years. For Label, you might enter something like Q1FY08 Start or Q1FY08 Close.
6. Click **Add**.
7. Repeat step 4 through step 6 until you have added all of the desired values to the list and click **OK**.

Create dynamic lists of values

To create a dynamic list of values

1. Go to the Parameter Definition dialog box by right-clicking the parameter in the Project Navigator and choosing **Edit Parameter**.
2. In the List of Values field, choose **SQL**.
3. Click the Browse button to the right of List of Values. The Dynamic List Definition dialog box displays.
4. For Database Connection, choose the connection (**Source**, **History**, **Repository**) against which you want to run the SELECT statement that populates the list.

   Source refers to the active database. History refers to the archive database.
5. Choose either the **Drop-Down List** or **Look-Up List** option. A drop-down list shows the list of values as a combo box with the field. A look-up list opens a dialog with the values and the user can enter a filter pattern.

   Tip

   If you choose Look-Up List, notice that a parameter named FilterPattern appears in the Parameters area. This parameter contains the search string entered by the user and you can reference this string from your SELECT statement. For example, you might use this string to further qualify your SELECT criteria and thereby reduce the size of the list.

   Use a drop-down list if the number of values is reasonably small.

   Use a look-up list if the number of values is very large.

6. For Database, choose the database to which the SELECT statement applies. You can have a different SELECT statement for as many of the vendors and versions as necessary. For example, you could associate a different SELECT statement with DB2, SQL Server, Oracle 8i, and Oracle 9i. At runtime, Structured Data Manager can determine the database vendor and version, and apply the clause that you assigned to that vendor and version.
    - Choose **Any** for the default WHERE clause. The Any WHERE clause is used for any vendor and version which does not have an assigned WHERE clause. For example, if you do not assign a WHERE clause to Oracle or any of its versions, the Any WHERE clause will be used when deploying against Oracle.
    - If the clause has vendor or version specific calls or syntax in it, expand the Any node until you see the necessary option for the vendor and version and select it.
7. For SQL, enter a SQL SELECT statement that populates the list of values for the parameter. The SELECT statement must return an `id` and a `label`. The `id` is the first return value, it will appear on the command line. Note that the data type of the `id` sets the data type for the parameter. The `label` is the second return value. It is displayed to the user in the drop-down list or look-up list in the Job Launcher in the Console.

    See Create lists of values for parameters for some SQL examples for your list of values.
8. Click **Validate** to confirm the syntax of your SELECT statement.
9. If you reference tables not included in your model, click **Associated Tables**.

    See also, Managing Associated Tables.
10. Click **OK**.

    Examples

    Following is an example SELECT statement with numeric values:

    ```
    ORDER BY 2
    ```

    where `cutoff_dt` is a runtime parameter.

    Following is an example SELECT statement with date values:

    ```
    SELECT TO_CHAR(SHIPDATE, 'YYYY.MM.DD') , TO_CHAR(SHIPDATE, 'DD-MON-YYYY') FROM ORDER_HEADER
    ```

    Note

    For the id (the first value in the SELECT statement), you must format the date value. For label, the formatting is optional. Note that Oracle requires MMM rather than MON.

1.3.3.5. Referencing Parameters

You can use Designer to reference parameters in your project by:

- Typing inside SQL `:parameter_name`. For example, the WHERE clause in a rule might reference a parameter named Min_Months_to_Retain as follows:

  ```
  add_months("ORDER_HEADER".SHIPDATE, :Min_Months_to_Retain)<sysdate
  ```

  > **Note**
  >
  > In SQL, if the parameter does not already exist, Structured Data Manager creates a parameter with default properties for you. See Creating Parameters Automatically for information on how to handle automatically created parameters.

- Typing `parameter_name` inside a Groovy script.

System parameters

The following table lists the system parameters built into Structured Data Manager, which you can reference within your projects.

> **Note**
>
> The system parameter names are reserved; you cannot create your own parameters with these names. Hence, when you reference a system parameter name, such as FilterPattern, Designer will not create a new parameter. It assumes that you meant to reference the system parameter instead.

System parameters

Parameter Name	Data type	Description
BUSINESS_FLOW_NAME	STRING	**Referenceable from: Groovy activity** The name of the business flow being run. This parameter is useful when querying the Structured Data Manager repository. It holds values such as Orders_BF or emp_BF.
BUSINESS_FLOW_VERSION	STRING	**Referenceable from: Groovy activity** The version of the business flow being run.
CURRENT_GROUP_RUN_ID	NUMBER	**Referenceable from: Groovy activity** The group identifier for the job at runtime. This parameter is useful when querying the Structured Data Manager repository. It holds values such as 1, 22, 34, or 42.
CURRENT_RUN_ID	NUMBER	**Referenceable from: Groovy activity** The identifier for the job at runtime. This parameter is useful when querying the Structured Data Manager repository. It holds values such as 1, 22, 34, or 42.
ENVIRONMENT_NAME	STRING	**Referenceable from: Groovy activity** The name of the environment in which the business flow runs.
FilterPattern	STRING	**Referenceable from: SQL popup list of values** The value entered by the user in the list of values. You can use this parameter to narrow your list of values based upon what the user has entered. Note that this parameter only provides the value entered by the user in the list. It does not append any special characters like % or *.

Parameter Name	Data type	Description
INTF_DB	groovy.sql.Sql	**Referenceable from: any Groovy** The database connection for the interface schema.
INTF_RELOC_DB	groovy.sql.Sql	**Referenceable from: any Groovy** The database connection for the relocation schema.
OBT_INTF_DBTYPE	groovy.sql.Sql	**Referenceable from: any Groovy** The database type, for example, DB2, Oracle, or SQL Server.
OBT_INTF_DBVER	groovy.sql.Sql	**Referenceable from: any Groovy** The database version, for example, 10g, 11g, 2008, 12.5, or 2.5.
PRODUCT_HOME	groovy.sql.Sql	Referenceable from: any Groovy The install directory of Structured Data Manager.
REPOS_DB	groovy.sql.Sql	**Referenceable from: any Groovy** The database connection for the repository. You can use it to connect to the repository database where Structured Data Manager stores its metadata.
UPDATE_ROWCOUNT	groovy.sql.Sql	**Referenceable from: any Groovy** The rowcount for the currently running Groovy job. Useful when you have a Groovy script in business flows that is moving rows and you need to update the row count for a particular job run.

Symbolic schema names

When specifying the schema name of tables, you can use symbolic notation for the schema names rather than hard coding the actual names. These symbolic schema names are valid in SQL fields (for example, in a rule) or Groovy scripts. Furthermore, any schema mappings you have created in Designer are also applied to these tokens.

For queries against the Oracle archive database:

`${HIST.<source_schema_name>.ORDER_HEADER}`

For queries against the SQL Server archive database:

`${HIST.<database>.<source_schema_name>.ORDER_HEADER}`

For queries against the Oracle active database:

```
${SOURCE.<source_schema_name>.ORDER_HEADER}
```

For queries against the SQL Server active database:

```
${SOURCE.<database>.<source_schema_name>.ORDER_HEADER}
```

Note

In database to file movement, only the SOURCE and FROM schema names are applicable because there is no HISTORY schema.

In the case of database to database movement, you can use the TO and FROM schema names to create rules applicable to both archive and reload cartridges. When archiving, the TO schema name points to the archive schema and, when reloading, it points to the active schema. Similarly, the FROM schema name points to the active schema for archiving and the archive schema for reloading.

For queries against the Oracle database to which you are moving data:

```
${TO.<source_schema_name>.ORDER_HEADER}
```

For queries against the SQL Server database to which you are moving data:

```
${TO.<database>.<source_schema_name>.ORDER_HEADER}
```

For queries against the Oracle database from which you are moving data:

```
${FROM.<source_schema_name>.ORDER_HEADER}
```

For queries against the SQL Server database from which you are moving data:

```
${FROM.<database>.<source_schema_name>.ORDER_HEADER}
```

Note

For database to file movement, only the FROM schema name is applicable.

1.3.3.6. Manage model compatibility

Each model in your project can have one or more dynamic parameters associated with it to verify the compatibility with the archive environment. If the parameter returns a string of false, then the cartridge referencing the model will not deploy or run and throw an error. For example, the script for the dynamic parameter could return false for Oracle 10.2 and true for Oracle 11.1 to indicate that a cartridge referencing the model can only deploy and run against Oracle 11.1.

To choose dynamic parameters for the evaluation of model compatibility

1. In the Project Navigator, right-click a model node and choose **Manage Compatibilities**.
2. In the list, select the dynamic parameters that you want to use for model compatibility. Note that only dynamic parameters appear in the list. If the list is empty, then you do not have any dynamic parameters. The parameter must return a string of true or false.
3. Click **OK**.

1.3.4. Preview models and cartridges

Preview is a feature of Designer that you can use to view the data eligible for archiving in your model-based cartridge. Preview enables you to confirm that your model or cartridge is behaving as expected before you deploy and run it.

This section includes:

1.3.4.1. About preview

Note
Preview creates temporary tables in your source database.

Preview provides a quick way to test your archive as you are building it. You need not wait until you deploy and run your cartridge to see if the model and cartridge work properly. It is best practice to preview your archive at each stage of development and preview it repeatedly until you are certain it performs as expected.

For example, you can preview your model before you create any rules and then preview it again after you add each rule. Preview identifies the rows that are excluded as well as which rule causes their exclusion. When you create your cartridge, you can preview its behavior as well.

With preview, you can determine:

- What data meets the criteria defined by the rules. You can choose at runtime which rules to enforce for that execution. In this way, you can check each rule by itself as well as in combination with the other rules.
- What data is excluded. Preview displays rows that will be excluded from the archive in red.
- Which rules prevent certain data from being archived. In preview, a column called Excluded By shows the name of the rule that excluded a particular row. If a row is included, this column is blank for that row.

You can run preview against models and cartridges with eligibility analytics on or off, and with any combination of rules applied. If you see differences between the preview for a model and a cartridge based on that model, it is most likely because the cartridge is not using the same combination of rules as the model.

If your model contains chaining logic, the data retrieved by preview will differ from the data that will be selected for archive by the cartridge.

If you have a large amounts of data and are frequently previewing, you can create rules that are Preview Only to improve performance.

Note
Preview is not available for read-only data sources.

For details about Preview table locations, see Work with projects and connections.

1.3.4.2. Create Preview rules

When in the development phase for your archive, a quick response time is helpful because you are frequently making model enhancements and previewing them. Limiting the data returned for testing purposes is often worthwhile. A preview rule enables you to define a rule that applies only in preview in Designer. The rule is not applied when you generate and deploy your cartridge for actual usage.

To create a Preview rule

1. Right-click a table in the model and select Add Rule from the pop-up menu.
2. For Usage, select **Preview Only**.
3. Define the other properties for your rule. For example, in Oracle, you might enter a WHERE clause such as this one to limit the number of rows returned:

   ```
   ORDER_ID in (100, 132, 5540)
   ```

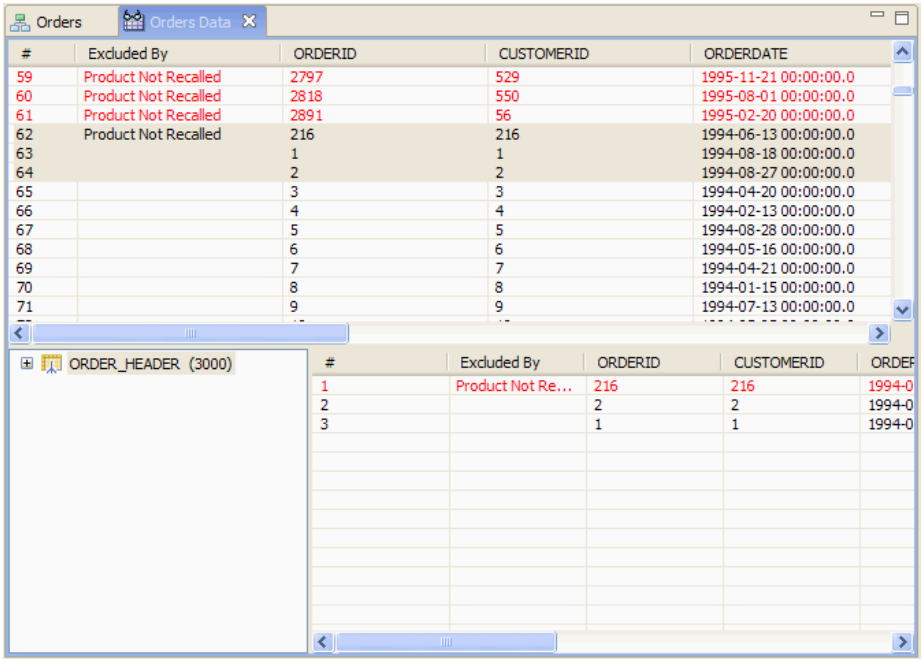
4. Click **OK**.
5. Click Preview in the toolbar to confirm that fewer rows are returned in the Preview tab.

1.3.4.3. Preview models and cartridges

To preview a model or cartridge

1. Open a model or cartridge in Designer.
2. Click **Preview** in the tool bar.
3. In the Parameters Values dialog, check the rules to apply and uncheck those to ignore for this run.
4. If you have any parameters, type/choose the values in the Parameter Values window.
5. Check or uncheck **Eligibility Analytics for all rules**, depending upon whether you want them. When checked, Designer creates analytics for each rule, regardless of what is set for the individual rules in the Rule dialog box. Enabling analytics causes the ineligible data to appear in red in the preview.
6. Click **OK**.

   The Preview pane opens.



   The top part of the window shows the rows of the driving table. Select a row or range of rows in the top part of the window to filter the rows displayed in the bottom part. Use Ctrl-click to select more than one row or clear the rows selected.

   The Excluded By column displays the rule that caused a row to be excluded. All rows that are excluded are displayed in red.

   Click on column headers to sort the rows by that value. For example, if you click the Excluded By column header, the rows are sorted according to the values of that column.

   Click and drag the column borders to resize the columns in the display.

   The model structure, including rules, is displayed in the lower left pane. Expand and collapse the node to view the tables and rule you want. Positive numbers in parentheses next to the driving table indicate the number of driving table rows included by that table. Negative numbers in parentheses next to a rule indicate the number of driving table rows excluded by the rule.

7. If you want to export the contents of the top table to a comma delimited file, right-click a row and choose **Export Data**. If you want to export the contents of the bottom table to a comma delimited file, right click on a table node in the lower left pane and choose **Export Details Data**.
8. Click **Refresh** from the toolbar to refresh the data from the database.
9. When you are done with this instance of preview, click the Close icon on the tab.

1.3.5. Working with cartridges

After you have your model and rules defined, you must consider how to deploy that archive. A cartridge is an instance of model- or schema-based eligibility criteria used to move or copy data from one location to another. The cartridge enables you to specify a set of characteristics to apply to a particular data movement activity, such as archive or reload.

This section includes:

1.3.5.1. About cartridges

There are several types of cartridges:

- Archive database to database cartridges

  Archive data from your active database to another database, typically on a less expensive platform with lesser performance characteristics.

- Reload database to database cartridges

  Reload data from your archive database back into your active database.

- Archive database to file cartridges

  Archive data from a database (active or archive) to a file, typically for the purpose of long term retention. The file may be in XML or comma separated values (CSV) format.

- Schema-based database to database cartridges

  Archive data from a specified set of tables (without use of a model) to another database. Schema-based cartridges are useful when you simply want to archive a set of tables without specifying relationships between them (no model).

  For example, suppose that you have a model that covers only portions of your application, leaving some tables unarchived. You might create schema-based cartridges for those tables not pulled in by your model.

- Schema-based database to file cartridges

  Archive data from a specified set of tables (without use of a model) to a file (XML or CSV). Schema-based cartridges are useful when you simply want to archive a set of tables without specifying relationships between them (no model).

  For example, if you are retiring an application, you might use schema-based cartridges to archive the tables before removing them from the database.

- Schema-based in-place masking cartridges

  Masks and unmasks source database using specific masking functions associated with different columns of selected tables without the use of a model.

- Data access cartridges

  Provide query access to retired data. By quickly adding data access rules and defining master-detail records, you can provide users with basic query access to retired data wherever it resides (file or database locations).

- In-place masking cartridges

  Provides a facility to mask or update the source database.

1.3.5.2. Create a new cartridge

To create a new cartridge

1. Select **Cartridges** in the Project Navigator, or select an existing model or cartridge.
2. Right click and select **New Cartridge**.
3. Type a name for the cartridge (for example, `salesorder`).

   The name you select appears in Designer, the Deployment Assistant and in the Web Console.
4. Select one of the following for **Type**:
   - **Database to Database**
   - **Database to File**
   - **Reload from Database**
   - **Data Access**
   - **In-Place Masking**
5. For Source, choose **Schema** or **Model**. For information about the difference between schema-based and model-based movement, see About Cartridges.

   > **Note**
   >
   > If you chose **Reload from Database** in the previous step, the Schema radio button is disabled because reloading from the database must be model-based.
   >
   > If the model contains chaining table then the designer won't allow to create **In-Place Masking** cartridge.
6. If you chose Model in the previous step, select a model from the list.
7. Click **OK** to create your new cartridge.

   The cartridge opens in the editor.
8. Edit the values. For more information, see the following sections:
   - Edit a database to database cartridge
   - Edit a schema-based database to database cartridge
   - Edit a database to file cartridge
   - Edit an in-place masking cartridge
   - Edit a schema-based database to database cartridge
   - Edit a schema-based database to file cartridge
   - Edit a data access cartridge

> **Note**
>
> If you edit a cartridge after deploying a business flow, then you must redeploy the business flow to see the latest changes that you made in the cartridge.

1.3.5.3. Edit a database to database cartridge

The database to database cartridge editor consists of the following tabs:

- Overview
- Operations
- Data Masking
- Custom Properties
- Custom Selection Program

Note

Database to database cartridges do not support non-intrusive environments. See the *Runtime Guide* for more details about setting up a non-intrusive deployment environment.

You can edit the information on each tab at any point during the cartridge editing process, as well as edit the model that is associated with the cartridge.

To edit a database to database cartridge

1. Expand Cartridges in the Project Navigator.
2. Select the database to database cartridge you want to edit.

   The Database to Database Cartridge dialog opens in the editor and displays the Overview tab.
3. Enter or select values for the following as necessary:

   Overview properties

Field	Description
Version	Enter a version number for the cartridge or accept the default.
Selection Rules	Select the rules to be applied to the cartridge. A model can have any number of rules, and you may or may not want to apply each of them to any particular cartridge.

4. Click the **Operations** tab. The Edit Operations dialog opens.

   Copy and Purge

   When using database to database, you must adhere to the following rules when selecting copy and purge options:

   - If your model uses a database constraint between a parent and a child, Designer does not allow you to purge the parent without purging the child.
   - Designer allows you to copy a child without copying the parent, but the cartridge will fail with a database constraint violation on the archive database unless you take some action. For example:
     - Copying the parent data through another cartridge run earlier in the same business flow.
     - Disabling the database constraint in the archive database.
5. For each table, choose the action to take on the eligible data.

   Partition-based delete

   Partition-based deletion deletes the entire partition rather than rows in the table. Partition-based deletion is much faster than deleting rows from a table because it swaps the partition segment with an empty table. In order to take advantage of partition based deletion:

- The source database is Oracle 11g or 12c.
- All the rows must be selected. You cannot use partition-based deletion if some rows are not selected.
- Partition-based deletion is supported for standard and advanced selection, but, for advanced selection, you must set the configuration parameter, USE_DATA_MOVEMENT_KEY, to false before the deployment of the business flow or make ROWID the data movement key in the model.

Copy and purge options

Action	Description
Copy	Copy the data from the active database.
Purge	Delete the data from the active database after it has been copied to the archive data store.
Partition-based	Perform the purge using partition-based deletion.

6. Select the **Data Masking** tab. The Edit Data Masking dialog opens and you can apply data masks to particular columns as necessary to protect privacy or hide confidential information, such as Social Security Numbers and credit card numbers. See Apply data masks.
7. Select the Custom Properties tab. The Custom Properties dialog opens.
8. In Custom Properties, you can optionally create, annotate, and delete name-value pairs that provide additional information about your archived data. Such information could be used by other applications to classify or otherwise operate on the archived data.
    1. Click **New** to create a new custom property.
    2. Enter a Name and Value for the property.
    3. Click **Annotation** to add a comment to the property.
    4. Use the **Delete** button to remove the currently selected property, if you do not need it.
9. Optionally, you can create a custom selection program for the cartridge. See Create custom selection programs.

See also

Generating and Deploying for information on generating and deploying your cartridge.

1.3.5.4. Edit a reload database to database cartridge

The reload database to database cartridge editor consists of the following tabs:

- Overview
- Operations
- Data Source
- Data Masking
- Custom Properties
- Custom Selection Program

Note

Reload cartridges will not create a table in the active database schema if it does not already exist. For example, if you try reloading rows from the ORDER_HEADER table and it does not already exist in the active database, then the business flow will fail. If you need to create the tables in the active database, you should use the upload feature of database to file.

To edit a reload database to database cartridge

1. Expand **Cartridges** in the Project Navigator.
2. Select the reload database to database cartridge you want to edit. The Reload Cartridge dialog opens in the editor and displays the Overview tab.
3. Enter or select values for the following as necessary:

Overview properties

Field	Description
Version	Enter a version number for the cartridge or accept the default.
Selection Rules	Select the rules to be applied to the cartridge. A model can have any number of rules, and you may or may not want to apply each of them to any particular cartridge.

4. Select the **Operations** tab. The Edit Operations dialog opens and behaves similarly to the Edit Operations dialog for a database to database cartridge, Edit a database to database cartridge.
5. Select the Data Source tab. The Edit Data Source dialog opens. From this dialog, you can choose the source from which to reload:
   - **Source** indicates that the data already resides in the active (source) tables (for example, a lookup table) and need not be reloaded.
   - **History** indicates that the data resides in the archive (target) tables and should be reloaded from there.
   - **Union** indicates that the data may reside in either the active or archive tables and Structured Data Manager should determine which is the appropriate source to use.
6. Select the **Data Masking** tab. The Edit Data Masking dialog opens and you can apply data masks to particular columns as necessary to reverse the masks used when you archived the data. See Apply data masks.

Tip

If you want to reverse masks upon reload, you must select reversible masks and specify the exact same masks on the same columns as the archive cartridge. Otherwise, the masks will not be reversed. See Data Masking and Reload/Undo.

7. Select the **Custom Properties** tab. The Custom Properties dialog opens and behaves similarly to the Custom Properties dialog for a database to database cartridge, Edit a database to database cartridge.
8. Optionally, if necessary you can create a custom selection program for the cartridge. See Create custom selection programs.

See also Generating and Deploying for information on generating and deploying your cartridge.

1.3.5.5. Edit a database to file cartridge

The database to file cartridge editor consists of the following tabs:

- Overview
- Operations
- Data Source
- Data Masking
- Validation
- Column Inclusion
- Name Override
- Custom Properties
- Custom Selection Program

Note

Only database to file cartridges support non-intrusive environments. Specifying a non-intrusive environment allows you to copy or archive data from read-only sources, especially in cases where the data is associated with older technologies that might not support basic SQL statements such as DELETE, or when the database administrator or company policy prohibits write access to the production environment. Non-intrusive environments are typically used in conjunction with JDBC.

When you set up a non-intrusive deployment environment, the Validation and File Indexes remain visible in the cartridge editor, however, any criteria defined within these two tabs will not be executed during deployment because no interface schema is created for a non-intrusive environment.

See the *OpenText™ Structured Data Manager Runtime Guide* for more details about creating and managing non-intrusive environments.

Tip

You cannot create a database to file cartridge based upon a model with a chaining table.

You can edit the information on each tab at any point during the cartridge editing process, as well as edit the model that is associated with the cartridge.

To edit a database to file cartridge

1. Expand Cartridges in the Project Navigator window.
2. Select the database to file cartridge that you want to edit.

   The Database to File Cartridge dialog opens in the editor.
3. Enter or select values for the following as necessary:

Overview Properties

Field	Description
Version	Enter a version number for the cartridge or accept the default.
Indexing	Choose an indexing cartridge from the list or leave it as [None].
Selection Rules	Select the rules to be applied to the cartridge. A model can have any number of rules, and you may or may not want to apply each of them to any particular cartridge.

4. Select the **Operations** tab. The Edit Operations dialog opens.

   Copy and purge

   When using database to file, you must adhere to the following rules when selecting copy and purge options:

   - If your model uses a database constraint between a parent and a child, Designer does not allow you to purge the parent without purging the child.
   - Due to the hierarchical structure of XML, you must copy the parent in order to copy a child.

5. On a table by table basis, decide what action to take on the eligible data.

Copy and purge options

Action	Description
Copy	Copy the data from the active database.
Purge	Delete the data from the active database after it has been copied to the archive data store.

6. Select the **Data Source** tab. The Edit Data Source for 3-tier Archive dialog opens.

   In a 3-tier topology, lookup tables can reside in the source database, the history database, or partially in source and partially in history. Hence, if you have a lookup table in your model, then you must indicate to Structured Data Manager the location from which it should obtain the lookup table (source, target or both).

   In cases where a table is referenced as both a lookup and transactional table in the same model, the table will always reside in the target database and you do not need the Data Source tab.

   From this dialog, choose the source you wish to use:

   - **Source** indicates that the data will be taken from the active (source) tables.
   - **History** indicates that the data will be taken from the archive (target) tables.
   - **Union** indicates that the data may reside in either the active or archive tables and Structured Data Manager should determine which is the appropriate source to use.

     See Data source for 3-tier archivesfor details about specifying the data source in a 3-tier archive.

7. Select the **Data Masking** tab. The Edit Data Masking dialog opens and you can apply data masks to particular columns as necessary to protect privacy or hide confidential information, such as Social Security Numbers and credit card numbers. See Applying Data Masks.

8. Select the **Validation** tab. The Edit Validation dialog displays.

   For database to file, you can validate that a value has not changed between selection time and deletion time. Use the Validation tab to select the columns you want to compare when the cartridge is run. If the values have changed between selection and deletion time, the data will not be deleted.

   You cannot validate LOB columns. Furthermore, if a column is validated, it must also be included on the Column Inclusion tab.

9. Select the **Column Inclusion** tab. The Edit Column Inclusion dialog opens.

   Tip

   If your indexing cartridge indexes a column or you validate it from the Validation tab, you cannot exclude it from the Column Inclusion tab.

   Use this tab to specify how table columns are included or excluded by your cartridge.

   Column inclusion options

Select	To
Include All	Archive all the columns found at deployment even if the column was not part of the table definition during the design of the cartridge.
Include Columns	Ensure that only the columns you specify are archived by the cartridge.
Exclude Columns	Ensure that specific columns are excluded from the archive. This option automatically archives all the other columns found at installation and their data, even if they were not part of the table definition during design of the cartridge.

   Tip

   Include All is not equivalent to selecting **Include Columns** and explicitly checking all columns. The former includes all columns found at deployment time. The latter only includes the columns you selected at development time.

   For example, if your development database contains only a subset of the columns in your production database, then selecting Include Columns with all columns checked may not archive all the columns in your production database.

10. Select the **Name Override** tab. The Edit Name Override dialog displays.

    Use this tab to define the element names to be used in the XML file. By default the element names are identical to the table and column names.

    The Name Override alters how a table or column name is represented in the XML document. A Name Override is necessary whenever a table or column name is not a valid XML element name, for example, multi-byte character sets (MBCS).

    If you wish to fix all the XML tags for all the tables in a cartridge according to your specifications, then click Fix all.

    If you wish to restore all default values, then click Clear all.

> **Tip**
>
> If you create a name override for a column, the column is included in the XML output, regardless of whether or not you excluded the column on the Column Inclusion tab.

Related information

See http://www.w3.org/TR/REC-xml/#NT-Name for information on valid XML element names.

1. Expand the database in the Database Name list, and select a column or table name.
2. Edit the value in the XML Name field so that the name is compatible with XML element naming conventions.

11. Select the **Custom Properties** tab. The Custom Properties dialog opens and behaves similarly to the Custom Properties dialog for a database to database cartridge, see Edit a database to database cartridge.
12. Optionally, you can create a custom selection program for the cartridge. See Create custom selection programs.

> **Tip**
>
> Each tab of the cartridge editor has its own Annotation button to enable you to enter comments. See Annotating Projects for more information.

See also Generating and Deploying for information on generating and deploying your cartridge.

Data source for 3-tier archives

For a database to file cartridge in a 3-tier environment, you need to fill out the Data Source tab. The Data Source tab specifies where the data should come from for lookup tables because lookup tables might exist on the source database, the target database, or partially on each.

The default data source for a lookup table is Source; however, when the table is used as both a lookup and managed table, only tables with History or Union (non-default) data sources are displayed. If a table is used as both a lookup table and a managed table, then the data source is always History.

For example, the model Orders has two transactional tables: ORDER_HEADER and ORDER_LINE, along with two lookup tables: CUSTOMER and PRODUCT. In this case, a database to database cartridge would archive the data from Source to History (tier 1 to tier 2). A database to file cartridge archives the data from History to files (tier 2 to tier 3). Here, you want the lookups in the database to file cartridge to come from Source because there is no data for the lookups in History.

Say you introduce a new database to database cartridge, Customers_D2D, that archives the CUSTOMERS table. Here, you would need to change the data source option in the cartridge Orders_D2F: the data source is History if the data is exclusively in the 2nd tier, or Union if it can be in both tier 1 and tier 2.

There is no data source option for schema-based cartridges because they do not contain lookups.

Output files

When you archive data to file, you can archive it as XML or CSV files. The generated XML or CSV files are described in this section.

- XML Output Files
- CSV Output Files

XML output files

When you archive to XML, the following files are created:

File	Description
summary XML	Contains summary information about the archive process that was run to create the files. This information includes: • what files were created • what data was archived • where the data was archived from • how much data was archived
summary XSD	XML Schema Definition file that defines the structure and makeup of the summary XML file.
group XML	The group XML file contains the following information: • information about the archive process used to generate the archived data within each particular XML file • the archived database data in XML form The number of group XML files depends on the data movement batch size. For example, if you set the batch size to 100, and there are 1000 driving table rows being archived, then 10 group XML files are generated. See the *OpenText™ Structured Data Manager Runtime Guide* for more information.
group IDX	The group IDX file contains the indexing directives associated with the group XML data file under a similar name. The group IDX file can be submitted to an Knowledge Discovery (IDOL) server using the IDOL server's DREADD command. A group IDX file is created for each group XML file if the job parameter Generate IDOL DRE is true. If the job parameter is false, no group IDX files are created. Each `attInstance` in the group XML file will have an associated set of indexing directives that associate the cartridge's file index columns to the `attInstance guid`. This allows IDOL searches to be expressed using the extracted data that returns results pointing to a specific `attInstance guid`. File indexes are defined for a cartridge in the File Indexes tab of the cartridge editor.
group XSD	XML Schema Definition file that defines the structure and makeup of the group XML file.

All the information needed to interpret the structure of the archived data is contained in these files, making the XML stand-alone for many years to come.

CSV Output Files

When you archive to CSV, the following files are created:

CSV output file types

File	Description
summary XML	Contains summary information about the archive process that was run to create the files. This information includes: • what files were created • what data was archived • where the data was archived from • how much data was archived
summary XSD	XML Schema Definition file that defines the structure and makeup of the summary XML file.
group XML	The group XML file contains a list of all of the CSV files for the group.
group IDX	The group IDX file contains the indexing directives associated with the group XML data file under a similar name. The group IDX file can be submitted to an IDOL server using the IDOL server's DREADD command. A group IDX file is created for each group XML file if the job parameter Generate IDOL DRE is true. If the job parameter is false, no group IDX files are created. For non-intrusive environments, the group IDX file contains indexing directives as described for XML output in CSV output files . For standard environments, each group IDX file will associate all the file index columns defined for the cartridge to the group XML file. This allows IDOL searches to be expressed using the extracted data that return results pointing to the group XML data.
group CSV	The group CSV file contains the he archived database data in CSV format. Each group has a set of CSV files for each table use. For example, if you have 10 groups and 3 table uses, you will have 30 group CSV files. See the *OpenText™ Structured Data Manager Runtime Guide* for more information.

1.3.5.6. Edit a model-based in-place masking cartridge

The in-place mask cartridge editor consists of the following tabs:

- Overview
- Operations
- Data Masking
- Custom Properties

You can edit the information on each tab at any point during the cartridge editing process, as well as edit the model that is associated with the cartridge.

To edit an in-place masking cartridge

1. Expand **Cartridges** in the Project Navigator window.
2. Select the in-place masking cartridge that you want to edit.

   The In-Place Masking Cartridge (IPMC) page opens in the editor.
3. Enter or select values for the following as necessary:

Overview Properties

Field	Description
Version	Enter a version number for the cartridge or accept the default.
Selection Rules	Select the rules to be applied to the cartridge. A model can have any number of rules, and you may or may not want to apply each of them to any particular cartridge.

4. Click the **Operations** tab. The Operations page opens with an Copy option.

   Note

   For IPMC feature, by default the copy option is selected and cannot be changed.
5. Select the **Data Masking** tab. The Edit Data Masking dialog opens and you can apply data masks to particular columns as necessary to protect privacy or hide confidential information, such as Social Security Numbers and credit card numbers. See Applying Data Masks.

   In case of IPMC as its name suggests, the data is updated to the source database. This is the difference between database to file cartridge and IPMC.
6. Click the **Column Inclusion** tab. The **Edit Column Inclusion** page opens.

   Use this tab to specify how table columns are included or excluded by your cartridge.

   Note

   It is recommended to always select **Include Columns** option to achieve better performance in data masking. With this, all the masking columns are included by default, hence the user needs to select only the additional columns required for context masking and the columns which are part of virtual constraint conditions.

   Do not select **Include All** option unless all the columns are required for masking.
7. Select the **Custom Properties** tab. The Custom Properties dialog opens and behaves similarly to the Custom Properties dialog for a database to database cartridge, see Editing a Database to Database Cartridge.

Tip

Each tab of the cartridge editor has its own Annotation button to enable you to enter comments. See Annotating Projects for more information.

Note

Look-up tables are not allowed to mask using IPM cartridge because of that in IPMC data masking tree, look-up table related information is not provided. Key (Primary, foreign, composite) columns are not allowed to mask because it is the violation of data integrity.

As of now, IPM cartridge supports only primary keys as Data Movement Key. If primary key exists then 'Default' option can be selected as Data Movement Key. Use of 'Row Id' as Data Movement Key is not supported.

While running the In-Place Masking business flow, user must ensure to disable the AQS cache from the databse to file system parameters. Also, the Destination Location for that business flow must be a **LOCAL** filesystem location.

See also Generating and deploying for information on generating and deploying your cartridge.

1.3.5.7. Edit a schema-based database to database cartridge

To edit a schema-based database to database cartridge

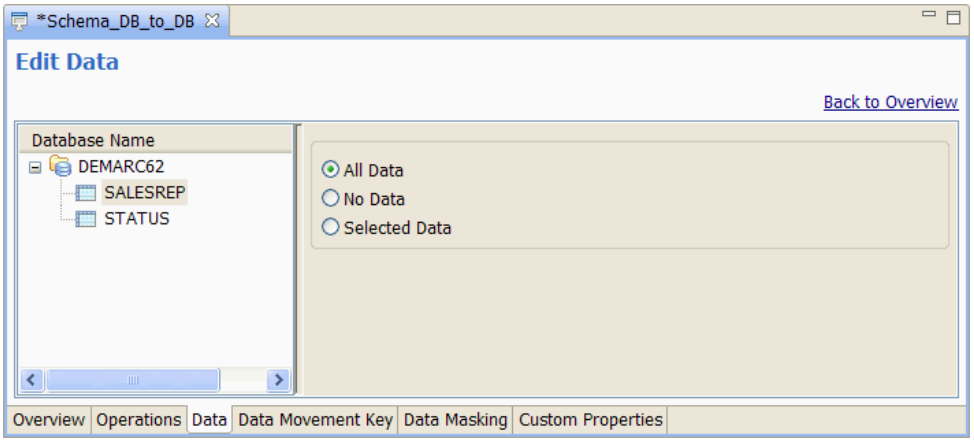
1. Expand Cartridges in the Project Navigator window.
2. Select the schema-based database to database cartridge that you want to edit. The Database to Database cartridge dialog opens in the editor.
3. Enter or select values for the following as necessary:

Overview properties

Field	Description
Version	Enter a version number for the cartridge or accept the default.

4. Click **Add** to display the Add Tables dialog box and select tables to be added to the archive.
5. Click **OK**.
6. If necessary, select tables for removal and click **Remove**.
7. Select the **Operations** tab. The Edit Operations dialog opens.

   See also Copy and Purge.
8. Select the **Data** tab.



9. Select one of the tables listed on the left.
10. Choose one of the radio buttons:
    - **All Data** indicates that the table will be created in the archive database and populated with data.
    - **No Data** indicates that the table will be created in the archive database but not populated with any data. This option is useful in cases where you need the tables present in order for the application to run, but you do not necessarily need the data.
    - **Selected Data** indicates that the table will be created in the archive database and a WHERE clause used to populate it with selected data from the active database.

      If you choose this radio button, type or select values for the following as necessary:

WHERE Clause for Choosing Data

Field	Description
Database	Choose the database to which the WHERE clause applies. You can have a different WHERE clause for as many of the vendors and versions as necessary. For example, you could associate a different WHERE clause with DB2, SQL Server, Oracle 8i, and Oracle 9i. At runtime, Structured Data Manager can determine the database vendor and version, and apply the clause that you assigned to that vendor and version. Choose **Any** for the default WHERE clause. The Any WHERE clause is used for any vendor and version which does not have an assigned WHERE clause. For example, if you do not assign a WHERE clause to Oracle or any of its versions, the Any WHERE clause will be used when deploying against Oracle. If the clause has vendor or version specific calls or syntax in it, expand the Any node until you see the necessary option for the clause and select it.
WHERE	Type or edit the WHERE clause.
Parameters and Columns	Double-click the parameter or column name to insert the name into the WHERE clause.
Insert	Alternatively, highlight a column and use the **<<Insert** button to move it to the WHERE clause.
Validate	Click **Validate** to verify that the syntax of your WHERE clause is correct. Note that validation does not ensure that your rule will behave as you expect. It simply confirms that it meets the syntax requirements of a WHERE clause for the database you are using.

11. Select the **Data Movement Key** tab.
12. Choose an option from the Key list of values:
    - Unique keys in the table
    - **[Default]** meansStructured Data Manager will choose a key for you at deployment.
    - **[Virtual Key]** means you want to use a virtual key and will specify the columns to use.
    - **[Use Row ID]** (Oracle only)
13. Select the **Data Masking** tab. The Edit Data Masking dialog opens and you can apply data masks to particular columns as necessary to protect privacy or hide confidential information, such as Social Security Numbers and credit card numbers. See Applying Data Masks.
14. Select the **Custom Properties** tab. The Custom Properties dialog opens and behaves similarly to the Custom Properties dialog for a database to database cartridge, see Editing a Database to Database Cartridge.

1.3.5.8. Edit a schema-based database to file cartridge

To edit a schema-based database to file cartridge

1. Expand Cartridges in the Project Navigator window.
2. Select the schema-based database to file cartridge that you want to edit. The Database to File cartridge dialog opens in the editor.
3. Enter or select values for the following as necessary:

Overview Properties

Field	Description
Version	Enter a version number for the cartridge or accept the default.

4. Click **Add** to display the Add Tables dialog box and select tables to be archived or copied.

Note

You can extract data from different database schemas (using different access rights) by creating a multi-source data model–this enables you to extract the data from all the schemas at once. Access rights only have an impact if you build the model in Designer. If you build the model in Designer, then you would need one account with read access to all the schemas you wish to include.

Tip

If the schemas are located in different databases, then you can use either views or database links to access the data (although this can impact performance), or, you can create multiple models or cartridges and combine them into a single business flow and run them sequentially.

Tip

If you build the model in Designer, then you would need one account with read access to all the schemas you wish to include.

5. Click **OK**.
6. If necessary, select tables for removal and click **Remove**.
7. Select the **Operations** tab. The Edit Operations dialog opens.

   See also Copy and Purge
8. Select the **Data** tab.
9. Choose one of the radio buttons:
   - **All Data** indicates that the table will be created in the archive data store and populated with data.
   - **No Data** indicates that only XSD files are generated. No data files are generated in this case. This option is useful in cases where you need the tables present in order for the application to run, but you do not necessarily need the data.
   - **Selected Data** indicates that the table will be created in the archive data store and a WHERE clause used to populate it with selected data from the active database.

     If you choose this radio button, type or select values for the following as necessary:

WHERE Clause for Choosing Data

Field	Description
Database	Choose the database to which the WHERE clause applies. You can have a different WHERE clause for as many of the vendors and versions as necessary. For example, you could associate a different WHERE clause with DB2, SQL Server, Oracle 8i, and Oracle 9i. At runtime, Structured Data Manager can determine the database vendor and version, and apply the clause that you assigned to that vendor and version. Choose Any for the default WHERE clause. The Any WHERE clause is used for any vendor and version which does not have an assigned WHERE clause. For example, if you do not assign a WHERE clause to Oracle or any of its versions, the Any WHERE clause will be used when deploying against Oracle. If the clause has vendor or version specific calls or syntax in it, expand the Any node until you see the necessary option for the clause and select it.
WHERE	Type or edit the WHERE clause.
Parameters and Columns	Double-click the column name to insert the name into the WHERE clause.
Insert	Alternatively, highlight a column and use the **<<Insert** button to move it to the WHERE clause.

10. Select the **Data Movement Key** tab.
11. Choose an option from the Key list of values:
    - Unique keys in the table
    - **[Default]** means Structured Data Manager will choose a key for you at deployment.
    - **[Virtual Key]** means you want to use a virtual key and will specify the columns to use.
    - **[Use Row ID]** (Oracle only)
12. Select the **Data Masking** tab. The Edit Data Masking dialog opens and you can apply data masks to particular columns as necessary to protect privacy or hide confidential information, such as Social Security Numbers and credit card numbers. See Apply data masks.
13. Select the **File Indexes** tab. The Edit File Indexes dialog behaves similarly to the Edit File Indexes dialog for a database to file cartridge, see Edit a database to file cartridge.
14. Select the **Column Inclusion** tab. The Edit Column Inclusion dialog behaves similarly to the Edit File Indexes dialog for a database to file cartridge, see Edit a database to file cartridge.
15. Select the **Name Override** tab. The Edit Name Override dialog behaves similarly to the Edit Name Override dialog for a database to file cartridge, see Edit a database to file cartridge.
16. Select the **Custom Properties** tab. The Custom Properties dialog opens and behaves similarly to the Custom Properties dialog for a database to database cartridge, see Edit a database to database cartridge.

1.3.5.9. Edit a schema-based in-place masking cartridge

To edit a schema-based In-Place masking cartridge:

1. Expand Cartridges in the Project Navigator window.
2. Select the schema-based In-Place masking cartridge that you want to edit. The In-Place masking cartridge dialog opens in the editor.
3. Enter or select values for the following as necessary:

Overview Properties

Field	Description
Version	Enter a version number for the cartridge or accept the default.

4. Click **Add** to display the Add Tables dialog box and select tables to be masked.

> **Note**
>
> You can extract data from different database schemas (using different access rights) by creating a multi-source data model–this enables you to extract the data from all the schemas at once. Access rights only have an impact if you build the model in Designer. If you build the model in Designer, then you would need one account with read access to all the schemas you wish to include.

> **Tip**
>
> If the schemas are located in different databases, then you can use either views or database links to access the data (although this can impact performance), or, you can create multiple models or cartridges and combine them into a single business flow and run them sequentially.

> **Tip**
>
> If you build the model in Designer, then you would need one account with read access to all the schemas you wish to include.

5. Click **OK**.
6. If necessary, select tables for removal and click **Remove**.
7. Select the **Operations** tab. The Operations page opens with an **Copy** option.

> **Note**
>
> For IPMC feature, by default the copy option is selected and cannot be changed.

8. Select the **Data** tab.
9. Choose one of the radio buttons:
   - **All Data** indicates that all the rows in the table will be masked.
   - **No Data** indicates no rows in the table will be will be masked.
   - **Selected Data** indicates only the selected rows in the table will be masked and a WHERE clause is used to define the range.

     If you choose this radio button, type or select values for the following as necessary:

WHERE Clause for Choosing Data

Field	Description
Database	Choose the database to which the WHERE clause applies. You can have a different WHERE clause for as many of the vendors and versions as necessary. For example, you could associate a different WHERE clause with DB2, SQL Server, Oracle 8i, and Oracle 9i. At runtime, Structured Data Manager can determine the database vendor and version, and apply the clause that you assigned to that vendor and version. Choose Any for the default WHERE clause. The Any WHERE clause is used for any vendor and version which does not have an assigned WHERE clause. For example, if you do not assign a WHERE clause to Oracle or any of its versions, the Any WHERE clause will be used when deploying against Oracle. If the clause has vendor or version specific calls or syntax in it, expand the Any node until you see the necessary option for the clause and select it.
WHERE	Type or edit the WHERE clause.
Parameters and Columns	Double-click the column name to insert the name into the WHERE clause.
Insert	Alternatively, highlight a column and use the **<<Insert** button to move it to the WHERE clause.

10. Select the **Data Movement Key** tab.
11. Choose an option from the Key list of values:
    - Unique keys in the table
    - **[Default]** means Structured Data Manager will choose a key for you at deployment.
    - **[Virtual Key]** means you want to use a virtual key and will specify the columns to use.
    - **[Use Row ID]** (Oracle only)

    Note

    As of now, IPM cartridge supports only primary keys as Data Movement Key. If primary key exists then 'Default' option can be selected as Data Movement Key. Use of 'Row Id' as Data Movement Key is not supported.

12. Select the **Data Masking** tab. The Edit Data Masking dialog opens and you can apply data masks to particular columns as necessary to protect privacy or hide confidential information, such as Social Security Numbers and credit card numbers. See Applying Data Masks.

    In case of IPMC as its name suggests, the data is updated to the source database. This is the difference between database to file cartridge and IPMC.

13. Click the **Column Inclusion** tab. The **Edit Column Inclusion** page opens.

Use this tab to specify how table columns are included or excluded by your cartridge.

> **Note**
>
> It is recommended to always select **Include Columns** option to achieve better performance in data masking. With this, all the masking columns are included by default, and hence the user needs to select only the additional columns required for context masking.
>
> Do not select **Include All** option unless all the columns are required for masking.

14. Select the **Custom Properties** tab. The Custom Properties dialog opens and behaves similarly to the Custom Properties dialog for a database to database cartridge, see Edit a database to database cartridge.

> **Note**
>
> While running the In-Place Masking business flow, user must ensure to disable the AQS cache from the databse to file system parameters. Also, the Destination Location for that business flow must be a **LOCAL** filesystem location.

1.3.5.10. Edit a data access cartridge

To edit a data access cartridge

1. Expand Cartridges in the Project Navigator window.
2. Select the data access cartridge that you want to edit. The Data Access cartridge overview opens in the editor.
3. Type or select values for the following as necessary.

Overview Properties

Field	Description
Query	A list of data access rules that you can turn on or off for the cartridge.
Allow Data Access	Indicates that you want the data access cartridge to appear under Data Access in the Web Console when you deploy it. Users can run the cartridge from there interactively and see the query results. In cases where you do not want users to run the cartridge directly in this fashion, you would uncheck this option to prevent it.
Allow Free Text Search	Is also checked by default. This option indicates that you want to be able to use the data access cartridge to render search results.
Title	The name that you want to appear to users of the cartridge.
Description	The descriptive text that appears to users where they select the data access cartridge in the Web Console.
Folder(s)	This option allows you to organize your archive into named folders, separated by semicolons.
Folder(s)	The names of folders where you want to place the cartridge upon deployment. This field is useful when you have many cartridges and want to organize them. Separate multiple folder names with a semicolon.

4. Select the **Master Template** tab. The master template determines what will be displayed to the user for the list of results.
5. In the upper left, select the check boxes next to the columns to add them. Once you check a column it will appear in the preview area. You can change the order of the columns by selecting them and clicking **Up** or **Down**. You can also change the heading of the column by selecting it and typing in the **Header** field.

   To add a pseudo column, which obtains its values from Groovy code, click **Add**.

   If you use HTML tags within your Groovy code to include remote objects using the <IMG> tag, those objects must be directly accessible through `HTTP. Proxies and file://` are not supported in Groovy fields.

6. Click the **Detail Template** tab. The detail template determines what will be displayed to the users when they drill down on the master.
7. In the upper left, select the table from which you want to add columns. The available columns below change to match the selected table.
8. Select the check boxes next to the columns you want to add to the detail. As with the Master template, you can change the order using **Up** and **Down**, and change the column heading using **Header**.

   As with the master template, you can click **Add** to insert pseudo columns, which obtain their values from Groovy code.
9. Select the Data Source tab. The Edit Data Source for 3-Tier Archive dialog opens.

   In a 3-tier topology, lookup tables can reside in the source database, the history database, or partially in source and partially in history. Hence, if you have a lookup table in your model, then you must indicate to Structured Data Manager the location from which it should obtain the lookup table data (source, target or both).

   In cases where a table is referenced as both a lookup and transactional table in the same model, the table will always reside in the target database and you do not need the Data Source tab.

   Choose the source you wish to use:

   - **Source** indicates that the data is in the active (source) tables.
   - **History** indicates that the data is in the archive (target) tables.
   - **Union** indicates that the data may reside in either the active or archive tables and Structured Data Manager should determine which is the appropriate source to use.
10. Select the **Custom Properties** tab. The Custom Properties dialog opens and behaves similarly to the Custom Properties dialog for a database to database cartridge, Editing a Database to Database Cartridge.

1.3.5.11. Apply data masks

By applying a data mask to a column, you can obfuscate its values in the archived data, thus protecting private, confidential information, such as credit card or social security numbers.

Tip

Values are not masked for Designer preview. They are only masked when running the business flow outside of Designer from Web Console or a command line interface.

Structured Data Manager provides pre-built masks for the data which most commonly requires protection, such as, credit card numbers and Social Security numbers. In addition, you can create your own, custom masks for data that you wish to mask.

Before applying a data mask, consider all of the following:

- The type of data masking you choose for archiving has implications for reloading data. If you choose a type of masking that cannot be reversed, then that data cannot be reloaded. You need to consider whether you expect to reverse masking upon reload when you choose the masking type. See Data Masking and Reload/Undo.
- If you choose to mask a primary key, you should always choose a reversible mask. Otherwise, when you reload the data into a database, your primary key values are lost and your table may no longer behave as expected.
- You must ensure that the data mask you apply corresponds to the data type of the column. For example, if the column is a numeric value, you must apply a numeric mask to it or use a function to convert the data to a data type that is compatible with the mask.
- If you apply a mask on a primary or foreign key, then database integrity is maintained because masks applied to primary keys are automatically applied to foreign keys and vice versa. This holds good in case of database to file but it will be a problem in case of IPM.

See also, Advanced Data Masking for the advanced data masking tutorial.

To apply a data mask

1. In the cartridge editor, select the **Data Masking** tab. The Edit Data Masking dialog opens.
2. If you need to apply a mask to a column, expand the table node that contains the column, for example, CUSTOMER.
3. Select the table column, for example, CREDITCARD.
4. Choose a value from **Mask Type** or, if you are using a mask you defined yourself, click **Custom Masks**. See Pre-Built Masks for more information.

Pre-built masks

The simplest way to mask data is to apply a pre-built mask. Of course, not all situations can be accommodated by pre-built masks, and in such cases you can create a custom data mask. See Create custom data masks in standard environmentsfor the tutorial on creating custom masks.

There are scalar function equivalents that are available when using Interactive SQL or the interactive JDBC client. See the *OpenText™ Structured Data Manager Runtime Guide* for more information about these scalar function equivalents.

The following table lists the pre-built string masks provided by Structured Data Manager.

Caution

If your column contains any invalid data, such as invalid characters, the Social Security, credit card, and ABA/Routing string masks will not mask the value at all. For example, if you have a Social Security Number value that contains an invalid special character like #, none of that value will be masked in the archive. Hence, if invalid data is an issue in your columns, you may wish to create a custom mask that includes special logic for handling invalid data as you desire.

Tip

The only reversible string mask is String map (DB2 and Oracle only). If reversing the string mask upon reload is a requirement, then you must select the String map (DB2 and Oracle only) mask or create a custom mask that you can reverse yourself.

Note

The pre-built data masking functions support generic data types, such as VARCHAR, CHAR, NUMBER, FLOAT, and so on. The pre-built masks will not work on columns that are database-specific, such as unique identifier in SQL Server.

Pre-Built Data Masks—Strings

Mask	Description
ABA/Routing number: random	Validates that the given ABA number adheres to a supported format and, if it does, returns a new valid random ABA number. Otherwise, it returns the same input number. Because returning the same number could be a security issue, you should confirm that the numbers are in valid format prior to running with this mask. This mask supports a nine digit ABA number of the format XXXXXXXXX, where X is any digit [0-9].
Credit card number: random	Validates that the given credit card number adheres to a supported format and, if it does, returns a new, random credit card number of the same type. Otherwise, it returns the same input number. [a] This mask supports Visa, Master Card, American Express, and Discover card numbers. See Credit card masks for more information.
Credit card number: XXX mask	Validates that the given credit card number adheres to a supported format and, if it does, returns the original number with all digits masked by X, except for the last four digits. Otherwise, the mask returns the same input number.a This mask supports Visa, Master Card, American Express, and Discover card numbers. See Credit card masks for more information.
Social Security Number: random	Validates the given US Social Security number adheres to a supported format and, if it does, returns a new, random Social Security Number starting with 897, which is not assigned to any individual and can be safely used. Otherwise, the mask returns the same input number.a This mask supports both formats, xxx-xx-xxxx and xxxxxxxxx. If the input contains dashes, this function returns a number with dashes inserted in the 4th and 7th place.
Social Security Number: XXX mask	Validates that the given US Social Security number adheres to a supported format and, if it does, replaces all digits from the original number with letter X, except the last four digits. Otherwise, the mask returns the same input number.a This mask supports both formats, xxx-xx-xxxx and xxxxxxxxx. If the input contains dashes, this function returns a number with dashes inserted in the 4th and 7th place.

Mask	Description
String mask: random length	Returns a string of random length, bounded by a user-specified maximum, using a user-specified character. • **Character** (Data type Character) is the character to use in the string. You can specify any printable character. • **Length** (Data type Integer) is the maximum allowable length of the string. The generated string will have a random length between 1 and the value of Length.
String mask: same length	Returns a string of the same length using a user-specified character. • **Character** (Data type Character) is the character to use in the string. You can specify any printable character.
Random string	Returns a string of random length, bounded by a user-specified maximum, using random, printable characters. **Length** (Data type Integer) is the maximum allowable length of the string. The generated string will have a random length between 1 and the value of Length.
Random string: [a-z]	Returns a string of random length, bounded by a user-specified maximum, using only random lowercase, English alphabetic characters. **Length** (Data type Integer) is the maximum allowable length of the string. The generated string will have a random length between 1 and the value of Length.
Random string: [A-Z]	Returns a string of random length, bounded by a user-specified maximum, using only random uppercase, English alphabetic characters. **Length** (Data type Integer) is the maximum allowable length of the string. The generated string will have a random length between 1 and the value of Length.
Random string: [a-zA-Z]	Returns a string of random length, bounded by a user-specified maximum, using only random English, alphabetic characters. **Length** (Data type Integer) is the maximum allowable length of the string. The generated string will have a random length between 1 and the value of Length.

Mask	Description
Random string: [a-zA-Z0-9]	Returns a string of random length, bounded by a user-specified maximum, using only random English, alphanumeric characters. **Length** (Data type Integer) is the maximum allowable length of the string. The generated string will have a random length between 1 and the value of Length.
String map (DB2 and Oracle only, standard environment)	Looks up a given string in a map and returns the value to which it is mapped. If a string is not mapped to any value, then a null string is returned. This mask is reversible upon reload and thus the mapping must be complete (encompassing all values). If the mapping is not complete, an exception is thrown during archiving. The mapping is stored in a lookup table, which you must create and populate prior to using this function. This mask supports VARCHAR columns up to 256 characters in length. It is best employed on short strings, such as zip code, driver's license number, and telephone number. The data type and lengths of the columns in the lookup table must match those of the columns in the managed tables. For example, if the column in the source table is VARCHAR(100), then the corresponding column in the lookup table must be VARCHAR(100) as well. **Map table name** (Data type VARCHAR) is the name of a lookup table. A lookup table has only two columns, both of type VARCHAR with a maximum size of 256 characters: • The column named `orig` holds the key. • The column named `masked` holds the value. The lookup table must be created and populated before using this mask. In single instance, database to file archiving, the lookup table must be created in the source database. In the case of distributed, database to database archiving, the lookup table must be created in both the source and target databases with identically qualified names, or in the OBT interface schema on the source database.

Mask	Description

String map	Returns a string corresponding to input value from a predefined map provided in file. You must enter the String map information in a file where each line follows the format `key=value`, and key is mapped to the value by the mask. You must also specify the absolute path to this mapping file in the mapping file parameter.
	
	Alternatively, you can place it in the default location (`obt\extensions\runtime\masking`) and the value of the mapping file parameter is then just the filename of the mapping file. The format of the mapping file is:
	
	`key1=val1`
	`key2=val2`
	
	Enter only one key/value pair per line.

Fixed string	Returns a constant, user-specified string. **Value** (Data type String) is the string with which to replace all values. This replacement string should be compatible with the column's data type, for example, it should match the maximum length of the column.

The following table lists the numeric masks provided by Structured Data Manager.

Tip

The only reversible numeric masks are skew number: add and number map. If reversing the number mask upon reload is a requirement, then you must select skew number, or create a reversible custom mask yourself.

Pre-Built Data Masks—Numeric

Mask	Description
Fixed number	Returns a user-specified constant, numeric value. **Value** (Data type Numeric) is the constant with which to replace the column's values. This number must be compatible with the column data type, for example, the precision and scale must match.
Random number	Returns a random number within a user-specified range of values, both inclusive. **Minimum** (Data type Integer) is the lower bound of the range. **Maximum** (Data type Integer) is the upper bound of the range.
Skew number: add	Returns a skewed value, the original value plus a user-specified number. **Amount** (Data type Integer) is the number by which to increment the original value. It must be a positive or negative whole number. This function is reversible and supports whole numbers on all database platforms. If you attempt to apply this mask to other column types (for example, decimals or strings), then you will encounter an error when deploying the cartridge. For Oracle only, all types of numeric columns–including decimals–are supported.
Skew number: multiply	Returns a skewed value, the original number multiplied by a user-specified number. **Amount** (Data type Numeric) is the number by which to multiply the original value. For all databases (except Oracle), the result is rounded off to the nearest whole number. For example, if you input 10.1, then a skew amount of 20.9 will return a result of 200.
Skew number: percent	Returns a skewed value, the original number increased by a user-specified percentage. For example, to increment a number by 10%, enter 10 for the Percent. **Percent** (Data type Numeric) is the percentage by which the original value will be increased. For all databases (except Oracle), the result is rounded off to the nearest whole number.

Mask	Description
Number map (DB2 and Oracle standard environments only)	Looks up a given number in a map and returns the value to which it is mapped. This mask is reversible upon reload and thus the mapping must be complete (encompassing all values). If the mapping is not complete, an exception is thrown during archiving. The mapping is stored in the lookup table, which you must populate prior to using this function. **Map table name** (Data type VARCHAR) is the name of a lookup table. A lookup table has only two columns, both of type VARCHAR: The column named orig holds the key. The column named masked holds the value. The lookup table must be created and populated before using this mask. In single instance, database to database and database to file archiving, the lookup table must be created in the source database. In the case of distributed, database to database archiving, the lookup table must be created in both the source and target databases with identically qualified names, or in the OBT interface schema on the source database.

Number map	Returns a number corresponding to input value from a predefined map provided in file. You must enter the Number map information in a file where each line follows the format `key=value`, where key is mapped to the value by the mask. You must also specify the absolute path to this mapping file in the mapping file parameter. Alternatively, you can place it in the default location (`obt\extensions\runtime\masking`) and the value of the mapping file parameter is then just the filename of the mapping file. The format of the mapping file is:
	
	`key1=val1`
	`key2=val2`
	
	Enter only one key/value pair per line.

The following table lists the date masks provided by Structured Data Manager.

Pre-Built Data Masks—Date

Mask	Description
Skew date	Returns a skewed value, the original date plus a user-specified number of days. **Days** (Data type Integer) is the number of days by which to increment the original date. This function is reversible.

Credit Card Masks

The following table lists the credit card types and formats for which Structured Data Manager supports masking.

Supported Credit Card Types and Formats

Card type	Prefix	Length	Format
American Express	34,36	15	123412345612345
		17	1234-123456-12345
		17	1234 123456 12345
Discover	6011	16	1234123412341234
		19	1234-1234-1234-1234
		19	1234 1234 1234 1234
Mastercard	51-55	16	1234123412341234
		19	1234-1234-1234-1234
		19	1234 1234 1234 1234
Visa	4	16	1234123412341234
		19	1234-1234-1234-1234
		19	1234 1234 1234 1234
		13	1234123412345
Diners Card International	36	14	12341234561234
		16	1234-123456-1234
		16	1234 123456 1234

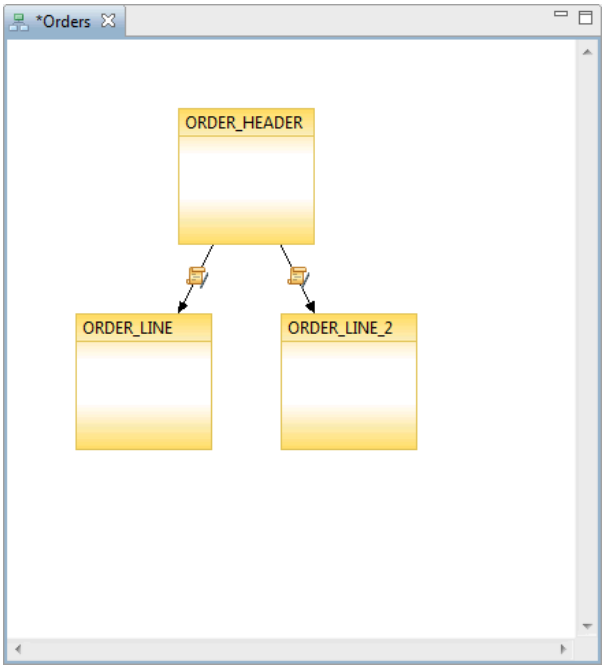
Card type	Prefix	Length	Format
Diners Card Carte Blanche	300-305	14	123412345631234
		16	1234-123456-1234
		16	1234 123456 1234

Data masking and referential integrity

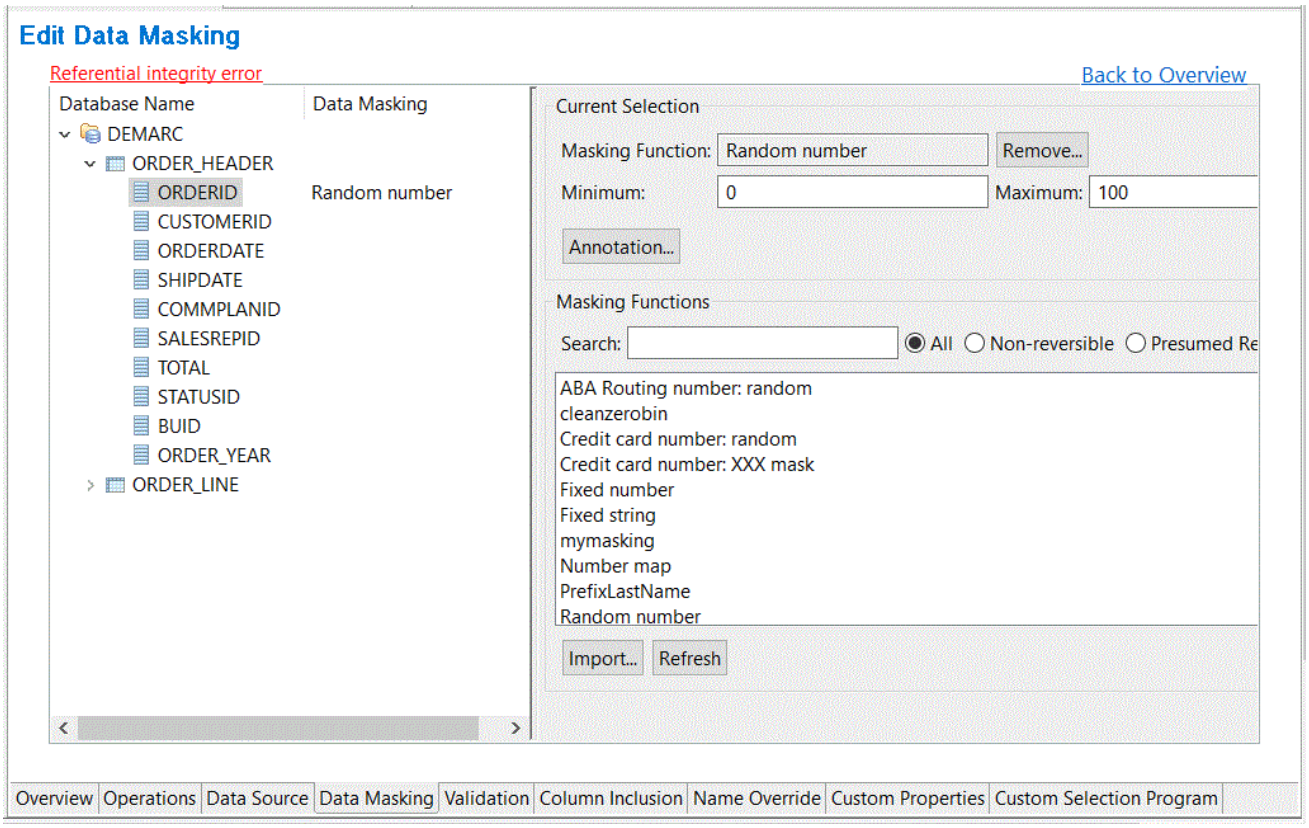
To maintain referential integrity in your archive database, you must take special care when applying data masks to primary or foreign key columns. Because referential integrity relies on primary and foreign keys, you must ensure that you mask them consistently. Otherwise, your archive will not work as expected when you copy or archive the data into another database. Also note that masks on primary keys must be one-to-one in order to preserve the uniqueness of primary keys.

Example

Suppose that you have a simple data model such as the following one. Notice that ORDER_LINE is used twice as a child transactional table in this model.



In a cartridge based on this model, you only see two tables in the Data Masking tab. Data masking is applied on a per table basis, not per table use. Hence, in the Data Masking tab, ORDER_LINE appears only once and any masks you set for it are applied to all of its table uses in the cartridge.



If you choose to mask the column containing the primary key (ORDERID) in ORDER_HEADER, you must also mask the column containing the corresponding foreign key in ORDER_LINE with a reversible mask to preserve the primary key constraints (uniqueness). If you try to apply a non-reversible mask or do not mask the column containing the foreign key as well, Designer issues a warning.

Notice the Referential integrity error link that appears in the upper left corner of the page. Click this link to view a more detailed error message.

Tip

If you choose a custom mask on a column containing the primary key, Designer cannot analyze your masking function's logic to determine whether or not it is one-to-one and thus will not issue a warning on a column's primary key masks. It is up to you to verify that the mask's logic is one-to-one. However, a referential integrity warning will be raised when custom masks on columns containing primary and foreign keys are different.

In this case, the message explains that you have two issues. First, you have masked the column with the primary key without similarly masking the column containing the foreign key. Consequently, the primary and foreign key values are no longer aligned with each other and could therefore no longer be used to find related rows. Second, you have masked the column containing the primary key (ORDERID) in ORDER_HEADER with a Random number mask, which is not a reversible mask.

If you change the mask of ORDERID in ORDER_HEADER to Skew number: add, which is reversible, you eliminate the latter error but not the former.

Note

Notice that the dialog still provides an informational message about the mask on the column containing the primary key (indicating that the mask is reversible). No further action on this item is required on your part.

The first message in the dialog is a warning that indicates that the column containing the foreign key is not masked, but the corresponding column containing the primary key is masked. To maintain referential integrity, columns containing the primary and foreign keys must be masked in a consistent manner. In this case, that would mean applying a Skew number: add mask with the same skew value to the foreign key column.

At the bottom of the dialog is a radio button that, when selected, will automatically apply the same reversible mask to the column containing the foreign key that has already been applied to the column containing the primary key. If you select this

option and click **OK**, the referential integrity warning is removed because both columns (with primary and foreign keys) are now masked by the same reversible mask.

Data masking and reload/undo

Masked data is reverted during undo/reload only if the corresponding masking function is reversible. In the case of reload cartridges, you must specify exactly the same masks on the same columns as you did in the archive cartridge. Data masked with irreversible masking functions cannot be recovered. If you perform an undo or reload on masked data that cannot be unmasked, you run the risk of corrupting the data in the source database.

For example, suppose you masked the Social Security number column when you archived with Social Security Number: XXX mask. In that case, a number like 299010191 is stored as XXXXX0191 in the archive data store. If you try to reload from the archive data store to the source database using undo or reload, the person with the Social Security Number 299010191 will end up getting XXXXX0191 assigned to them because the mask was not reversible.

To avoid this problem, Structured Data Manager includes a business flow configuration parameter named **Unmask data on undo and reload**. The default value is true, which causes the reload or undo operation to fail if the business flow used any irreversible masking functions. You must explicitly change the value of this parameter to false if you really want to reload the masked values into your source database. You should only set this parameter to false if you have some other way to restore the values of masked columns in the source database.

Similarly, for database to file uploads you can use the configuration parameter Unmask data on upload to specify whether or not data will be unmasked during an upload job. The default value is false, in which case data will not be unmasked during upload even if the mask is reversible. If you set the parameter to true, then data will be unmasked during upload, provided the mask being used during archive is reversible. If the mask used during archive is not reversible, then no attempt is made to revert the values and masked values will be uploaded as is from the file archive.

Reversible masks

The following masks are reversible:

- String map, if the map is one-to-one and encompasses all values
- Skew number: add
- Number map, if the map is one-to-one and encompasses all values
- Skew date
- Custom, if you include your own logic to reverse it

See also *OpenText™ Structured Data Manager Runtime Guide*

1.3.5.12. Create custom selection programs

Custom selection programs should only be used when a rule cannot be expressed as a WHERE clause on a table.

Prerequisites

- In non-intrusive environments where there is no custom selection table, the custom selection program must include a single SELECT statement against the driving table and the statement must select all columns. In standard environments, the custom selection is performed via an INSERT statement in a custom selection table.
- Due to the limitations of SQL Server, you cannot create a custom selection program for a managed table that contains IMG, NTEXT, TEXT, or XML columns.
- When specifying a custom selection table for DB2, use only alphanumeric characters. If you use a special character (such as a double quotation mark (“), single quotation mark (‘), etc.), then you may encounter an error when archiving.
- Custom selection program does not apply to in-place masking cartridges.

To create a custom selection program

1. In the cartridge editor for a model-based cartridge, select the **Custom Selection Program** tab. The Edit Custom Selection Program page opens.
2. Create the SQL to define your custom selection logic.

SQL Selection Properties

Element	Description
Table	Optionally, the name of the table to which the custom selection should apply. The default name is the cartridge name with _CS appended to it.
Database	Choose the database to which the code applies. You can have different code for as many of the vendors and versions as necessary. For example, you could associate different code with DB2, SQL Server, Oracle 8i, and Oracle 9i. At runtime, Structured Data Manager can determine the database vendor and version, and apply the code that you assigned to that vendor and version. Choose **Any** for the default code. The Any code is used for any vendor and version which does not have assigned code. For example, if you do not assign code to Oracle or any of its versions, the Any code will be used when deploying against Oracle. If the code has vendor or version specific calls or syntax in it, expand the Any node until you see the necessary option for the clause and select it.
Code	Use to type in the code you want to use for the custom selection program. The code can be an INSERT statement or procedural code. Use PL/SQL for Oracle or Transactional SQL for SQL Server. For Oracle and SQL Server non-intrusive environments only, you must use a single SELECT statement instead of PL/SQL or Transactional SQL.
Insert button	Use to insert parameter names into the custom selection code.
Associated Tables	See Managing associated tables.
Create Parameters	Parses the code and adds new default parameters to the project if it finds proper parameter references that have no existing, corresponding parameter.

Note

For Oracle, the custom selection table includes a column, OBT_SAVED_ROWID, which must be populated with the driving table's ROWID.

1.3.6. Working with business flows

A business flow is a set of activities including a cartridge and any other activities necessary to accomplish an archive solution.

This section includes:

1.3.6.1. About business flows

Business flows consist of various activities that you want to perform in sequence. Some activities, such as archive activities, can be split into steps and you can perform other activities in between those steps. For example, you might run a script that calculates the number of rows to be archived in between the selection and copy steps of an archive activity.

A business flow can include any of the following activities:

- Archive activities that invoke an archive cartridge
- Reload activities that invoke a reload cartridge
- Conditions that enable you to branch your business flow
- Interrupts that stop your business flow
- Groovy scripts that perform some additional processing, including but not limited to any of the following:
    - Executing operating system commands, such as copying a file, deleting a file, compressing a file, or transferring a file (ftp)
    - Sending an email or other notification
    - Database operations, such as dropping indexes, making a tablespace read-only, or analyzing statistics for a table

You edit business flows in a graphical environment where you can easily see the relationships among activities.

Note

If you edit a cartridge after deploying a business flow, then you must redeploy the business flow to see the latest changes that you made in the cartridge.

1.3.6.2. Add an archive database to database cartridge

To add an archive database to database cartridge

1. Expand the Business Flows node in the Project Navigator and double-click a business flow.
2. Click the **Archive** tool.
3. Click the location in the business flow where you want to add the cartridge. The Archive dialog box displays.
4. Select an archive database to database cartridge from the list.

   > **Note**
   >
   > You must have already created the cartridge in your project for it to appear in this list. See Create a new cartridge for more information about creating cartridges.

5. Choose one of the following selection methods for Selection:

   - **Standard** is a method of data selection that restricts itself to the rows identified by the model. It does not attempt to traverse related rows across multiple tables.
   - **Advanced** is a method of data selection that discovers all of the interrelated rows from multiple tables and conceptually places them in the same application partition for archiving.

   See also *OpenText™ Structured Data Manager Concepts Guide* for more information about data selection methods.

6. Click **OK**. The activity is added to the business flow diagram.

   > **Tip**
   >
   > If you add a database to database archive cartridge and a database to file cartridge to the same business flow, the latter is assumed to be a 2-tier configuration. That is, the data will be moved/copied from the active database, not the archive database. If you want a 3-tier configuration (data moving from the archive database to file), you need to put your database to file cartridge in a separate business flow.

1.3.6.3. Creating a business flow

To create a business flow

1. Select the Business Flows node in the Project Navigator.
2. Right-click and select **New Business Flow**. The New Business Flow dialog opens.
3. Type a name for the business flow and click **OK**. The Business Flow Editor opens.
4. Click **Properties** to display and optionally change the business flow's properties.
5. On the **General tab**, you can see the version of the business flow and order the parameters used by the business flow.
6. Select the **Installation Script** tab. In the Code area, you can enter a Groovy script to be executed at deployment time. **Templates** provide Groovy script templates to help you get started.
7. Select the **Custom Properties** tab. The Custom Properties dialog opens and behaves similarly to the Custom Properties dialog for a database to database cartridge, see Editing a Database to Database Cartridge.
8. Click **OK**.

   You can now add activities to your business flow. See the following sections for more information:

   Note

   If you have multiple activities in your business flow and one of them fails at runtime, subsequent activities will not run. Hence, you should consider only including activities that are related in the same business flow. Unrelated activities that can run independently of one another need not be in the same business flow.

   - Adding an Archive Database to Database Cartridge
   - Adding a Reload Database to Database Cartridge
   - Adding an Archive Database to File Cartridge
   - Add an in-place masking cartridge
   - Splitting an Activity
   - Adding or Editing a Script
   - Adding or Editing a Condition
   - Adding or Editing an Interrupt

   See also Generating and Deploying for more information on generating and deploying your business flow.

1.3.6.4. Add a reload database to database dartridge

To add a reload database to database cartridge:

1. Expand the Business Flows node in the Project Navigator and double-click a business flow.
2. Click the **Reload** tool.
3. Click the location in the business flow where you want to add the cartridge. The Reload dialog box displays.
4. Select a reload database to database cartridge from the list.

   > **Note**
   >
   > You must have already created the cartridge in your project for it to appear in this list. See Creating a New Cartridgefor more information about creating cartridges.

5. For Selection, choose one of the following selection methods:

   - **Standard Selection** is a method of data selection that restricts itself to the rows identified by the model. It does not attempt to traverse related rows across multiple tables.
   - **Advanced Selection** is a method of data selection that discovers all of the interrelated rows from multiple tables and conceptually places them in the same application partition for archiving.

   > **Note**
   >
   > You cannot choose Advanced Selection if working in a non-intrusive Oracle environment. For more details about non-intrusive environments, see the *OpenText™ Structured Data Manager* Concepts Guide or *OpenText™ Structured Data Manager Runtime Guide*.

   See also *OpenText™ Structured Data Manager Concepts Guide* for more information about data selection methods.

6. Click **OK**. The activity is added to the business flow diagram.

1.3.6.5. Add an archive database to file cartridge

To add an archive database to file cartridge to a business flow

1. Expand the Business Flows node in the Project Navigator and double-click a business flow.
2. Click the **Archive** tool.
3. Click the location in the business flow where you want to add the cartridge. The Archive dialog box displays.
4. Select a database to file cartridge from the list.

   > **Note**
   >
   > You must have already created the cartridge in your project for it to appear in this list. See Creating a New Cartridge for more information about creating cartridges.

   > **Note**
   >
   > **Standard Selection** is the only method of data selection for database to file cartridges. It restricts itself to the rows identified by the model. It does not attempt to traverse related rows across multiple tables.

5. For Data Movement, choose a data movement method. The possible choices are:

   Copy. Copies data from the active database to a file.

   ITIP f you choose to copy rather than archive, you can still remove the data from the database at a later time (deferred delete) using the Web Console. See the *OpenText™ Structured Data Manager Runtime Guide* for more information about relocating data.

   Archive. Copies data from the active database to a file and removes the rows from the active database.

   See also

   OpenText™ Structured Data Manager Runtime Guide for more information about configuring connections to backend devices.

   OpenText™ Structured Data Manager Concepts Guide for more information about data movement methods.

6. Click **OK**. The activity is added to the business flow diagram.

   > **Tip**
   >
   > If you add a database to database archive cartridge and a database to file cartridge to the same business flow, the latter is assumed to be a 2-tier configuration. That is, the data will be moved or copied from the active database, not the archive database. If you want a 3-tier configuration (data moving from the archive database to file), you need to place your database to file cartridge in a separate business flow.

> **Note**
>
> In non-intrusive environments only, 2-tier and 3-tier configurations are not supported.

1.3.6.6. Add an in-place masking cartridge

To add a in-place masking cartridge

1. Expand the Business Flows node in the Project Navigator and double-click a business flow.
2. Click the **Masking** tool to create a masking activity.

   Select the In-Place Masking cartridge from **Insert a Cartridge** dialog box.

   > Tip
   >
   > You must have already created the cartridge in your project for it to appear in this list. See Create a new cartridge for more information about creating cartridges.

3. Click **OK**.

   The activity is added to the business flow diagram.

1.3.6.7. Add a schema-based archive cartridge

Adding a schema-based cartridge works the same way as Adding an Archive Database to Database Cartridge and Adding an Archive Database to File Cartridge except that you can only perform standard selection.

1.3.6.8. Split an activity

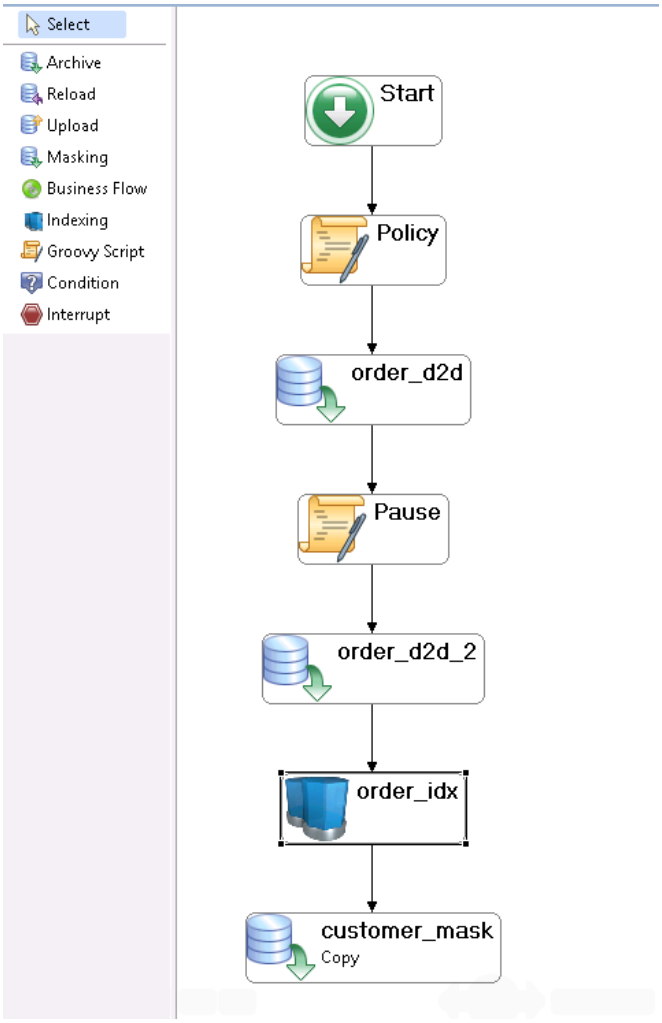
> **Note**
>
> In non-intrusive environments only, selection and copy occur at the same time. If you split a cartridge in a non-intrusive environment, selection will result in an empty operation, whereas copy will incorporate both selection and copy. Clean-up also results in an empty operation.

To split an activity

1. Expand the Business Flows node in the Project Navigator and double-click a business flow.
2. Right-click on an archive or a reload activity in the editor and select **Split**. The Split Activity dialog opens.

   > **Note**
   >
   > You cannot split an activity based on a schema-based cartridge.

3. Use the shuttle buttons to indicate where the job split should be located.
4. Click **OK**. The split activity is displayed.



   The bullets represent the steps of the job. Green bullets indicate that the activity contains the corresponding step. Empty bullets indicate that the activity does not contain the step.

   The step name is displayed when you move the mouse cursor over a bullet.

5. Repeat step 2 through step 4 until you have completed all the splits that you want in your business flow.
6. Delete an activity to reverse a split. The steps are merged into the remaining activity.

7. Save your work.

See also Generating and Deploying for more information on generating and deploying your business flow.

1.3.6.9. Upload to a database

You can use Upload to add an upload activity immediately after the archive cartridge activity in the business flow. Uploading to a database in this manner saves you from having to upload to the database separately from the Web Console.

To add an upload activity to a business flow

1. Expand the Business Flows node in the Project Navigator and double-click a business flow.
2. Click Upload in the tool bar and drop it in the editor below an archive cartridge. The Upload Activity dialog opens.
3. Select an Archive activity that precedes the insertion point and click OK.

   > **Note**
   >
   > If there is no Archive activity before this point, then there is nothing to select. Ensure that the insertion point follows an Archive activity.

   The Upload activity is now displayed in the business flow.
4. Save your work.

When you run the business flow from the Web Console, you will be prompted to specify the database location to which to upload the archive data. The upload occurs immediately after the archive successfully completes.

1.3.6.10. Add or edit a script

Groovy scripts used in a business flow must be written in the Groovy language version 1.6.8. You should test your Groovy scripts in a standalone Groovy environment before including them in your business flow.

> **Note**
>
> Structured Data Manager does not validate your code. You must validate and ensure that your code performs as expected at runtime yourself.

To add or edit a script

1. Expand the Business Flows node in the Project Navigator and double-click a business flow.
2. Click **Groovy Script** in the tool bar and drag it to the editor, or double-click an existing script activity. The Groovy Script dialog box opens.
3. Type or edit the name for the script.
4. Type the Groovy code in the Code field.
5. Optionally, click **Parameters** to add or edit a parameter. See Creating or Editing Parameters Using the Parameters Button.
6. Click **Template** to choose a code template for your Groovy script.
    1. Select a template in one of the following categories:
        - SQL
        - Operating System
        - Utilities
        - Utilities - Install BF
        - SRMS
    2. Click **OK**.
7. Edit the code as necessary.

   The code you type is analyzed. If there are any errors, the error message is displayed at the bottom of the window.

   Click the error message to move the caret to the error position.
8. Click **OK** and save your work.

See also Generating and Deploying for information on generating and deploying your Business Flow.

Add custom Groovy events to a business flow

In addition to including Groovy scripts within business flows, you can insert Groovy events to be executed before and after certain Structured Data Manager events. Groovy events can be executed at the following points:

- Before deployment. This Groovy code fires just before Structured Data Manager deploys your business flow to the environment. This event is useful for confirming that anything required for the business flow to deploy successfully is in place prior to deployment. For example, you might use it to check that certain system parameters are properly seeded.
- After deployment. This Groovy code fires just after Structured Data Manager completes deployment of your business flow to the environment. This event is useful for any sort of cleanup that you might want to perform after deployment.
- Before launch. This Groovy code fires just before Structured Data Manager runs your business flow in the environment. This event is useful for confirming that anything required for the business flow to run successfully is in place prior to execution. For example, you might use it to check that certain locations are accessible and available before trying to execute the business flow.
- After launch. This Groovy code fires just after Structured Data Manager completes execution of your business flow in the environment. This event is useful for any sort of cleanup or reporting that you might want to perform after execution. For example, you might use it to generate a report showing you certain results from the business flow's execution.

Your Structured Data Manager installation includes sample Groovy scripts for all of these supported events. You can simply modify these samples for your own purposes. For example, if you need to collect auditing information upon deployment or execution of a business flow, you can customize the sample Groovy files for the after deployment or launch events to include the necessary logic. Or, suppose that you must provide an external locking mechanism for the deployment of a business flow, a custom script can be of help.

The following sample scripts are located in the home directory under `extensions\runtime\events\<event_name>`:

- `beforeDeployBf.sample`
- `afterDeployBf.sample`
- `beforeLaunchBf.sample`
- `afterLaunchBf.sample`

To use the scripts, simply rename the files by replacing the .sample file extension with .groovy. You can view all output in the home directory under log. For more information about the home directory, see Managing the home directory.

If multiple events exist, then the order in which the scripts are run is determined by basic alphanumeric sorting rules. The following list shows the order in which a sample group of multiple Groovy events would run:

```
20runme.groovy
S10runme.groovy
S20runme.groovy
Zrunme.groovy
aRunMe.groovy
s10runme.groovy
s11runme.groovy
s1runme.groovy
s20runme.groovy
s2runme.groovy
s3runme.groovy
```

See Adding or Editing a Script

1.3.6.11. Add or edit a condition

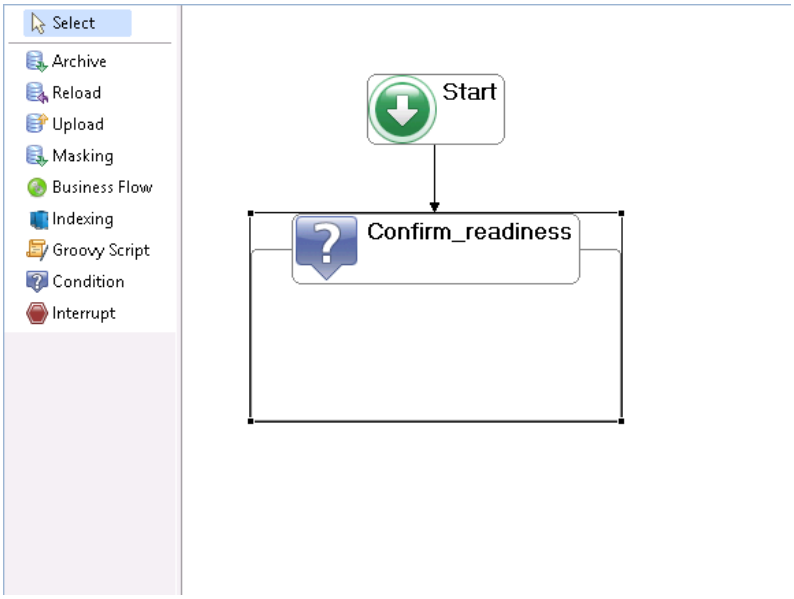
In some cases, you may want to check some condition before proceeding with an activity in your business flow. If the condition is met, then you will proceed with the activity. If not, you skip the activity and go to the next activity, if there is one, or end the business flow.

To add or edit a condition

1. Expand the Business Flows node in the Project Navigator and double-click a business flow.
2. Click **Condition** in the tool bar and drag it to the editor, or double-click an existing condition. The dialog box opens.
3. Type or edit the name for the script.
4. Type or edit the Groovy code in the Code field for your condition. The code must return a Boolean value to indicate whether the condition was met. For example, you might enter something like:

```
import java.io.*
 new File("c:/temp/x.txt").exists()
```

5. Click **OK**. Your condition is created or modified. Notice the bounding box around the condition.



6. To place an activity within the condition, click a tool in the tool bar (Archive, Reload, Groovy Script, Condition, Interrupt).
7. Click inside the bounding box of the condition and complete the resulting dialog as necessary. The activity will be governed by the condition. If the condition returns true, the activities inside of its bounding box are executed. If the condition returns false, the activities inside of the bounding box are skipped.

> **Note**
>
> You cannot split an archive or reload activity and place some of its pre- or post-activities inside of the condition and some outside the condition. All of the pre- and post-activities for the archive or reload must be contained either inside or outside of the condition. To stop execution between pre- and post-activities of an archive or reload activity, you can use an interrupt. See Adding or editing an interrupt.

8. Repeat step 6 and step 7 for any additional activities you want to place under the control of the condition.

1.3.6.12. Add or edit an interrupt

In some cases, you may wish to stop or pause your business flow, typically after checking some condition. To stop or pause a business flow at any point, you can introduce an interrupt into the business flow.

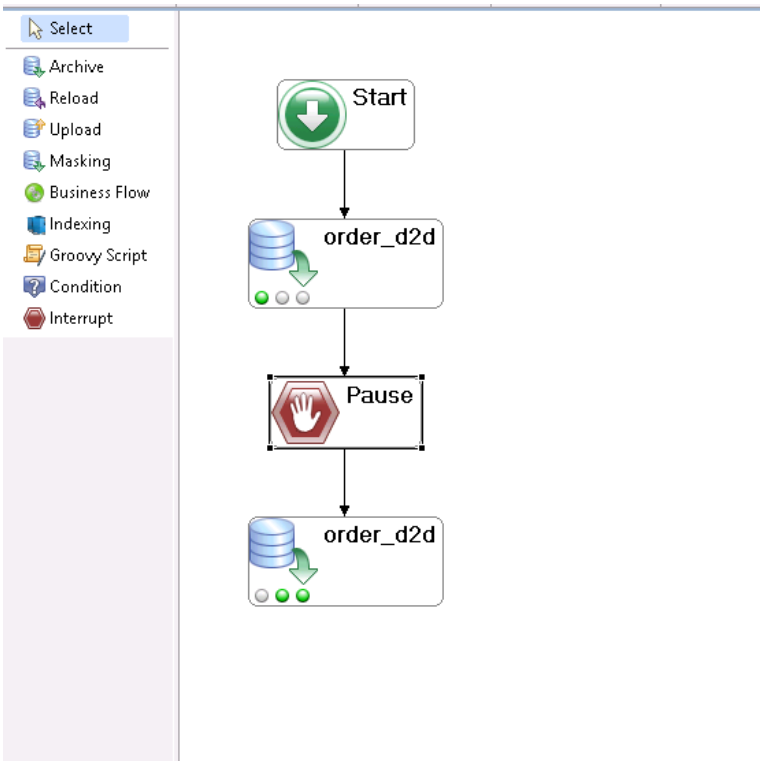
To add or edit an interrupt:

1. Expand the Business Flows node in the Project Navigator and double-click a business flow.
2. Click **Interrupt** in the tool bar and drag it to the editor, or double-click an existing interrupt activity. The Interrupt Flow Activity dialog box opens.
3. Choose the Type of interrupt you want to create:
   - **Error** indicates that the business flow stops with an error condition, meaning that it cannot continue.
   - **Pause** indicates that the business flow execution is temporarily suspended and can be continued later.
   - **Exit Successfully** indicates that the business flow completed its activities successfully.
   - **Exit with Warning** indicates that the business flow completed, but some activities caused warnings to be issued.
4. Type or edit the Message to display when the interrupt is executed.
5. In Condition, type the Groovy code for your condition. The code must return a Boolean value to indicate whether the condition was met. For example, you might enter something like:

```
import java.io.*
 new File("c:/temp/x.txt").exists()
```

> **Tip**
>
> A condition is required if your interrupt type is **Error**. For the other types of interrupt, a condition is optional.

6. Click **OK**. Your interrupt is created.



1.3.6.13. Test business flows

With any business flow, you should carefully test it before running it in your production environment. As best practice, you should:

- Thoroughly test and debug your Groovy scripts in a Groovy IDE.
- For interrupts and conditions, ensure that you have test cases that meet and do not meet the condition. Otherwise, you cannot exercise all branches.
- Preview any models referenced by cartridges in your business flow. Ensure that you vary parameter values within reasonable ranges.
- Preview any archive cartridges used in your business flow.
- Deploy and run the business flow in a development environment. Carefully check all logs and confirm all results before trying the business flow in production.

See also *OpenText™ Structured Data Manager Troubleshooting Guide*

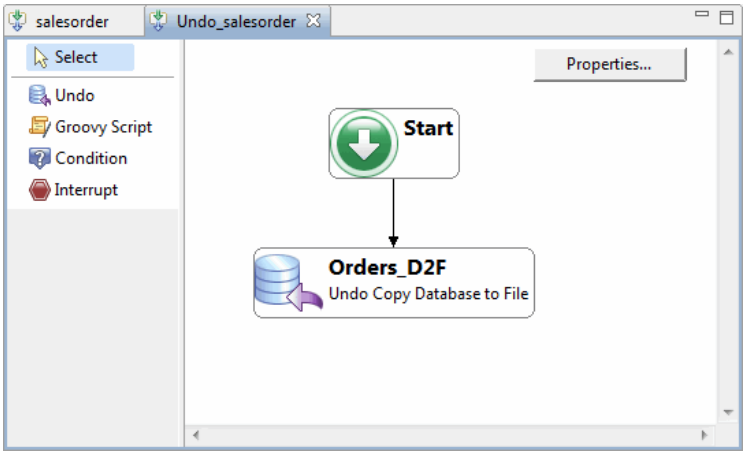
1.3.6.14. Create an undo business flow

Given the critical nature of application data, it is generally a good practice to prepare undo versions of your business flows. Such business flows provide a useful safeguard in case a problem arises immediately after you execute a business flow. The undo business flow effectively lets you reload everything that was archived by the original business flow. If you should discover a flaw in your business flow logic or any other issue after running a business flow, you can run the undo business flow to effectively return your environment to its original state, prior to running the business flow.

Designer can automatically generate undo business flows that will undo cartridge activities. The generated undo business flow is a good starting point for you, but you need to test and confirm that the undo business flow logic is correct. For example, suppose your business flow contains additional business logic in Groovy scripts. Designer cannot automatically determine how to undo that logic.

To create an undo business flow:

1. Right click your business flow in the Project Navigator and choose **Create Undo Business Flow**.
2. Enter the name you want to give the undo business flow (for example, `Undo_salesorder`) and click **OK**.
3. Click **OK**. Designer generates an undo business flow for you.



Tip

As with any business flow, you should fully test your undo business flow in a development environment before deploying and running it in production. SeeTesting Business Flows.

1.3.7. Generating and deploying

After your cartridge or business flow is complete and tested, you are ready to run it on the actual active database, outside of Designer. To run against your active database, you must first deploy the business flow or cartridge to your deployment environment. The deployment environment is where Structured Data Manager actually performs operations on your specified source (active) database. Until your cartridge or business flow is deployed to the environment, you cannot execute it.

This section includes:

- Deploy cartridges and business flows
- Generate deployment files

1.3.7.1. Deploy cartridges and business flows

Deployment Assistant displays different pages depending on the type of cartridge you are deploying.

Note

When you deploy a cartridge by itself, Structured Data Manager wraps it in a business flow for deployment purposes.

Tip

The sections that follow describe how to deploy using your Web Console URL and credentials. Alternatively, you can choose the Deploy Locally option if you are deploying to a Web Console instance on the same machine where you are running Designer. When you choose to deploy locally, the Deployment Assistant prompts you for encryption key.

This section includes:

- Deploying a Database to Database Cartridge
- Deploying a Reload Cartridge
- Deploying a Database to File Cartridge
- Deploy an in-place masking cartridge
- Deploying a Data Access Cartridge
- Deploying a Business Flow
- Deploy a Business Flow with SAML

Deploy a database to database cartridge

To deploy a database to database cartridge

1. In Designer, expand the Cartridges node in the Project Navigator.
2. Right click the cartridge you want to deploy and select **Deploy**. The Deployment Assistant opens.
3. Select the **Deploy Remotely** radio button and enter the URL and credentials for the Web Console instance to which you want to deploy the cartridge.
4. Optionally, check **Include Documentation**.

   See also Annotating Projects.
5. Click **Next**.
6. Choose one of the following selection methods for Selection:

   Standardis a method of data selection that restricts itself to the rows identified by the model. It does not attempt to traverse related rows across multiple tables.

   Advancedis a method of data selection that discovers all of the interrelated rows from multiple tables and conceptually places them in the same application partition for archiving.

   See also *OpenText™ Structured Data Manager Concepts Guide* for more information about data selection methods.
7. Click **Next** and type the encryption key value, which was defined during the repository installation. For example, EKEY. The encryption key is case sensitive. The encryption key is remembered for the rest of the session.
8. Click **Next**. If you have previously deployed the cartridge, the Job History Retention page displays. If it does not display, you can skip directly to step 12.
9. If the Job History Retention dialog displays, use the check boxes to specify what, if any, history you want to retain from previous deployments:

   Selecting **Drop job history of existing Business Flows** removes the job history. The job history enables you to see previous runs of the business flow to review information such as parameter settings and job results. Typically, you want

to retain this history for reference purposes, but you can drop it if you no longer plan to see it.

For database to database business flows, selecting **Drop Database to Database History tables of existing Business Flows** removes the history tables and the archived data. This option is typically used only in a development environment on test data. You would rarely, if ever, select this option in a production environment because it deletes the archived data.

> **Caution**
>
> Exercise extreme caution when choosing this option. It should rarely, if ever, be used in a production environment on real data because it can result in permanent data loss.

10. Click **Next** and type your active database administrator user name and password.
11. Click **Next**. If you have a distributed archive configuration, the Deployment Assistant prompts you for your archive database credentials. Type your archive database administrator user name and password. If not prompted, you can skip to step 12.
12. Click **Next** and enter your target and archive access passwords. These are the passwords you use to access the target and archive access schemas.
13. Click **Next**.

    For Oracle, you choose the tablespaces for your cartridge. Highlight one or more rows in the table. Use the controls at the bottom of the page to choose the desired tablespaces for the selected cartridges.

    For SQL Server, you can change History Data Size, History Log Size, AA Data Size, and AA Log Size.
14. Click **Next**.
15. On the Summary page, click **Finish**. Your cartridge may take a few moments to deploy.

Deploy a reload cartridge

To deploy a reload cartridge

1. In Designer, expand Cartridges in the Project Navigator.
2. Right click on the reload cartridge you want to deploy and select **Deploy**. The Deployment Assistant opens.
3. Select the **Deploy Remotely** radio button and enter the URL and credentials for the Web Console instance to which you want to deploy the cartridge.
4. Optionally, check **Include Documentation**.

   See also Annotating projects.
5. Click **Next.**
6. Choose one of the following selection methods for Selection:
   - **Standard** is a method of data selection that restricts itself to the rows identified by the model. It does not attempt to traverse related rows across multiple tables.
   - **Advanced** is a method of data selection that discovers all of the interrelated rows from multiple tables and conceptually places them in the same application partition for reloading.

   See also *OpenText™ Structured Data Manager Concepts Guide* for more information about data selection methods.
7. Click **Next** and choose the environment to which you wish to deploy, for example, Oracle_OLTP.

   See also *OpenText™ Structured Data Manager Concepts Guide* for more information about deployment environments.
8. Click **Next** and type your active database administrator user name and password.
9. Click **Next**. If you have a distributed archive configuration, the Deployment Assistant prompts you for your archive database credentials. Type your archive database administrator user name and password. If not prompted, you can skip to step 10.
10. Click **Next** and enter your target and archive access passwords. These are the passwords you use to access the target and archive access schemas.

11. Click **Next**.

    For Oracle, you choose the tablespaces for your cartridge. Highlight one or more rows in the table. Use the controls at the bottom of the page to choose the desired tablespaces for the selected cartridges.

    For SQL Server, you can change History Data Size, History Log Size, AA Data Size, and AA Log Size.

12. Click **Next**.
13. On the Summary page, click **Finish**. Your cartridge may take a few moments to deploy.

Deploy a database to file cartridge

To deploy a database to file cartridge

1. In Designer, expand Cartridges in the Project Navigator.
2. Right click the cartridge you want to deploy and select **Deploy**. The Deployment Assistant opens.
3. Select the **Deploy Remotely** radio button and enter the URL and credentials for the Web Console instance to which you want to deploy the cartridge.
4. Optionally, check **Include Documentation**.

   See Annotating Projects.

5. Click **Next**.

   Note

   Standard is the only method of data selection for database to file cartridges. It restricts itself to the rows identified by the model. It does not attempt to traverse related rows across multiple tables.

6. Choose one of the following methods for Data Movement:
   - **Copy**. Copies data from the active database to a file.
   - **Archive**. Copies data from the active database to a file and removes the rows from the active database.

   See also

   OpenText™ Structured Data Manager Concepts Guide for more information about data movement methods.

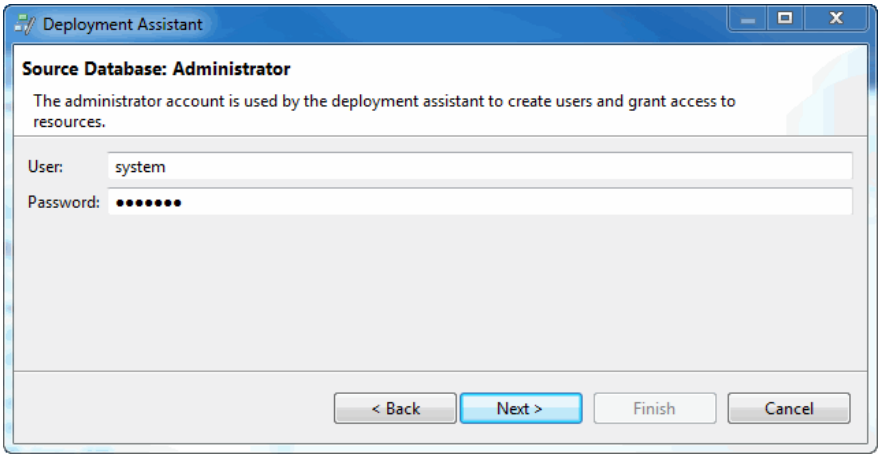
   OpenText™ Structured Data Manager Runtime Guide for more information about configuring connections to backend devices and data movement methods.

7. Click **Next** and choose an environment.
8. Click **Next**. If you the selected environment has both database to database and database to file installed, the Deployment Assistant prompts you for your topology. If not prompted for topology, you can skip directly to step 10. Otherwise, choose the topology for your archive. The possible choices are:
   - **2 Tier Configuration**. In a 2-tier configuration, database to XML archives directly from the active database to an XML store (for example, a file on the file system). If you choose this option, the Deployment Assistant next prompts you for your active database administrator credentials.
   - **3 Tier Configuration**. In a 3-tier configuration, database to XML archives from the archive database, which is populated through database to database archiving, to an XML store. If you choose this option for a distributed instance, the Deployment Assistant next prompts you for your archive database administrator credentials. If you choose this option for a single instance, the Deployment Assistant next prompts you for your active (source) database administrator credentials.

     See also *OpenText™ Structured Data Manager Concepts Guide*, for more information about 2 and 3 tier configurations.

9. Click **Next**. Depending upon which topology you chose and whether you have a single or distributed instance, you will be prompted for either your active database administrator credentials (2-tier) or your archive database administrator credentials (3-tier).
10. Enter the login information for your active or archive database, whichever one the Deployment Assistant prompts you for.

> **Note**
>
> In non-intrusive environments only, skip this step (as there is no login prompt).



11. Click **Next**.
12. On the Summary page, click **Finish**. Your cartridge may take a few moments to deploy.

Deploy an in-place masking cartridge

To deploy an in-place masking cartridge

1. Expand Cartridges in the Project Navigator.
2. Right-click the in-place masking cartridge you want to deploy and select **Deploy**. The Deployment Assistant opens.
3. Select **Deploy Remotely**and enter the URL and credentials for the Web Console instance to which you want to deploy the cartridge.
4. Optionally, select **Include Documentation**.

   See Annotating Projects.
5. Click **Next**.
6. Choose an environment.
7. Click **Next**.
8. On the Summary page, click **Finish**.

   Your cartridge may take a few moments to deploy.

Deploy a data access cartridge

To deploy a data access cartridge

1. In Designer, expand Cartridges in the Project Navigator.
2. Right click the cartridge you want to deploy and select **Deploy**. The Deployment Assistant opens.
3. Select the **Deploy Remotely** radio button and enter the URL and credentials for the Web Console instance to which you want to deploy the cartridge.
4. Click **Next**.
5. Select the environment and the sources that you want to union when the data access cartridge runs.
6. Click **Finish**.

Deploy a business flow

Note

If using a new, generic JDBC driver in a database to file non-intrusive environment, checks the driver's Name Types and JDBC Type values upon your first connection. If data is missing, then you may encounter an error at runtime. If there is a data mismatch, then you can encounter data loss upon business flow deployment. Before deploying a business flow for a database to file, non-intrusive environment, examine your data closely to ensure that there are no "mismatches" between the name Type and JDBC Type.

Note

See the *OpenText™ Structured Data Manager Runtime Guide* and the *OpenText™ Structured Data ManagerTroubleshooting Guide* for details about data mapping errors and mismatches when using generic JDBC drivers in database to file non-intrusive environments.

To deploy a business flow

1. In Designer, expand **Business Flows** in the Project Navigator.
2. Right-click on the business flow you want to deploy and select **Deploy**. The Deployment Assistant opens.
3. Select the **Deploy Remotely** radio button and enter the URL and credentials for the Web Console instance to which you want to deploy the business flow.
4. Optionally, check **Include Documentation**.

   See also Annotating Projects
5. Click **Next** and type the encryption key value. For example, EKEY. The encryption key is case sensitive. The encryption key is remembered for the rest of the session.
6. Depending upon the type of archive or reload cartridge that your business flow calls, the Deployment Assistant will display the appropriate pages for that type of cartridge:
   - If your business flow calls a database to database cartridge, the Deployment Assistant behaves as described in step 10 through step 15.
   - If your business flow calls a reload database to database cartridge, the Deployment Assistant behaves as described in step 8 through step 10.
   - If your business flow calls a database to file cartridge, the Deployment Assistant behaves as described in step 8 through step 12.
   - If your business flow calls both a database to database cartridge (archive, schema-based, or reload) and a database to file cartridge (archive or schema-based), the Deployment Assistant behaves as described in step 10 through step 15. In this scenario, the Deployment Assistant defaults to a 2 tier configuration where the database to file cartridge archives from the active database. If you want the database to file cartridge to archive from the archive database (3 tier), then you need to place it in a separate business flow

Deploy a business flow with SAML

Prerequisite

Before configuring SAML support in the Designer,

- Enable SAML on the SDM server. See *SAMLv2 Integration* in *OpenText™ Structured Data Manager Runtime Guide*
- Make sure that the RelayState is enabled by default on your IdP server.

To enable SAML support in the Designer

1. Navigate to `<Designer_HOME>` location, open `config.properties` in a text editor.

   The default path is `C:\Users\<USERNAME>\SDM\Designer`
2. Update the property `saml.enabled.webconsole.url` with the SAML enabled WebConsole URL.

   For example, `saml.enabled.webconsole.url=` https://<hostname>:8080/WebConsole

> **Note**
>
> SAML support in Designer is incompatible with self-signed certificates and if the Web Console is configured to use HTTPS, it must be set up with a CA-signed certificate. See *Configure HTTPS*, in *OpenText™ Structured Data Manager Runtime Guide*.

To deploy a business flow with SAML

1. In Designer, expand **Business Flows** in the Project Navigator.
2. Right-click on the business flow you want to deploy and select **Deploy**. The Deployment Assistant opens.
3. Select **Deploy with SAML**.
4. Click **Next**. Your are redirected to Idp server login page.
5. Log in using the SAML credentials.

   The SAML Authentication Status window opens upon successful login.
6. Navigate to the Designer Window and select the environments to continue business flow deployment with SAML.

> **Note**
>
> To disable the SAML support in the Designer, remove the URL value added to `saml.enabled.webconsole.url` in the `config.properties` file.

1.3.7.2. Generate deployment files

In some cases, you may not have direct access to the production system where the business flow or cartridge is to be deployed. For example, you might be providing cartridges or business flows to another organization with systems separate from your own. In such situations, you can generate the deployment files for your cartridges and business flows, and then move those files to the other system for subsequent deployment.

To generate and deploy files

1. Generate a deployment file from the Deployment Assistant configured with your development instance (that is, the database for which Designer is installed and configured).

   Generating a cartridge or business flow

2. Move the deployment file to the remote system and invoke the Web Console associated with the remote system to deploy that file.

   Deploying generated files

Generate a cartridge or business flow

To generate a cartridge or business flow

1. In Designer, expand Cartridges or Business Flows in the Project Navigator.
2. Right click on the cartridge or business flow you want to deploy and select **Deploy**. The Deployment Assistant opens.
3. Select the **Generate** radio button.
4. Optionally, check **Include Documentation**.

   See also Annotating Projects

5. Click **Next**. If you are deploying a cartridge, you are prompted to choose the data selection and movement methods. If you are deploying a business flow, you will proceed directly to step 7.
6. Choose the selection and data movement methods from the lists of values and click **Next**.
7. Type or browse to the location where you want to store the deployment file. For the sake of convenience, you may wish to choose a path (a network or shared drive) that is accessible from the remote system on which you plan to deploy the cartridge or business flow. Otherwise, you will need to move the deployment files to such a location after you generate them.

   Note

   The `\OBTHOME\businessflow` directory typically has subdirectories for each deployment environment. In most cases, you will want to generate to the subdirectory for the environment where you plan to deploy.

8. Click **Finish**. The deployment file is generated in the path that you chose in the Deployment Assistant. In that path, you should find a .busflow file with the same name as your business flow or cartridge. If you chose to include documentation, you should also find a PDF file in the same path.
9. Perform the steps in Deploying Generated Files on the remote system where you want to deploy and run the business flow or cartridge.

Deploy generated files

Before performing the steps in this section, you must have performed the steps in Generating a Cartridge or Business Flow and have a login with deployment privileges to the Web Console for the system where you plan to deploy.

Note

The Web Console must already be started on the remote machine for this procedure.

To deploy generated files

1. In a browser, invoke the Web Console URL for the system where you want to deploy and run the business flow or cartridge (`http://<hostname>:8080/WebConsole`).
2. Login as a user with privileges to deploy business flows, for example, `admin` .

   > **Note**
   >
   > For the purposes of this procedure, we assume that you are the admin user or a user with comparable privileges in the Web Console. If not, the pages and links described in the steps that follow may not all be visible to you.

3. From the Web Console home page, click **Business Flow Management**. Ensure that the currently active environment is the one in which you want to deploy the business flow.
4. From the menu at the top of the page, choose **Deployment**. If you stored the generated files in the home directory under `businessflow` , then they will appear in the list of business flows available for deployment. For more information about the home directory, see Managing the Home Directory.

   If you do not see the business flows you want to choose in the list, do the following:

   1. Click **Import** in the left navigation pane.
   2. Import the business flows in one of two ways:

      - Choose the **Path to the Server directory** radio button, enter a valid path on the server where the deployment files (*.busflow) are stored, and click **Copy**.
      - Choose the **Business flow file to upload** radio button and click **Browse** to select the deployment files (*.busflow).

      The business flows are uploaded from the client to the server.

   3. From the menu at the top of the page, choose **Deployment**. The business flow(s) that you just imported should now appear in the list.
5. Check the business flow(s) in the list that you wish to deploy.
6. Click **Next**. Depending upon the type of cartridge(s) contained in the business flow(s), pages will appear where you can provide the necessary information for deployment. For more information on deployment through the Web Console, see *OpenText™ Structured Data Manager Runtime Guide*.
7. When you reach the summary page, click **Finish** to deploy your business flow.

1.3.8. Customizing archive projects

Using Designer, you can safely customize a project you receive from a third party. When the third party releases revisions to the project, you can merge your customizations into the revised project from the third party. Thus, you need not constantly repeat your changes each time you receive a revision from the vendor.

This section explains how to customize a project you receive and merge new versions with your customizations.

Note

If you are a third party vendor developing projects for consumption, you should contact OpenText about becoming a partner and learning more about developing customizable projects.

This section includes:

1.3.8.1. About customization

When you receive a project that was pre-built for a packaged application such as PeopleSoft or Oracle E-Business Suite, you typically need to modify the project for your environment. For example, you might need to add to the model some tables that are required by your environment but that are not part of a default implementation of the packaged application.

As a consumer of a customizable archive project, you must be aware of several considerations and tasks:

- When you receive a project, it should be locked. When a project is locked by a vendor, it means that the various files that comprise the project are all locked. For example, each model in a project has a file associated with it. When the vendor locks the project, the existing model files are all locked.

  Tip

  Note that if you take an action, such as adding a new model to the project, that results in the creation of a new file, then the new file would not be locked.

- Even though the project is locked, you can perform most modifications that you like on it. You should be aware, though, that some modifications may cause the modified model, cartridge, business flow, or parameter to be unsupported when upgraded to a newer revision of the project. To ensure that your modifications are completely supported, you should modify the project according to the guidelines in Guidelines for Customizations.
- Generate a list of unique identifiers (UIDs) to use in validating your customizations.
- When the vendor releases new versions of the project, merge your customizations into the new version.

1.3.8.2. Guidelines for customizations

As you customize a project, you must take the following into account:

- By default, locked projects open in what is known as Customization view. In this mode, Designer gives you various visual cues in the form of icons in the model editor, to call out customizations. You can toggle Customization view on and off from the toolbar using the Custom View tool.
- Ensure that the project files are locked before you begin making modifications. If the Custom View tool in the model editor is disabled, it means that the files are not locked. If the files are not locked, you should lock them before you begin by selecting **Customization > Lock Project**.
- You can make modifications to the model and cartridges in a locked project. As long as you adhere to the guidelines in this section, those customizations are supported, which means you can merge them into a new version of the project when you receive it.
- When you make certain changes to the model or cartridge in a locked project, you receive confirmation dialogs with warning messages to ensure that you really want to make the change.

  If you click OK, the operation is carried out and, in the model editor, the object is marked in some fashion to indicate the customization.
- You can add new models, cartridges, business flows, and parameters to the project, but note that they are not locked because they create new unlocked files within the project.
- In a locked project, if you delete an original table use or a mandatory rule, it is marked as deleted rather than actually deleted. If you unlock the project, objects marked as deleted are actually removed. Similarly, if you add new tables or rules to a locked project, these are not added to the base project. If you unlock the project, then the new objects are added to the base project.
- Because original tables are not actually removed from a locked project, you need to take special care when removing a table. If you later recreate that table in the model, you could end up with the same table in the model twice, which could cause problems for you at some point.
- Each rule in the project is categorized according to its Customization property as follows:

  -Optional rules can be changed or deleted without triggering warnings or changing the UID.

  -Recommended rules can also be changed or deleted but only with great caution. Changing a recommended rule will trigger a warning, but the UID is not changed.

  -Mandatory rules cannot be changed or deleted. Such changes trigger a warning and cause the UID to change. If you choose to make the change anyway, the original rule is marked as deleted and a new rule is created to replace it.
- Business flow changes are not merged. If you make changes to a business flow, you cannot merge those customizations into a new version of the original project.

1.3.8.3. Customize a project

When you receive a new project to customize, you first import it into Designer for customization. Once you have the project in Designer, you can modify it and generate its UIDs for future reference when you merge a new version from the vendor with your modifications.

To customize a project:

1. In Designer, choose **File > New Project**. The New Project dialog box displays.
2. Enter a name for the project and select **Customize an Existing Project**.
3. Click **OK**. The Import Project dialog, from which you can select a .hdp file, displays.
4. Browse to and select the .hdp that you received from your vendor.
5. Click **Open**. The selected project is imported into Designer under the specified project name.
6. When you receive a new project, ensure that the project is locked. Open a model in the project for editing and, if the Custom View tool in the toolbar is disabled, it means that the project is not locked. If the project is not locked, you should lock it before you begin by selecting **Customization > Lock Project**.
7. Modify the project as desired but within the guidelines described in Guidelines for Customizations.
8. Save the project.
9. Choose **Customization > Generate UIDs**.
10. Click **Copy to Clipboard**.
11. Paste the UIDs into a document and save it for future reference.
12. Click **OK** in the Unique IDs dialog box.

1.3.8.4. Merge customizations into a new project revision

Before merging a customization, you should back up your project files.

To merge customizations into a new project revision:

1. Open your customized version of the project.
2. Review the guidelines in Guidelines for Customizations to ensure that you understand which of your modifications will be merged and supported.
3. Choose **Customization > Merge Project**.
4. In the Merge Project dialog box, browse for the new project file from your vendor, which has the .hdp extension.

   .hdp is an exchange format for Structured Data Manager projects and contains all of the files associated with the project. You create .hdp files by exporting a project from Designer, **File > Export Project**.
5. Click **OK**
6. Optionally, if you find that some of your customizations were not merged, compare the UIDs from your customized project to those of the developer's original project. Wherever the UIDs do not match, it indicates that you have an unsupported change.

Tip

If you want to create a new project based on a project from a vendor, you can do so by choosing **File > New Project** and checking Customize an Existing Project. See Creating a New Project for more details.

1.4. Appendixes

- SQL masking API
- Java masking API
- Appendix A: Extensions
- Appendix B: Custom Data Mask

1.4.1. SQL masking API

A SQL API is available that you can use for creating custom masks. For Oracle, the APIs are contained in the obt_masking package. The API consists of the following functions:

- GENERATE_ABA_ROUTING_NUMBER
- GENERATE_CREDIT_CARD_NUMBER
- GENERATE_SSN
- GET_CREDIT_CARD_TYPE
- LOOKUP_NUMBER
- LOOKUP_STRING
- MASK_ABA_ROUTING_NUMBER
- MASK_CREDIT_CARD_NUMBER
- MASK_SSN
- MASK_STRING
- RANDOM_FLOAT
- RANDOM_INTEGER
- RANDOM_STRING
- REVERT_LOOKUP_NUMBER
- REVERT_LOOKUP_STRING
- REVERT_SKEW_DATE
- REVERT_SKEW_INTEGER
- SKEW_DATE
- SKEW_FLOAT
- SKEW_INTEGER
- SKEW_PERCENT
- VALID_ABA_ROUTING_NUMBER
- VALID_CREDIT_CARD_NUMBER

For additional background and details about custom masking, see Creating Custom Data Masks in Standard Environments and Creating Custom Data Masks in Non-Intrusive Environments.

1.4.1.1. GENERATE_ABA_ROUTING_NUMBER

Generates a random ABA routing number.

Syntax

```
GENERATE_ABA_ROUTING_NUMBER()
```

Parameters

None

Returns

Returns an ABA routing number as a VARCHAR(9).

1.4.1.2. GENERATE_CREDIT_CARD_NUMBER

Generates a random credit card number.

Syntax

```
GENERATE_CREDIT_CARD_NUMBER(type, length)
```

Parameters

Parameter	Data Type	Description
type	VARCHAR(30)	The type of credit card number to generate. The possible values are: AMEX DISC MCRD VISA DINT DCBL for American Express Discover Master card Visa Diners Club Diners Club Carte Blanche
length	NUMBER	The length of the credit card number to return. The following are the valid values: 13 for VISA 14 for DINT and DCBL 15 for AMEX 16 for DISC, MCRD and VISA

Returns

Returns an a credit card number as a VARCHAR(16).

1.4.1.3. GENERATE_SSN

Generates a new, random social security number starting with 897. Social security numbers starting with 897 are not assigned to any individual and can be safely used.

Syntax

```
GENERATE_SSN(include_dashes)
```

Parameters

None

Parameter	Data Type	Description
include_dashes	VARCHAR(1)	Specifies whether to include dashes in the SSN. Specify 'Y' to get a number that includes dashes. For example, 897-23-2920. Specify 'N' to get a number with out dashes. For example, 897239320.

Returns

Returns a random SSN, where the first three digits are 897. The value returned is of type VARCHAR(11).

1.4.1.4. GET_CREDIT_CARD_TYPE

Based on the number of digits and prefix of a credit card number, this function determines if a credit card is American Express, Visa, and so on.

Syntax

```
GET_CREDIT_CARD_NUMBER(number)
```

Parameters

Parameter	Data Type	Description
number	VARCHAR(16)	The credit card number to validate.

Returns

Returns AMEX, DISC, MCRD, VISA, DINT or DCBL, depending on the credit card number input to the function. If no match is found, UKNW is returned. The value returned is of type VARCHAR(4).

1.4.1.5. LOOKUP_NUMBER

Masks a number by mapping the number to another number. The function looks up the original number in a table and returns the mapped number. The mapping is based on the mapping that you specify in a mapping table. The table contain two columns 'orig' and 'masked'. Your mapping table must cover all values in a one-to-one fashion. For example, if you are mapping two numbers, the file maps two distinct numbers to two other distinct numbers.

Syntax

```
LOOKUP_NUMBER(originalNumber, mappingTable)
```

Parameters

Parameter	Data Type	Description
originalNumber	NUMBER	Number to mask, using the mapping file.
mappingTable	VARCHAR(256)	Name of the mapping table to use.

Returns

Returns a masked value from a mapping table ("masked" column) that corresponds to the input value specified by `originalNumber`. The value returned is of type NUMBER.

1.4.1.6. LOOKUP_STRING

Masks a string by mapping the string to another string. The method looks up the original string in a table and returns the mapped string. The mapping is based on the mapping that you specify in a mapping table. The table contains two columns 'orig' and 'masked'. Your mapping table must cover all values in a one-to-one fashion. For example, if you are mapping two strings, the file maps two distinct strings to two other distinct strings.

Syntax

```
LOOKUP_STRING(originalString, mappingTable)
```

Parameters

Parameter	Data Type	Description
originalString	VARCHAR(254)	String to mask, using the mapping table.
mappingTable	VARCHAR(256)	Name of the mapping table to use.

Returns

Returns a masked value from a mapping table ("masked" column) that corresponds to the input value specified by `originalNumber`. The value returned is of type VARCHAR.

1.4.1.7. MASK_ABA_ROUTING_NUMBER

Validates an ABA routing number. If the number is valid, returns a random routing number.

Syntax

```
MASK_ABA_ROUTING_NUMBER(abaNumber)
```

Parameters

Parameter	Data Type	Description
abaNumber	VARCHAR(9)	The ABA routing number to validate then mask.

Returns

If ABA-routing number is valid, `MASK_ABA_ROUTING_NUMBER` returns a randomly generated routing number as type VARCHAR(9). Otherwise, the number input is returned.

1.4.1.8. MASK_CREDIT_CARD_NUMBER

Validates a credit card number. If the number is valid, returns a randomly generated credit card number of the same type or masks the original number.

Syntax

```
MASK_CREDIT_CARD_NUMBER(creditCardNumber, maskType)
```

Parameters

Parameter	Data Type	Description
creditCardNumber	VARCHAR(30)	The credit card number to validate then mask. For the supported credit card types and formats, see the following table.
maskType	VARCHAR(3)	Specifies the type of mask to return. The `maskType` parameter can be one of the following: NEW—generate a new random credit card number. XXX—replace all digits with an X except for the last four digits of the credit card number.

The following table lists the supported credit cards and formats supported by MASK_CREDIT_CARD_NUMBER.

Card type	Prefix	Length	Format
American Express	34,36	15	123412345612345
		17	1234-123456-12345
		17	1234 123456 12345
Discover	6011	16	1234123412341234
		19	1234-1234-1234-1234
		19	1234 1234 1234 1234
Mastercard	51-55	16	1234123412341234
		19	1234-1234-1234-1234
		19	1234 1234 1234 1234
Visa	4	16	1234123412341234
		19	1234-1234-1234-1234
		19	1234 1234 1234 1234
		13	1234123412345
Diners Card International	36	14	12341234561234
		16	1234-123456-1234
		16	1234 123456 1234
Diners Card Carte Blanche	300-305	14	123412345631234
		16	1234-123456-1234
		16	1234 123456 1234

Returns

If the credit card number is valid, `MASK_CREDIT_CARD_NUMBER` returns one of the following, depending on the mask type specified:

A random credit card number of the same type.

The masked credit card number where all digits of the input number are replaced by the letter X, except for the last four numbers.

If the credit card number is invalid, `MASK_CREDIT_CARD_NUMBER` returns the number that was input.

The value returned is of type VARCHAR(16).

1.4.1.9. MASK_SSN

Validates a social security number. If the number is valid, returns a randomly generated social security number or masks the original number, depending of the type of masking specified.

Syntax

```
MASK_SSN(ssn, maskType)
```

Parameters

Parameter	Data Type	Description
ssn	VARCHAR(11)	The social security number to mask. Supports the formats XXXXXXXXX and XXX-XX-XXXX.
maskType	VARCHAR(3)	Specifies the type of mask to return. The `maskType` parameter can be one of the following: 'NEW'—generate a new random social security number. 'XXX'—replace all digits with an X except for the last four digits of the social security number.

Returns

`MASK_SSN` returns one of the following, depending on the mask type specified:

A random social security number of the same type.

The masked social security number where all digits of the input number are replaced by the letter X, except for the last four numbers.

If the social security number is invalid, `MASK_SSN` returns the number that was input.

The value returned is of type VARCHAR(11).

1.4.1.10. MASK_STRING

Returns a masked string based on a specified type and length. The string and string length returned are determined by the substitute character, whether the length of the string should be random or fixed, and the maximum length that should be returned.

Syntax

```
MaskString(orignalString, substituteCharacter, lengthType, maxLength)
```

Parameters

Parameter	Data Type	Description
originalString	VARCHAR(1024)	String to be masked.
substituteCharacter	CHAR	Character to use for masking the string.
lengthType	CHAR	Length of the string returned. The following are the possible values for `lengthType`: R specifies that the masked string returned has a random length no longer than `maxLength`. O specifies that the masked string is always the same length as the original string. For example, if the original string is 5 characters long, the masked string returned is 5 characters long. S specifies that only the substitute character is returned as the string mask, and the length of the string is 1.
maxLength	NUMBER	Maximum length of the masked string that can be returned when `lengthType` is 'R'.

Returns

Returns a string created by replacing all characters in `originalString` by the substitute character. Length of the returned string is controlled by `lengthType` and `maxLength`.

1.4.1.11. RANDOM_FLOAT

Generates a random float number with a user-specified range.

Syntax

```
Number RANDOM_FLOAT(Number lower, Number upper)
```

Parameters

Parameter	Data Type	Description
lower	Number	The lower bound of the user-specified range.
upper	Number	The upper bound of the user-specified range.

Returns

Returns a randomly-generated float number within the lower and upper number range, both of which are inclusive. The return value data type has a precision of 31 and a scale of 15.

1.4.1.12. RANDOM_INTEGER

Generates a random integer between a specified range, inclusive of the upper and lower values for the range.

Syntax

```
RANDOM_INTEGER(lower, upper)
```

Parameters

Parameter	Data Type	Description
lower	NUMBER	The minimum number that the method can return.
upper	NUMBER	The maximum number that the method can return.

Returns

Returns a randomly generated whole number in the range specified by the lower to upper values. The value returned is of type NUMBER.

1.4.1.13. RANDOM_STRING

Generates a random string of the specified length and type.

Syntax

```
RANDOM_STRING(characterType, length)
```

Parameters

Parameters	Data Type	Description
characterType	VARCHAR(20)	The type of string to generate. The types are: ANY_ALPHA_CHAR—The string returned contains letters. ANY_NUMERIC_CHAR—The string returned contains alphanumeric characters. LOWER_CASE_ALPHA—The string returned contains only lower-case letters. UPPER_CASE_ALPHA—The string returned contains only uppercase letters.
length	NUMBER	The length of string to generate.

Returns

Returns a randomly generated string, containing characters as specified by `characterType` with the length specified by `length`. The value returned is of type VARCHAR(1024).

1.4.1.14. REVERT_LOOKUP_NUMBER

This function is used to recover a value masked by LOOKUP_NUMBER. It unmasks the number by mapping the number to another number. The function looks up the original number in a table and returns the mapped number. The mapping is based on mapping that you specify in a mapping table. The table contains two columns 'orig' and 'masked'. Your mapping table must cover all values in a one-to-one fashion. For example, if you are mapping two numbers, the table maps two distinct numbers to two other distinct numbers.

Syntax

```
REVERT_LOOKUP_NUMBER(originalNumber, mappingFile)
```

Parameters

Parameter	Data Type	Description
originalNumber	NUMBER	Number to recover, using the mapping file.
mappingTable	VARCHAR(256)	Name of the mapping table to use.

Returns

Returns a masked value from a mapping table ('masked' column) that corresponds to the input value specified by `originalNumber`. The value returned is of type NUMBER.

1.4.1.15. REVERT_LOOKUP_STRING

This method is used to recover a value masked by LOOKUP_STRING. It unmasks the string by mapping the string to another string. The function looks up the original string in a table and returns the mapped string. The mapping is based on mapping that you specify in a mapping table. The table contains the mappings in two columns 'orig' and 'masked'. Your mapping table must cover all values in a one-to-one fashion. For example, if you are mapping two strings, the table maps two distinct strings to two other distinct strings.

Syntax

```
REVERT_LookupString(originalString, mappingTable)
```

Parameters

Parameter	Data Type	Description
originalString	VARCHAR(254)	String to recover, using the mapping file.
mappingTable	VARCHAR(256)	Name of mapping table to use.

Returns

Returns a masked value from the mapping table that corresponds to the input value specified by `originalString`. The returned value is of type VARCHAR.

1.4.1.16. REVERT_SKEW_DATE

Performs the opposite function of SKEW_DATE, allowing you to revert the result. REVERT_SKEW_DATE subtracts the specified number of days from the date.

Syntax

```
REVERT_SKEW_DATE(originalDate, skewAmount)
```

Parameters

Parameter	Data Type	Description
originalDate	DATA	Original date to unskew.
skewAmount	NUMBER	The number of days by which to decrement the original date.

Returns

Returns a date as a DATE type, where the date is skewed by subtracting `skewAmount` days from the original date.

1.4.1.17. REVERT_SKEW_INTEGER

Performs the opposite function of SKEW_INTEGER, which skews the original integer by adding, subtracting, multiplying, or dividing the number by a specified amount.

Syntax

```
REVERT_SKEW_INTEGER(originalNumber, action, skewAmount)
```

Parameters

Parameter	Data Type	Description
originalNumber	NUMBER	Base value to be skewed
action	VARCHAR(10)	A string that specifies the action to use to skew `originalNumber`. The action is one of the following: Add—adds `skewAmount` to `originalNumber`. Subtract—subtracts `skewAmount` from `originalNumber`. Multiply—multiplies `originalNumber` by `skewAmount`. Divide—divides `originalNumber` by `skewAmount`.
skewAmount	NUMBER	The amount by which to skew the original number value.

Returns

Returns a skewed amount, using the specified action. The value returned is of type NUMBER.

1.4.1.18. SKEW_DATE

Returns a skewed date based on an original date plus a specified number of days.

Syntax

```
SKEW_DATE(originalDate, skewAmount)
```

Parameters

Parameter	Data Type	Description
originalDate	Date	Original date to skew.
skewAmount	Number	The number of days by which to increment the original date.

Returns

Returns a date as a Date type, where the date is skewed by adding `skewAmount` days to the original date.

1.4.1.19. SKEW_FLOAT

Skews a float value. The skewed value is created from an original float value plus a user-specified number.

SKEW_FLOAT is not reversible (due to possible truncation of values).

Syntax

```
Number SKEW_FLOAT(Number originalNumber, String action, Number skewAmount)
```

Parameters

Parameter	Data Type	Description
originalNumber	Number	Value to be skewed
action	String	A string that specifies the method to use to skew `originalNumber`. The action is one of the following: • Subtract—subtracts `skew_amount` from `originalNumber` • Add—adds `skewAmount` to `originalNumber` • Mulitply—multiplies `originalNumber` by `skewAmount`. • Divide—divides `originalNumber` by `skewAmount`
skewAmount	Number	The amount by which to skew the original number.

Returns

Returns a numeric value that is skewed from the input number input to the skewed amount, using the specified action. This function can handle numbers with a precision of 31 and a scale of 15; the return value type also has a precision of 31 and scale of 15.

1.4.1.20. SKEW_INTEGER

Skews an original integer by adding, subtracting, multiplying, or dividing the number by a specified amount.

Syntax

```
SKEW_INTEGER(originalNumber, action, skewAmount)
```

Parameters

Parameter	Data Type	Description
originalNumber	NUMBER	Base value to be skewed
action	VARCHAR(10)	A string that specifies the action to use to skew `originalNumber`. The action is one of the following: Add—adds `skewAmount` to `originalNumber`. Subtract—subtracts `skewAmount` from `originalNumber`. Multiply—multiplies `originalNumber` by `skewAmount`. Divide—divides `originalNumber` by `skewAmount`.
skewAmount	NUMBER	The amount by which to skew the original number value.

Returns

Returns the number input to the function by the skew amount, using the specified action. The value returned is of type NUMBER.

1.4.1.21. SKEW_PERCENT

Returns a skewed value where the original number is increased by a specified percentage.

Syntax

```
SKEW_PERCENT(originalNumber, skewPercent)
```

Parameters

Parameter	Data Type	Description
originalNumber	NUMBER	The value to skew.
skewPercent	NUMBER	The percentage of the original number to be added to skew the number. Specify whole number for this parameter instead of decimals. For example, to skew the number by 10%, specify 10 for this parameter, not 0.10.

Returns

Returns originalNumber * (1 + skewAmount)/100 as a value of type NUMBER.

1.4.1.22. VALID_ABA_ROUTING_NUMBER

Validates an ABA routing number.

Syntax

```
VALID_ABA_ROUTING_NUMBER(abanumber)
```

Parameters

Parameter	Data Type	Description
abaNumber	VARCHAR(9)	The ABA routing number to validate.

Returns

Returns VARCHAR(10) value that is the string VALID if the ABAnumber is valid or INVALID if it is not valid. For DB2, return type is Boolean, returning true for a valid number and false for an invalid one.

1.4.1.23. VALID_CREDIT_CARD_NUMBER

Validates a credit card number by using the credit card checksum algorithm (Luhn).

Syntax

```
VALID_CREDIT_CARD_NUMBER(number, length)
```

Parameters

Parameter	Data Type	Description
number	VARCHAR(24)	The credit card number to validate.

Returns

Returns a VARCHAR(10) value that is the string VALID if the credit card number is valid or INVALID if it is not valid. For DB2, the return type is Boolean, returning true for a valid number and false for an invalid one.

1.4.2. Java masking API

A Java API is available that you can use for creating custom masks. The methods for this API are contained in the class com.outerbay.foundation.components.datamasking.java.DataMasking as static methods. This class contains the following methods:

- GENERATE_ABA_ROUTING_NUMBER
- GENERATE_CREDIT_CARD_NUMBER
- GENERATE_SSN
- GET_CREDIT_CARD_TYPE
- LOOKUP_NUMBER
- LOOKUP_STRING
- MASK_ABA_ROUTING_NUMBER
- MASK_CREDIT_CARD_NUMBER
- MASK_SSN
- MASK_STRING
- RANDOM_FLOAT
- RANDOM_INTEGER
- RANDOM_STRING
- REVERT_LOOKUP_NUMBER
- REVERT_LOOKUP_STRING
- REVERT_SKEW_DATE
- REVERT_SKEW_INTEGER
- SKEW_DATE
- SKEW_FLOAT
- SKEW_INTEGER
- SKEW_PERCENT
- VALID_ABA_ROUTING_NUMBER
- VALID_CREDIT_CARD_NUMBER

For additional background and details about custom masking, see See "Creating custom data masks in standard environments" and See "Creating custom data masks in non-intrusive environments".

1.4.2.1. GENERATE_ABA_ROUTING_NUMBER

Generates a valid, random ABA routing number.

Syntax

```
String GENERATE_ABA_ROUTING_NUMBER()
```

Parameters

This function has no parameters.

Returns

Returns a new, random ABA routing number that is nine digits in length.

1.4.2.2. GENERATE_CREDIT_CARD_NUMBER

Generates a random credit card number of a specific type and length.

Syntax

```
String GENERATE_CREDIT_CARD_NUMBER(String creditCardType, Integer NumDigits)
```

Parameters

Parameter	Data Type	Description
creditCardType	String	Type of credit card number to generate. Possible values include: • AMEX • DISC • MCRD • VISA • DINT • DCBL for American Express • Discover • Mastercard • Visa • Diner's Club • Diner's Club Carte Blanche
numDigits	Integer	The number of digits in the credit card. Possible values include: • 13 for Visa • 14 for DINT and DCBL • 16 for DISC, MCRD and VISA

Returns

A randomly-generated credit card number of a specific type containing a specified number of digits. If there is a mismatch, then an exception is returned.

1.4.2.3. GET_CREDIT_CARD_TYPE

Based upon the number of digits and the prefix of the credit card number, determines the type of credit card (AMEX, Visa, etc.).

Syntax

```
String GET_CREDIT_CARD_TYPE(String creditCardNumber)
```

Parameters

Parameter	Data Type	Description
creditCardNumber	String	The valid credit card number–no dashes, no spaces.

Returns

If the credit card number is not valid or incorrect, returns UNKW (unknown). Otherwise, returns one of the following:

- AMEX
- DISC
- MCRD
- VISA
- DINT
- DCBL

1.4.2.4. GENERATE_SSN

Generates a random U.S. social security number. The first three digits generated are always 897 because these numbers are not yet publicly circulated or in use.

Syntax

```
Boolean GENERATE_SSN(Boolean Include-Dashes)
```

Parameters

Parameter	Data Type	Description
Include-Dashes	Boolean	Generates a social security number that includes dashes. For example, 897-23-2920; otherwise, the return value is 897232920.

Returns

Returns a randomly generated U.S. social security number whose first three digits are 897.

1.4.2.5. LOOKUP_NUMBER

Masks a number by mapping the number to another number. The method looks up the original number in a file and returns the mapped number. The mapping is based on the mapping that you specify in a mapping file. The file must contain the mappings in the format original=masked. Your mapping file must cover all values in a one-to-one fashion. For example, if you are mapping two numbers, the file maps two distinct numbers to two other distinct numbers.

Syntax

```
BigDecimal LOOKUP_NUMBER(BigDecimal originalNumber, String mappingFile)
```

Parameters

Parameter	Data Type	Description
originalNumber	BigDecimal	Number to mask, using the mapping file.
mappingFile	String	Absolute path to the mapping file to be used to map the original number.

Returns

Returns a masked value from the mapping file that corresponds to the input value specified by `originalNumber`.

1.4.2.6. LOOKUP_STRING

Masks a string by mapping the string to another string. The method looks up the original string in a file and returns the mapped string. The mapping is based on the mapping that you specify in a mapping file. The file must contain the mappings in the format original=masked. Your mapping file must cover all values in a one-to-one fashion. For example, if you are mapping two strings, the file maps two distinct strings to two other distinct strings.

Syntax

```
String LOOKUP_NUMBER(String originalString, String mappingFile)
```

Parameters

Parameter	Data Type	Description
originalString	String	String to mask, using the mapping file.
mappingFile	String	Absolute path to the mapping file to be used to map the original string.

Returns

Returns a masked value from the mapping file that corresponds to the input value specified by `originalString`.

1.4.2.7. MASK_ABA_ROUTING_NUMBER

Validates an ABA routing number. If the number is valid, returns a random routing number.

Syntax

```
String MASK_ABA_ROUTING_NUMBER(String abaNumber)
```

Parameters

Parameter	Data Type	Description
abaNumber	String	The ABA routing number to validate and then mask.

Returns

If ABA-routing number is valid, `MASK_ABA_ROUTING_NUMBER` returns a randomly generated routing number. Otherwise, the same input number is returned.

1.4.2.8. MASK_CREDIT_CARD_NUMBER

Validates a credit card number. If the number is valid, returns a randomly generated credit card number of the same type or masks the original number.

Syntax

```
String MASK_CREDIT_CARD_NUMBER(String creditCardNumber, String maskType)
```

Parameters

Parameter	Data Type	Description
creditCardNumber	String	The credit card number to validate then mask. For the supported credit card types and formats, see the following table.
maskType	String	Specifies the type of mask to return. The `maskType` parameter can be one of the following: NEW—generate a new random credit card number. XXX—replace all digits with an X except for the last four digits of the credit card number.

The following table lists the supported credit cards and formats supported by the `MASK_CREDIT_CARD_NUMBER`.

Card type	Prefix	Length	Format
American Express	34,36	15	123412345612345
		17	1234-123456-12345
		17	1234 123456 12345
Discover	6011	16	1234123412341234
		19	1234-1234-1234-1234
		19	1234 1234 1234 1234
Mastercard	51-55	16	1234123412341234
		19	1234-1234-1234-1234
		19	1234 1234 1234 1234
Visa	4	16	1234123412341234
		19	1234-1234-1234-1234
		19	1234 1234 1234 1234
		13	1234123412345
Diners Card International	36	14	12341234561234
		16	1234-123456-1234
		16	1234 123456 1234
Diners Card Carte Blanche	300-305	14	123412345631234
		16	1234-123456-1234
		16	1234 123456 1234

Returns

If the credit card number is valid, MASK_CREDIT_CARD_NUMBER returns one of the following, depending on the mask type specified:

A random credit card number of the same type.

The masked credit card number where all digits of the input number are replaced by the letter X, except for the last four numbers.

If the credit card number is invalid, MASK_CREDIT_CARD_NUMBER returns the number that was input.

1.4.2.9. MASK_SSN

Validates a social security number. If the number is valid, returns a randomly generated social security number or masks the original number, depending of the type of masking specified.

Syntax

```
String MASK_SSN(String ssn, String maskType)
```

Parameters

Parameter	Data Type	Description
ssn	String	The social security number to mask. Supports the formats XXXXXXXXX and XXX-XX-XXXX.
maskType	String	Specifies the type of mask to return. The `maskType` parameter can be one of the following: NEW—generate a new random social security number. XXX—replace all digits with an X except for the last four digits of the social security number.

Returns

If the social security number is valid, `MASK_SSN` returns one of the following, depending on the mask type specified:

A random social security number of the same type.

The masked social security number where all digits of the input number are replaced by the letter X, except for the last four numbers.

If the social security number is invalid, MASK_SSN returns the number that was input.

1.4.2.10. MASK_STRING

Returns a masked string based on a specified type and length. The string and string length returned are determined by the substitute character, whether the length of the string should be random or fixed, and the maximum length that should be returned.

Syntax

```
String MASK_STRING(String originalString, Character substituteCharacter, Character lengthType,
Integer maxLength)
```

Parameters

Parameter	Data Type	Description
originalString	String	String to be masked.
substituteCharacter	Character	Character to use for masking the string.
lengthType	Character	Length of the string returned. The following are the possible values for `lengthType`: R specifies that the masked string returned has a random length no longer than `maxLength`. O specifies that the masked string is always the same length as the original string. For example, if the original string is 5 characters long, the masked string returned is 5 characters long. S specifies that only the substitute character is returned as the string mask, and the length of the string is 1.
maxLength	Integer	Maximum length of the masked string that can be returned when `lengthType` is 'R'.

Returns

Returns a String created by replacing all characters in `originalString` by the substitute character. Length of the returned String is controlled by `lengthType` and `maxLength`.

1.4.2.11. RANDOM_FLOAT

Generates a random float number with a user-specified range.

Syntax

```
Float RANDOM_FLOAT(Float lower, Float upper)
```

Parameters

Parameter	Data Type	Description
lower	Float	The lower bound of the user-specified range.
upper	Float	The upper bound of the user-specified range/

Returns

Returns a randomly-generated float number within the lower and upper number range, both of which are inclusive.

1.4.2.12. RANDOM_INTEGER

Generates a random integer between a specified range, inclusive of the upper and lower values for the range.

Syntax

```
Integer RANDOM_INTEGER(Integer lower, Integer upper)
```

Parameters

Parameter	Data Type	Description
lower	Integer	The minimum number that the method can return.
upper	Integer	The maximum number that the method can return.

Returns

Returns a randomly generated, whole number in the range specified by the lower to upper values.

1.4.2.13. RANDOM_STRING

Generates a random string of the specified length and type.

Syntax

```
String RANDOM_STRING(String characterType, Integer length)
```

Parameters

Parameters	Data Type	Description
characterType	String	The type of string to generate. The types are: • ANY_ALPHA_CHAR—The string returned contains letters. • ANY_NUMERIC_CHAR—The string returned contains alphanumeric characters. • LOWER_CASE_ALPHA—The string returned contains only lower-case letters. • UPPER_CASE_ALPHA—The string returned contains only uppercase letters.
length	Integer	The length of string to generate.

Returns

Returns a randomly generated String, containing characters as specified by `characterType` with the length specified by `length`.

1.4.2.14. REVERT_LOOKUP_NUMBER

This method is used to recover a value masked by LOOKUP_NUMBER. It unmasks the number by mapping the number to another number. The method looks up the original number in a file and returns the mapped number. The mapping is based on mapping that you specify in a mapping file. The file must contain the mappings in the format original=masked. Your mapping file must cover all values in a one-to-one fashion. For example, if you are mapping two numbers, the file maps two distinct numbers to two other distinct numbers.

Syntax

```
BigDecimal LOOKUP_NUMBER(BigDecimal originalNumber, String mappingFile)
```

Parameters

Parameter	Data Type	Description
originalNumber	BigDecimal	Number to recover, using the mapping file.
mappingFile	String	Absolute path to the mapping file to be used to recover the original number.

Returns

Returns a value from the mapping file that corresponds to the input value specified by `originalNumber`.

1.4.2.15. REVERT_LOOKUP_STRING

This method is used to recover a value masked by See "LOOKUP_STRING". It unmasks the string by mapping the string to another string. The method looks up the original string in a file and returns the mapped string. The mapping is based on mapping that you specify in a mapping file. The file must contain the mappings in the format original=masked. Your mapping file must cover all values in a one-to-one fashion. For example, if you are mapping two strings, the file maps two distinct strings to two other distinct strings.

Syntax

```
String REVERT_LOOKUP_STRING(String originalString, String mappingFile)
```

Parameters

Parameter	Data Type	Description
originalString	String	String to recover, using the mapping file.
mappingFile	String	Absolute path to the mapping file to be used to recover the original number.

Returns

Returns a value from the mapping file that corresponds to the input value specified by `originalString`.

1.4.2.16. REVERT_SKEW_DATE

Performs the opposite function of See "SKEW_DATE", allowing you to revert the result. `REVERT_SkewDate` subtracts the specified number of days from the date.

Syntax

```
Date REVERT_SKEW_DATE(Date originalDate, Integer skewAmount)
```

Parameters

Parameter	Data Type	Description
originalDate	Date	Original date to unskew.
skewAmount	Integer	The number of days by which to decrement the original date.

Returns

Returns a Date object that contains the original date skewed by subtracting `skewAmount` days from the date.

1.4.2.17. REVERT_SKEW_INTEGER

Performs the opposite function of See "SKEW_INTEGER", allowing you to revert the result. `REVERT_SKEW_INTEGER` performs the opposite operation from what you specify for type. For example, if you specify Add, the method performs a subtraction operation.

Syntax

```
BigDecimal REVERT_SKEW_INTEGER(BigDecimal originalNumber, String action, Integer skewAmount)
```

Parameters

Parameter	Data Type	Description
originalNumber	BigDecimal	Base value to be skewed
action	String	A string that specifies the method to use to skew the base value specified by target. The type is one of the following: • Add—subtracts `skewAmount` from `originalNumber` • Subtract—adds `skewAmount` to `originalNumber` • Mulitply—divides `originalNumber` by `skewAmount`. • Divide—multiplies `originalNumber` by `skewAmount`
skewAmount	BigDecimal	The amount by which to skew the base value.

Returns

Returns a BigDecimal value that is skewed from the number input to the method by the skew amount, using the specified action, where the action performed is the opposite of the action specified.

1.4.2.18. SKEW_DATE

Returns a skewed date based on an original date plus a specified number of days.

Syntax

```
Date SKEW_DATE(Date originalDate, Integer skewAmount)
```

Parameters

Parameter	Data Type	Description
originalDate	Date	Original date to skew.
skewAmount	Integer	The number of days by which to increment the original date.

Returns

Returns a Date object that contains the original date skewed by adding `skewAmount` days to the date.

1.4.2.19. SKEW_FLOAT

Skews a float value. The skewed value is created from an original float value plus a user-specified number.

SKEW_FLOAT is not reversible (due to possible truncation of values).

Syntax

```
BigDecimal SKEW_FLOAT(BigDecimal originalNumber, String action, BigDecimal skewAmount)
```

Parameters

Parameter	Data Type	Description
originalNumber	BigDecimal	Value to be skewed
action	String	A string that specifies the method to use to skew `originalNumber`. The action is one of the following: • Subtract—subtracts `skew_amount` from `originalNumber` • Add—adds `skewAmount` to `originalNumber` • Mulitply—multiplies `originalNumber` by `skewAmount`. • Divide—divides `originalNumber` by `skewAmount`
skewAmount	BigDecimal	The amount by which to skew the original number.

Returns

Returns a BigDecimal value that is skewed from the number input to the method by the skew amount, using the specified action.

1.4.2.20. SKEW_INTEGER

Skews an integer value. The skewed value is created from an original number skewed by adding, subtracting, multiplying, or dividing.

Syntax

```
BigDecimal SKEW_INTEGER(BigDecimal originalNumber, String action, Integer skewAmount)
```

Parameters

Parameter	Data Type	Description
originalNumber	BigDecimal	Base value to be skewed
action	String	A string that specifies the action to use to skew `originalNumber`. The action is one of the following: • Add—adds `skewAmount` to `originalNumber`. • Subtract—subtracts `skewAmount` from `originalNumber.` • Mulitply—multiplies `originalNumber` by `skewAmount`. • Divide—divides `originalNumber` by `skewAmount`.
skewAmount	Integer	The amount by which to skew the original number value.

Returns

Returns a BigDecimal value that is skewed from the number input to the method by the skew amount, using the specified action.

1.4.2.21. SKEW_PERCENT

Returns a skewed value where the original number is increased by a specified percentage.

Syntax

```
Float SKEW_PERCENT(Float originalNumber, Float skewPercent)
```

Parameters

Parameter	Data Type	Description
originalNumber	Float	The value to skew.
skewPercent	Float	The percentage of the original number to be added to skew the number. Specify whole number for this parameter instead of decimals. For example, to skew the number by 10%, specify 10 for this parameter, not 0.10.

Returns

Returns originalNumber * (1 + skewAmount)/100 as a Float value.

1.4.2.22. VALID_ABA_ROUTING_NUMBER

Validates the ABA routing number.

Syntax

```
String VALID_ABA_ROUTING_NUMBER(abaNumberToValidate)
```

Parameters

Parameter	Data Type	Description
abaNumberToValidate	String	The nine digit ABA routing number.

Returns

Returns true when the ABA routing number is valid. Otherwise, returns false.

1.4.2.23. VALID_CREDIT_CARD_NUMBER

Validates credit card numbers using a credit card checksum algorithm.

Syntax

```
String VALID_CREDIT_CARD_NUMBER(String creditCardNumber, Integer Length)
```

Parameters

Parameter	Data Type	Description
creditCardNumber	String	The credit card number to be validated. Include only digits–no dashes or spaces.
Length	Integer	The length of the credit card number.

Returns

Returns true when the input credit card number is valid. Otherwise, returns false.

1.4.3. Extensions

If necessary, you can extend SDM functionality by writing extensions using Groovy programming language.

Web Console extensions

Web Console provides you an easy way of accessing SDM functionality through a various sub-modules (or the tiles). You can have your own tile on Web Console home page for an extension written by you. Web Console extensions are written using Groovy and Groovy Server Pages. Here is a simple “hello world” example to demonstrate such an extension

1. Create a Groovy program and GSP, for example:

   `index.groovy` contains:

   ```
   String helloworld = "Hello World: I'm a WebConsole extension!"
   [helloworld: helloworld]
   ```

   `index.gsp` contains:

   ```
   <html>
    <head>
    <title>WebConsole Hello World</title>
    </head>
    <body>
    <h1>${helloworld}</h1>
    <hr>
    <h3><a href="/WebConsole">Home</a></h3>
    </body>
   </html>
   ```

2. In the OpenText™ Structured Data Manager application data directory, create a subdirectory for your program files under `extensions/webconsole/menus/main`. For example:

   ```
   C:\SDM\OBTHOME\extensions\webconsole\menus\main\Hello World
   ```

3. Copy your Groovy and GSP files into the newly created directory.
4. Log into the Web Console. On the main page, you should see your newly added extension.
5. Click **Hello World**.
6. Click **Home**.

Tile and Access Control

Above example showed just a simple approach of adding your own extension to Web Console. However, this is good for simple extensions but for sophisticated extension you may like to change the icon, title and description displayed on Web Console home page as well as control the users who can access functionality of your extension. This can be done by providing details in `extension.groovy` file. Here is a sample file (also available in `<OBT_HOME>/extensions/WebConsoleExtensionDemo.zip`):

Note

SDM prior to 7.65 provided only icon, title and description changes through `index.properties` file but now you can provide role based access through `extension.groovy`. Prior approach also works but it is recommended to switch to newer approach of `extension.groovy`.

```
name = 'Extension Demo'
shortName = 'DEMO' // Should be 3 to 5 characters long (used
 // to prefix the role names, please refer
 // to roles section below), if more than
 // 5, it is truncated to 5 characters,
 // use capital letters (A to Z) only.
accessRole = 'ROLE_DEMO_USER' // Here we specify the role to
 // which the access to this extension/
 // module is granted by default, this is
 // must
tile {
 title {
 en = 'Extension Demo'
 hi = 'Extension प्रदर्शन'
 }
 icon = '...'
 description {
 en = 'This demonstrates Web Console extension'
 hi = 'यह Web Console एक्सटेंशन प्रदर्शित करता है'
 }
 help='/WebConsole/static/help/Content/extension/ext.htm'
}
roles {
 ROLE_DEMO_USER {
 privilege {
 en = 'Access extension demo'
 hi = 'Extension प्रदर्शन पहुँच'
 }
 groups = ['Administrators', 'Users'] // Default groups to which
 // this role is assigned.
 }
 ROLE_DEMO_ADMIN {
 privilege {
 en = 'Extension demo administrative'
 hi = 'Extension प्रदर्शन प्रशासनिक'
 }
 groups = ['Administrators']
 }
}
obsoleteRoles = ['ROLE_DEMO_MANAGER'] // Roles from the
 // previous version which are not in use
 // and are to be deleted.
 // Please note that this will impact the
 // groups to which this role is assigned.
```

Access Control in GSP and Groovy

Roles for the current users are loaded into the spring security & we can make use of it to check/restrict users from accessing the content.

For example,

1. To check if the user has a particular role.

```
if (grails.plugin.springsecurity.SpringSecurityUtils.ifAnyGranted('ROLE_DEMO_ADMIN',
'ROLE_DEMO_USER')) {
// Statements
}
```

2. To redirect the user to access denied page, throw ForbiddenException

```
if (!grails.plugin.springsecurity.SpringSecurityUtils.ifAnyGranted('ROLE_DEMO_ADMIN',
'ROLE_DEMO_USER')) {
 throw new javax.ws.rs.ForbiddenException("User doesn't have required permissions");
}
```

3. Use spring security taglib to restrict the user from accessing a DOM element in GSP file.

```
<sec:ifAnyGranted roles="ROLE_DEMO_ADMIN, ROLE_DEMO_USER">
<button type="submit">Save</button>
</sec:ifAnyGranted>
```

Runtime extensions

1. Create a Groovy program. For example, **rm.groovy**
2. In the OpenText™ Structured Data Manager application data directory, create a subdirectory for your program files under `extensions/runtime/storageadapters`. For example:

   `C:\SDM\OBTHOME\extensions\runtime\storageadapters`
3. Copy your Groovy files into the newly created directory.
4. Log into the Web Console. Navigate to D2F environment created.
5. Click on **Active Environment**. Go to **Locations** > **New**.
6. You should be able to view a new location type **rm**.

Customize extensions

The tile on the landing page displays the information about the extension based on the details provided in the index.properties file. To customize the extension:

1. Create or edit the index.properties file in the following directory path:

   <OBT_HOME>\extensions\webconsole\menus\main\<Name_Of_Extension>\
2. Add the below details in the index.properties file:
   - Icon: Enter the base64 string which need to be converted to an image. For example:

     ```
     < img src="$
     {ICON}
     " alt="">
     ```
   - Title: Enter the title of an extension instead of directory name.
   - Description : Provide the description of the extension instead of: Application Extension: directory name
   - Help: Provide the URL to access extension related help topics. This URL appears as "Find Out More" link text in the appropriate tile on the landing page.

1.4.4. Custom Data Masks

Apart from the built-in data masks available with SDM, you can have custom data masks in a similar way through Groovy scripts. While defining the custom data masks ensure the following points:

1. Each of the custom data masks has a unique id.
2. The `configuration` field can be used to provide `id`, `function`, `displayName`, and `description`.
3. The Groovy script can reside under `<OBT_HOME>/extensions/runtime/masking` or its sub-folders. Keeping them in sub-folders help in grouping the related data masks. For example, string data masks are under string sub-folder and number data masks are under number sub-folder.
4. You can separate the implementation from configuration to reuse the code. In other words, you can keep `mask` and `revert_mask` methods in a separate file and specifying the same as function. Such reusable file names should start with '_' so that they don't get displayed to the user for masking function selection. You may like to refer to built-in `SkewNumberByAddition.groovy` and `SkewNumberByMultiplcation.groovy` Groovy scripts as both of them refer to a common implementation `_skewNumber.groovy` (these files are under number sub-folder). To use a function from another sub-folder you can use dot (.) separator same as Groovy package convention.
5. The design time parameters are to be specified as annotations to `configuration` field and the `mask` and `revert_mask` methods as described in the following section. These parameters can be validated through built-in validators (as described in the comment of the example below).

In a simple approach you can create `mask` and `revert_mask` methods in a Groovy file to which the original value of a given column is passed as a Java Object. The method signatures are as below:

```
Object mask(Object in)
Object revert_mask(Object in)
```

In this case file name (without the extension) will be used for `id`, `function`, `displayName`, and `description`. However, you by providing the details explicitly in the file through `configuration` field you can be much more specific and user can be better informed.

In an elaborate approach, where the user defined parameters are required the method signature will be as below:

```
Object mask(Object in, List<String> params)
Object revert_mask(Object in, List<String> params)
```

The `params` will have the same order as what is defined in the configuration. This way the developer can use the respective 0 based index to get value of specific parameter.

Additionally, the SDM passes the following context information to custom groovy functions. This helps you to change the data of a given column based on the other column values.

- `tableName`: Name of the table on which the `mask/revert_mas`k method is applied.
- `colName`: Name of the column for the field on which the `mask/revert_mask` method is applied.
- `row`: Details of the row on which `mask/revert_mask` method is applied. It is a `LinkedHashMap` of key-value pair of Column Name and `DBParam` object.

For more information, refer to `PrefixLastName.groovy` for a sample code on the use of context variables.

Note

- If you create a functions under sub-directory of `<OBT_HOME>/extensions/runtime/masking` then you need to specify the package in the Groovy file. Refer to the example below, the example files reside under `<OBT_HOME>/extensions/runtime/masking/number` directory.
- Once you add or remove any function to `<OBT_HOME>/extensions/runtime/masking` or its sub-folders it is recommended to re-compile the code using `compileGroovyExtensions.bat/.sh` script to have good design time and runtime performance for masking functions.

Example

The following example shows creating custom data mask with parameter. In this example the amount by which the number is skewed is taken as a parameter during design time. Also there is a hidden parameter specifying the type of the function (addition – '+' or multiplication '*') to apply. This is a comprehensive example catering to all the necessary details required for you to define any of the custom data mask. So, pay attention to the comments as well. You may like to refer to `SkewNumberByMultiplication.groovy` to get an idea on reuse of `_skewNumber.groovy's` `mask` and `revert_mask` method implementation.

File: SkewNumberByAddition.groovy

```
package number

import com.outerbay.foundation.services.groovy.annotation.DataMasking
import com.outerbay.foundation.services.groovy.annotation.DataMaskingParam
import com.outerbay.foundation.services.groovy.annotation.DataMaskingParams
import groovy.transform.Field

// @DataMasking annotation is used to provide details for the function level
@DataMasking(id = "skew_number_add", // This should be unique across all the
 // masking functions
function = "number._skewNumber", // This is optional, if the mask and
 // revert_mask methods are in another file
 // then only you need to specify this
 // parameter. If the function file is in
 // another directory then you need to specify
 // relative path from masking directory
 // (separated by ., i.e. package convention)
displayName = "Skew number: add", // The name to display in the
 // designer
description = "Skews a given number by adding a specified amount.",
 // At present not in use but we will think of
 // using this to provide more information to
 // the user
trueReversible = true, // This is optional, indicates whether masking
 // function is reversible is not. Set this flag
 // to true only for the functions which are
 // truly reversible.
 // If not specified then the system decides
 // reversibility based on revert_mask function
 // but in that case SDM will mark it as
 // "presumed reversible"
validator = "System.none") // This is optional, can be used to do
 // validation across all the parameters for
 // example, if your parameters are min and
 // max and you want to make sure that
 // min < max then you can use
 // System.rangeValidator
// @DataMaskingParams annotation is used to provide details for the function
// parameters, as current version of Groovy does not support repeatable
// annotations. So, this annotation is required to wrap @DataMaskingParam, when
// we move to higher version of Groovy, we can use @DataMaskingParam multiple
// times to provide details for all the parameters
@DataMaskingParams([@DataMaskingParam(name = "Plus",
 defaultValue = "+"),
@DataMaskingParam(name = "Amount", // Parameter name, this is optional
 label = "Amount", // Label to use for the parameter in the designer, if
 // not specified then the parameter is hidden.
 defaultValue = "0", // The default value for the parameter
 validator = "System.numberValidator")]) // The validator to be used,
 // Currently following can be used:
 // System.doubleNumberValidator
 // System.lengthValidator
 // System.numberValidator
 // System.positiveNumberValidator
 // System.none
// Declare configuration as a field as all the above annotations are applied to
// this field
@Field configuration
```

File: _skewNumber.groovy

```
package number

import com.outerbay.common.OBTLog
import java.math.RoundingMode
/**
* Mask the number either by adding, subtracting, multiplying,
* dividing by the skewed amount.
* @param value Input value that needs to be masked.
* @param params List of Strings having an element which specifies the
* number is to be skewed
* @return Object number obtained by applying the specified action on
* the input.
*/
Object mask(Object value, List<String> params) {
 if (value == null) {
 return null
 }
 BigDecimal input
 if (value instanceof BigDecimal) {
 input = (BigDecimal) value
 } else if (value instanceof String) {
 try {
 input = new BigDecimal(value)
 } catch (NumberFormatException e) {
 OBTLog.warn("_skewNumber could not be applied to" +
 " non-number " + value)
 return value
 }
 } else if (value instanceof Integer || value instanceof Long ||
 value instanceof Float || value instanceof Double) {
 input = new BigDecimal(value)
 } else {
 OBTLog.warn("_skewNumber can not be applied to " +
 value.class.name)
 return value
 }
 final String action = params[0]
 final BigDecimal skew = new BigDecimal(params[1])
 skewNumber(input, action, skew)
}
/**
*
* @param value number need to be unmask
* @param params List of Strings having an element which specifies the
* operation
* @return Object as an unmask value
*/
Object revert_mask(Object value, List<String> params) {
 def rParams = []
 switch (params[0]) {
 case '+':
 rParams << '-'
 break
 case '-':
 rParams << '+'
 break
 case '*':
 rParams << '/'
 break
```

```
 case '/':
 rParams << '*'
 break
 }
 rParams << params[1]
 mask(value, rParams)
}
/**
*
* @param val Input value that needs to be masked.
* @param action Either Addition, Subtraction, Multiplication or
* division
* @param skewAmount other operand needed for action. This is an
* BigDecimal argument in order to make this function
* reversible.
* @return BigDecimal number obtained by applying the specified action
* on the input.
*/
private BigDecimal skewNumber(BigDecimal val, String action,
 BigDecimal skewAmount) {
 BigDecimal ret = val;
 switch (action) {
 case "+":
 ret = val + skewAmount
 break;
 case "-":
 ret = val - skewAmount
 break;
 case "*":
 ret = val * skewAmount
 break;
 case "/":
 if (skewAmount != BigDecimal.ZERO) {
 ret = val.divide(skewAmount, 0, RoundingMode.HALF_UP);
 }
 break
 }
 ret
}
```

For more info, visit https://docs.microfocus.com