Libraries and Data Source Connectors

Best practices

North & South America
CloverDX Inc.
1101 Wilson Blvd
6th Floor
Arlington VA 22209
United States

EMEA/APAC
Javlin Ltd. / CloverDX
46-48 Red Lion Court
Park St
London SE1 9EQ
United Kingdom

Czech Republic
Prague
CloverDX a.s.
Vinohradská 174
130 00 Prague
Czech Republic

www.cloverdx.com
info@cloverdx.com

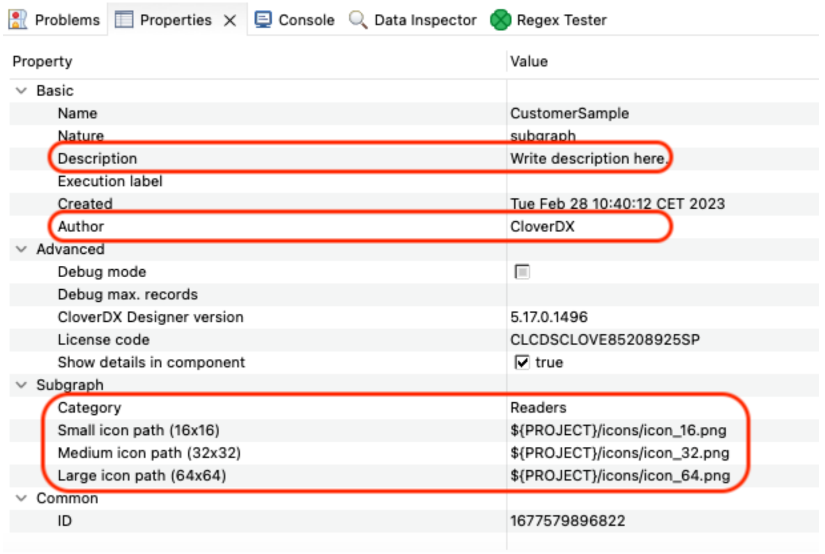
Table of Contents

Creating beautiful, understandable data sources

Data sources can be enhanced to make them more understandable and user-friendly. This involves adding clear and concise descriptions, labels, icons, and other information that will be displayed in the Data Catalogue. By doing so, you can present your resources in a professional and appealing way.

The following is a list of possibilities for customizing your Data Source Connectors:

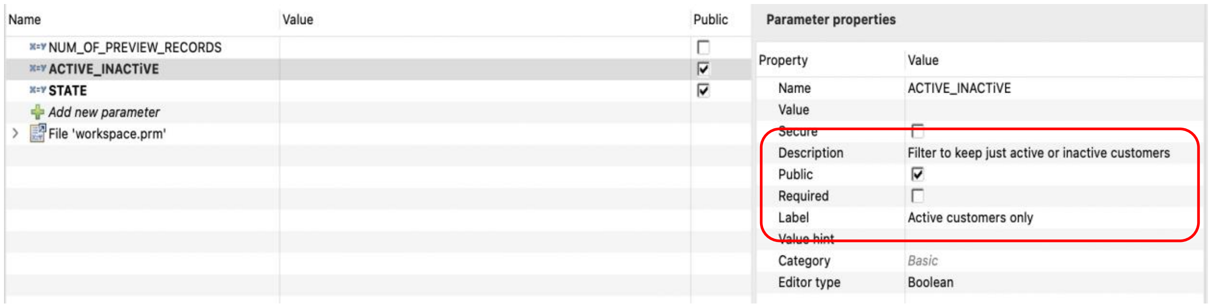
- **Name** of the Data Connector shown in the Data Catalog is taken from the file name of the subgraph.
- **Description** can be set in Properties → Description (click an empty area of a graph/subgraph and then navigate to Properties tab at the bottom of your screen). This description will visible under the name of the Data Source in Data Catalog.
- Use **icons** and subgraph **categories** to enhance look-and-feel, adhering to built-in conventions, such as green for readers and blue for writers.
  - Create a separate directory `${PROJECT}/icons` to store icons used in subgraphs.
  - Use properly sized PNG icons, including 64 x 64 px, 32 x 32 px, and 16 x 16 pixels.
- CloverDX graph/subgraphs stores information about author. It could be updated only in source code (internal XML).



Property	Value
Basic	
Name	CustomerSample
Nature	subgraph
Description	Write description here.
Execution label	
Created	Tue Feb 28 10:40:12 CET 2023
Author	CloverDX
Advanced	
Debug mode	☐
Debug max. records	
CloverDX Designer version	5.17.0.1496
License code	CLCDSCLOVE85208925SP
Show details in component	✓ true
Subgraph	
Category	Readers
Small icon path (16x16)	${PROJECT}/icons/icon_16.png
Medium icon path (32x32)	${PROJECT}/icons/icon_32.png
Large icon path (64x64)	${PROJECT}/icons/icon_64.png
Common	
ID	1677579896822

Properties tab of a subgraph with Description, Author, Category and Icon

- Provide human-readable **labels and descriptions for graph parameters** (Edit parameters in Outline)



Name	Value	Public
NUM_OF_PREVIEW_RECORDS		☐
ACTIVE_INACTiVE		☑
STATE		☑
Add new parameter		
File 'workspace.prm'		

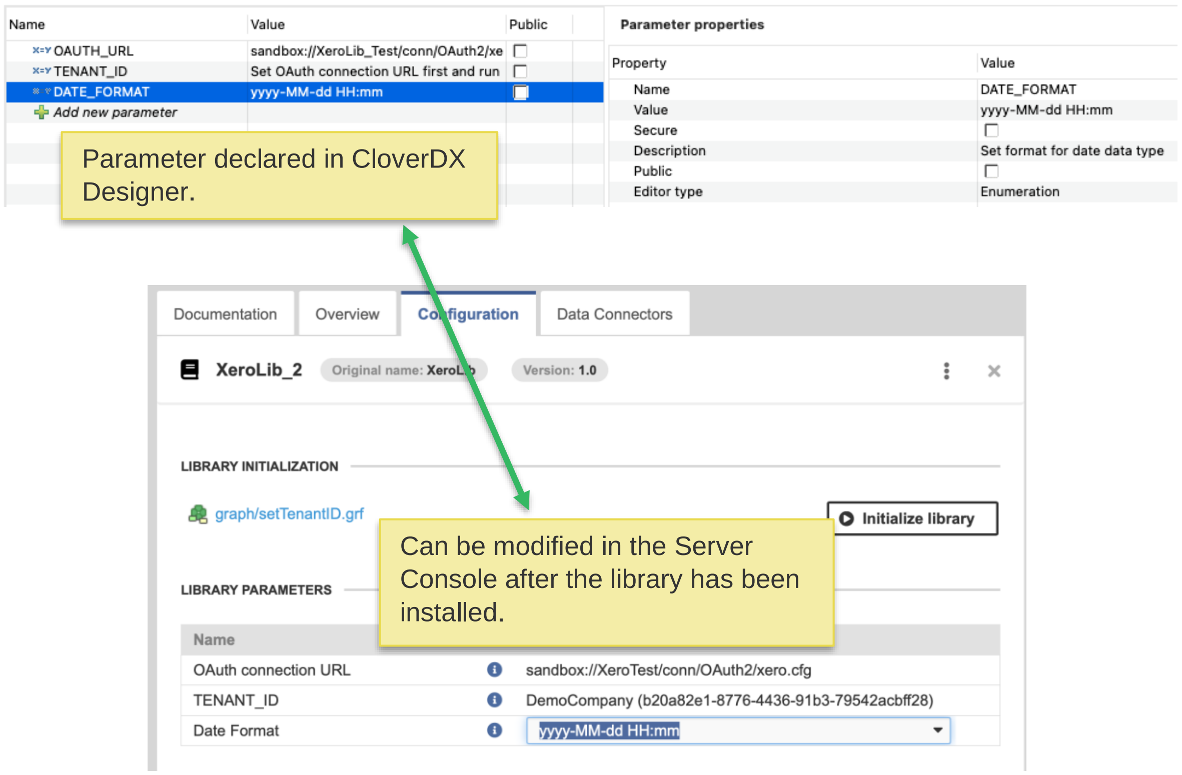
Parameter properties

Property	Value
Name	ACTIVE_INACTiVE
Value	
Secure	☐
Description	Filter to keep just active or inactive customers
Public	☑
Required	☐
Label	Active customers only
Value hint	
Category	Basic
Editor type	Boolean

Edit parameters dialog with Description and Label

- The labels and descriptions for graph parameters will appear in multiple locations, including the library configuration dialog in CloverDX Server UI, parameter mapping dialog when using a component from the Library in CloverDX Designer, and CloverDX Wrangler UI.
- There is a special parameter file `library.prm`. The file is not created automatically. However, if you create this parameter file in your project, all parameters from this file will be available in CloverDX Server UI for settings.
- This is a global configuration file that the administrator can view and modify values of parameters.
- If values are set in this parameter file, they will be used in the execution. However, the parameters could be overridden when the graph is executed. In that case, values specified in `library.prm` could be considered as default values.

  Here you can see example of three graphs parameters in CloverDX Designer and the corresponding view from CloverDX Server UI.



Name	Value	Public
OAUTH_URL	sandbox://XeroLib_Test/conn/OAuth2/xe	☐
TENANT_ID	Set OAuth connection URL first and run	☐
DATE_FORMAT	yyyy-MM-dd HH:mm	☐
Add new parameter		

Parameter properties

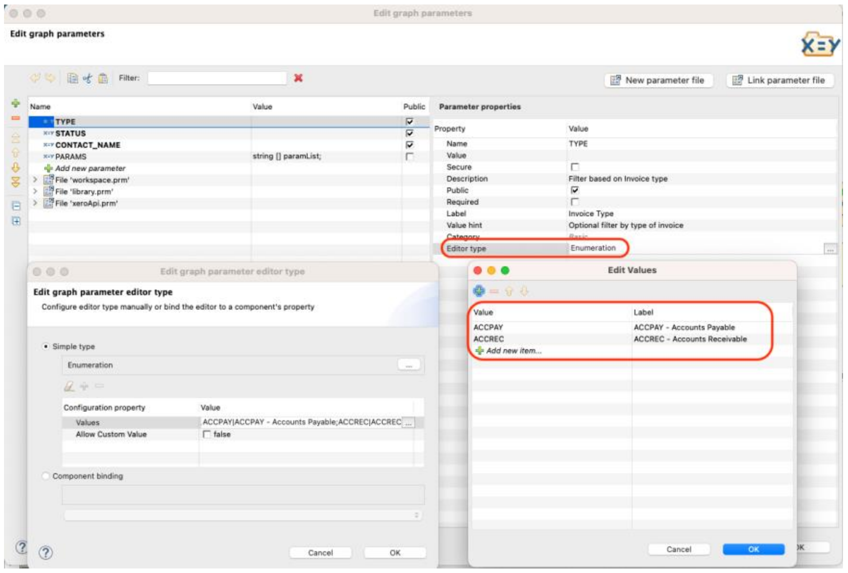
Property	Value
Name	DATE_FORMAT
Value	yyyy-MM-dd HH:mm
Secure	☐
Description	Set format for date data type
Public	☐
Editor type	Enumeration

Parameter declared in CloverDX Designer.

Can be modified in the Server Console after the library has been installed.

Name		
OAuth connection URL	ℹ	sandbox://XeroTest/conn/OAuth2/xero.cfg
TENANT_ID	ℹ	DemoCompany (b20a82e1-8776-4436-91b3-79542acbff28)
Date Format	ℹ	yyyy-MM-dd HH:mm

- Use appropriate **Editor types** for your parameters, such as enums, numeric, or boolean, to simplify the library setup process for administrators. Descriptions are visible as a tooltip on the info (i) icon.
-



Name	Value	Public
TYPE		☑
STATUS		☑
CONTACT_NAME		☑
PARAMS	string [] paramList;	☐
Add new parameter		
File 'workspace.prm'		
File 'library.prm'		
File 'xeroApi.prm'		

Parameter properties

Property	Value
Name	TYPE
Value	
Secure	☐
Description	Filter based on Invoice type
Public	☑
Required	☐
Label	Invoice Type
Value hint	Optional filter by type of invoice
Category	Basic
Editor type	Enumeration

Edit graph parameter editor type

Configure editor type manually or bind the editor to a component's property

(●) Simple type

Enumeration

Configuration property	Value
Values	ACCPAY|ACCPAY - Accounts Payable;ACCREC|ACCREC
Allow Custom Value	☐ false

( ) Component binding

Edit Values

Value	Label
ACCPAY	ACCPAY - Accounts Payable
ACCREC	ACCREC - Accounts Receivable
Add new item...	

Edit parameters dialog, using enumerations will help users provide correct values

- Typically, the parameters are connection details (e.g., to a database or an external service - API keys, passwords).
- Consider which library parameters will be visible to the administrator who can then provide global settings for your library. This allows users to utilize the library without needing to have in-depth knowledge of all the settings.
- Some of the parameter values, such as connection details, could be necessary for the library initialization.
- If a Data Source Connector public parameter is defined, it will be visible in CloverDX Wrangler. To assist Wrangler users in selecting and using your Data Source Connector, provide a label, a description, and an editor type for this parameter. If possible, use enums to prevent incorrect user input.

Initializing libraries (pre-generating metadata)

A library can contain a so called "initialization job" which can be used to perform initial setup of the library after installation. For example, generic libraries will have initialization jobs that pull custom metadata and other configuration based on a specific connection which is only available after the library has been installed and configured.

The initialization job can be any graph or jobflow that has been designated as an initialization job during export of the library.

A Library initialization job will automate setup of configuration that cannot be done when the library is created and must be performed after the library is installed/instantiated (e.g., generating metadata based on provided connection to source system). It is usually executed by CloverDX Server administrator during Library installation & initialization as one of the last steps or on demand anytime later (when a refresh is needed due to changes in configuration(s)). Initialization graph could also be executed automatically via Scheduler defined on CloverDX Server (may be used for periodical updates of dynamic structures within installed Library).

Init graphs can also rewrite default parameter files such as `library.prm` with some pre-defined values.

Example of when and how to use the initialization graph:

- Your library connects to a cloud-based CRM system, e.g. to Deals, Customers, or Contacts.
- The entities in the CRM have custom fields, so you need to authenticate to your specific account and then read the entities, including your custom fields.
- As a first step, configure the connection details to the CRM system. This setting can be done in the CloverDX Server UI in the library configuration section.

- The next step is to run the initialization graph, which establishes a connection to the target system, reads the structure, and generates metadata of the entities. This metadata is then stored within the library.
- From this point on, users of the library (in CloverDX project or in Wrangler) can access the auto-propagated metadata in the same way as it is defined in your CRM system.
- In the settings, you can specify a list of required fields, in case you only need a subset of all the fields.
- Additionally, the initialization graph has the potential to change the `library.prm` file.

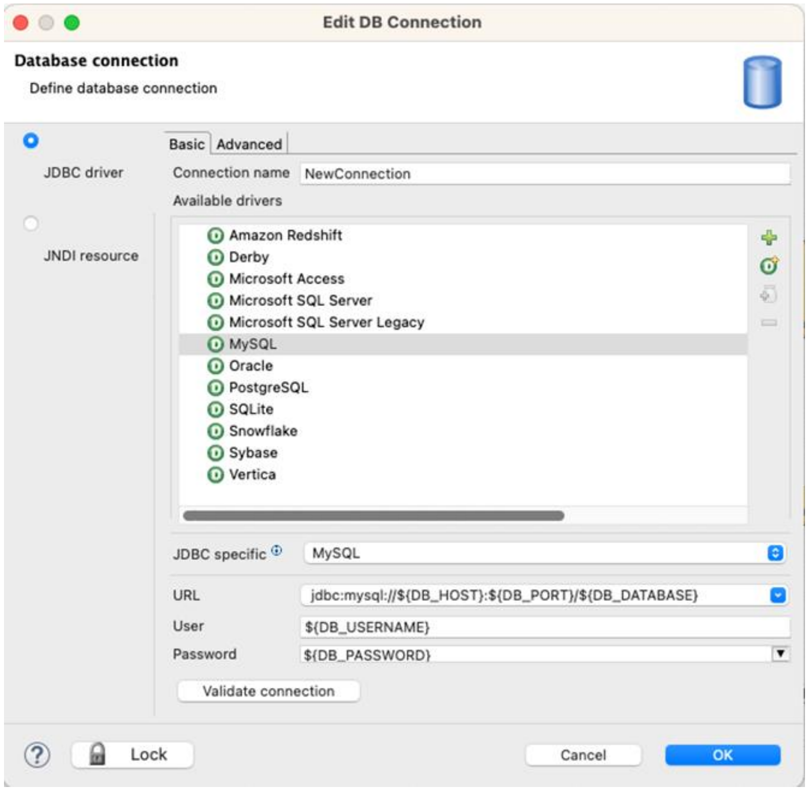
Managing user credentials and secrets

If your library requires user credentials or secrets, such as private tokens, it's important to store these parameters securely. When you use secrets as a graph parameter (e.g., in the `library.prm` file), you can mark them as 'secure' to protect them from unauthorized access and comply with security best practices. Please note that you will need to set a Master password in CloverDX Server to enable secure parameters. This password is used to encrypt and decrypt the secure parameters.

Another option is to use secret managers to retrieve the value of the parameter from a secure storage. This can be useful if you don't want to store sensitive data in the `library.prm` file. For more information about using secret managers, please refer to the documentation.

How to handle Database/JDBC Connections

- To use a database connection in your Library, provide all the connection details as parameters. This will make your Library more flexible and reusable.
- Specify the connection parameters in the CloverDX connection details. This will ensure that your Library can connect to the database without errors.



Connection name: NewConnection

Available drivers:
- Amazon Redshift
- Derby
- Microsoft Access
- Microsoft SQL Server
- Microsoft SQL Server Legacy
- MySQL
- Oracle
- PostgreSQL
- SQLite
- Snowflake
- Sybase
- Vertica

JDBC specific: MySQL

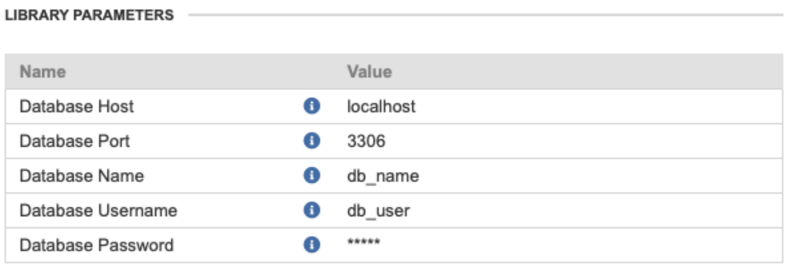
URL: jdbc:mysql://${DB_HOST}:${DB_PORT}/${DB_DATABASE}

User: ${DB_USERNAME}

Password: ${DB_PASSWORD}

Edit DB Connection dialog – parameters used for the connection details.

- Store the connection parameters in a `library.prm` parameter file. This will allow you to set the database connection in the CloverDX Server UI and manage it centrally.
- Use secure parameters for any sensitive information, such as passwords. This will protect your data from unauthorized access and comply with security best practices.
- If possible, let users choose the JDBC connection name. This will give them more control over the database connection and avoid conflicts with other connections.



LIBRARY PARAMETERS

Name	Value
Database Host	localhost
Database Port	3306
Database Name	db_name
Database Username	db_user
Database Password	*****

Connection parameters from Server UI, the values will be propagated into the connection.

OAuth2 connections in Data Sources

When your library needs to connect to a cloud-based system, it often requires an OAuth2 connection. In such cases, it's recommended to use an external connection from your library. The file URL for the connection can be defined using a graph parameter, which allows for greater flexibility and easier configuration.

After you create the connection, you can specify the path to the connection in the library configuration. This approach simplifies the process of setting up and managing connections and allows your library to seamlessly integrate with external systems.

To use OAuth2 connection in your library, you can follow these steps:

Check out a video tutorial on how to set up OAuth2 connection for a library at https://www.youtube.com/watch?v=8R5ZUAoYE7I&ab_channel=CloverDX

Quick summary

- Firstly, create an external OAuth2 connection in the server's sandbox and authorize it (from CloverDX Designer).
- After that, create a parameter in `library.prm` that specifies the File URL of the connection. The URL should be in the following format: `sandbox://<sandbox_code>/conn/OAuthConnection.cfg`
- Finally, use the created parameter in your library code to access the OAuth2 connection.

Optimizing performance

The Data Source Connector should implement a preview mode in which only a limited number of records is requested by the Server, thus allowing the connector to save API calls or data transfers. If the Connector ignores the preview mode, the initial use of the Data Source will have delays in the user interface.

The preview mode is indicated to the Connector from the Server by passing a parameter called NUM_OF_PREVIEW_RECORDS.

The Data Source Connectors utilized in Wrangler require the publication of a specific graph parameter known as NUM_OF_PREVIEW_RECORDS. This parameter is critical in enabling Wrangler to display a preview of the data sourced from the connector. It is essential to ensure that this parameter is utilized correctly within the Data Source Connector to restrict the number of records returned. Accurate implementation of this record limitation is crucial in achieving speedy previews.

When returning records, Wrangler works with the number of records retrieved. If the number of records is less than or equal to the requested number, Wrangler assumes that the preview represents the entire dataset. If a Data Source Connector only returns a random number of records, Wrangler users may be misled into thinking that only that quantity of records is available.

To optimize performance, Wrangler caches the preview in its internal memory and thus does not require data retrieval each time the preview is refreshed. However, for cloud-based systems, it is crucial to consider API limit handling mechanisms such as retries and timeouts.

Installing multiple instances of a single data source

- CloverDX Server provides the capability of installing multiple instances of a single library. This feature can be useful when there is a need to set up different instances of a library with varying configurations. For example, it is possible to prepare a data source for different groups of users such as "Customers for Finance" and "Customers for Marketing". In addition, different instances of the same library could potentially return different metadata sets, such as "Essential" vs "Full Record".
- Furthermore, it is possible to assign different permissions to different instances for specific user groups.
- However, the downside to this approach is that the same library or functionality is duplicated multiple times within the environment. To address this issue, it is recommended to extract common functionality and place it in a separate library, "core/base" library, which can be referenced by custom libraries that use this functionality.
- When another library is utilized as a dependency in a custom library, a parameter file named `dependencies.prm` is automatically create in the root directory. This file contains references to the dependent libraries and their respective versions. Typically, there is no need to take care about the file, it is created automatically in CloverDX Designer. If you need, you

can edit it and add labels and description to the parameters. This information will be visible in CloverDX Server UI and could help administrator during the library installation.

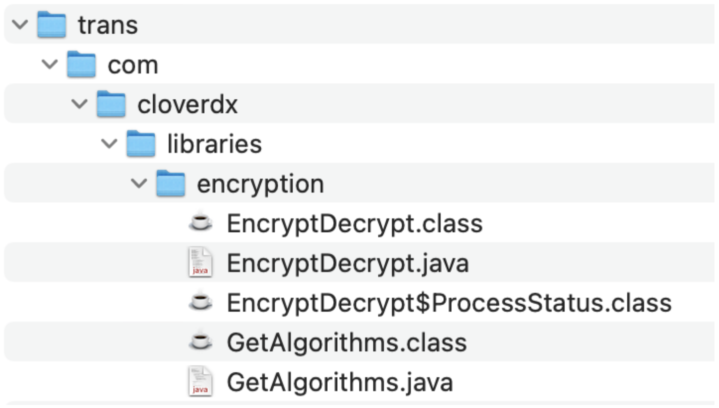
Live debugging and testing libraries

- When utilizing a subgraph from a library in CloverDX Designer, it behaves as a black box, and the user cannot view or modify the library's internal operations. However, in certain situations, it can be helpful to examine the execution directly within the library, especially when debugging the subgraph.
- To facilitate this process, it is possible to add the installed library from CloverDX Server to CloverDX Designer in the same way as connecting to any other server project. In CloverDX Designer, navigate to File -> New -> CloverDX Server Project, and provide the connection details to the server. In the following step, select the desired library. It is important to note that to import the library to CloverDX Designer, the user must have the necessary permission to manage libraries.
- Once the library has been added, the user can interact with it just like any other sandbox, including making direct changes to the codebase.

Using Java code in libraries

Sometimes projects require Java code to work. This code may include JAR files or custom classes (or both). To ensure proper functionality, please follow these rules:

- If you need to include a JAR file into your library, put it in the `lib` directory and add a reference to the classpath (in CloverDX Designer, go to: Project properties -> Java Build Path -> Libraries, click ok Add JARs… and select JAR file from your ${PROJECT}/lib directory. This applies especially to libraries and connectors that users cannot inspect. However, do not bundle JAR files that are already on the classpath, such as Bouncy Castle or certain database/JDBC drivers to avoid conflicts.
- If you need your own custom Java code to be used in the CloverDX library, put it into a proper package. The naming pattern should look like this:
  `com.<your company name>.libraries.<libraryName>.<className>`.
  Source code of the Java class should be stored in ${PROJECT}/trans directory. Java classes from this directory are compiled automatically using CloverDX Designer during a development (for every Java class in .java file, .class file will be created).



```
trans
  com
    cloverdx
      libraries
        encryption
          EncryptDecrypt.class
          EncryptDecrypt.java
          EncryptDecrypt$ProcessStatus.class
          GetAlgorithms.class
          GetAlgorithms.java
```

Example of structure of Java libraries in CloverDX project.

- Make sure you include all the .class files (compiled Java classes) in your CloverDX library (when exporting the library in Designer). Otherwise, the library will not be able to use the Java classes. If you use a versioning system, you may want to store the compiled classes in the repository (for example, if you use Git, do not exclude .class files in your .gitignore).

File structure of a Data Source Connector

Data Source Connectors are an integral part of the Library and are distributed in the form of a .clib file. The .clib file, in essence, is a zip archive of a CloverDX Project. As such, if you rename the file extension to .zip, you can extract the archive and gain access to the actual project contents. Additionally, you can import the project directly to your CloverDX Designer.

The Library is essentially a standard CloverDX project that contains some additional files with specific functionality.

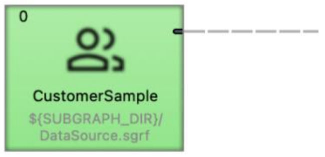
- `library.json`
  - Library descriptor file.
  - Automatically created when a CloverDX Library is created in CloverDX Designer.
  - It contains information about available public entitites, initialization graph, Data Source Connectors.
- `library.prm`
  - The Parameter file is used to define public parameters, of the Library.
  - This files is not create automatically.
  - All parameters included in this file are visible in the CloverDX Server UI and can be used for global configuration of the library, such as user credentials or constants.
  - It is recommended to use proper labels and descriptions for the parameters to help the administrator with the settings.
  - Please note that it is not necessary to set all parameters as public, as all parameters in the file will be included in the server UI by default.

- If you need to access these parameters, you can link the parameter file to your job file.
- `favicon.png`
  - Icon of the whole library.
  - It is visible in CloverDX Catalogue.
  - Use PNG file with a transparent background, maximum recommended width is 150px.
- `README.md`
  - This file should contain a description of the functionality and installation details in Markdown syntax.
  - The documentation from this file is visible in the CloverDX Server UI.
  - The README.md file is not created automatically.
  - An example of the README.md file can be found in the Template Connector.
- `dependencies.prm`
  - When you use another library as a dependency in your project, a parameter file named `dependencies.prm` is automatically generated in the root directory.
  - This file contains references to the dependent libraries and their versions.
  - If you need, you can edit it and add labels and description to the parameters. This information will be visible in CloverDX Server UI and could help administrator during the library installation.
- `icons/`
  - Directory dedicated to storing icons of subgprahs/Data Souce Connectors.

The rest of the project structure is the same as in any other CloverDX project.

Requirements for a Data Source Connector

- A unique name within the library.
- Only one output port that is used for outputting data into Wrangler
- The connector must declare output metadata.

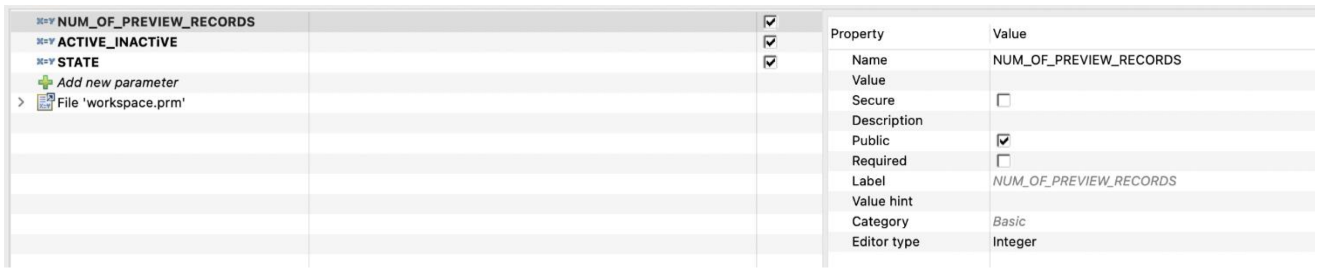


Preview of a connector – one output port with auto-propagated metadata

- Output metadata could be either auto-propagated or statically assigned.
  - If the metadata structure cannot be universally defined – e.g. it depends on a particular configuration of a source system (e.g. CRM systems are being customized) then you can generate the structure upon library installation via special purpose

CloverDX job - an "init graph" which can be executed manually or be part of a scheduled job – see below.

- Output metadata of a Data Connector cannot contain variant, map, list, byte or cbyte field types (due to Wrangler limitations).
- A public graph parameter called `NUM_OF_PREVIEW_RECORDS` that defines how many records to return when running in Wrangler's transformation design mode.
    - In your data source connector, implement getting of this preview as fast as possible by limiting the number of records early on. Users of Wrangler will wait for the the preview in the UI.

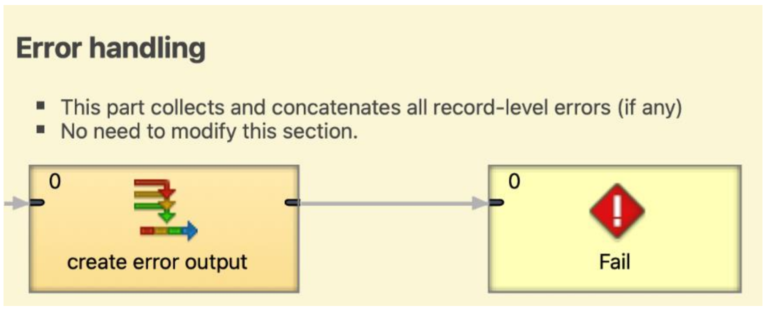


NUM_OF_PREVIEW_RECORDS		✓
ACTIVE_INACTIVE		✓
STATE		✓
Add new parameter		
File 'workspace.prm'		

Property	Value
Name	NUM_OF_PREVIEW_RECORDS
Value	
Secure	☐
Description	
Public	✓
Required	☐
Label	*NUM_OF_PREVIEW_RECORDS*
Value hint	
Category	*Basic*
Editor type	Integer

Parameters dialog – required NUM_OF_PREVIEW_RECORDS parameter

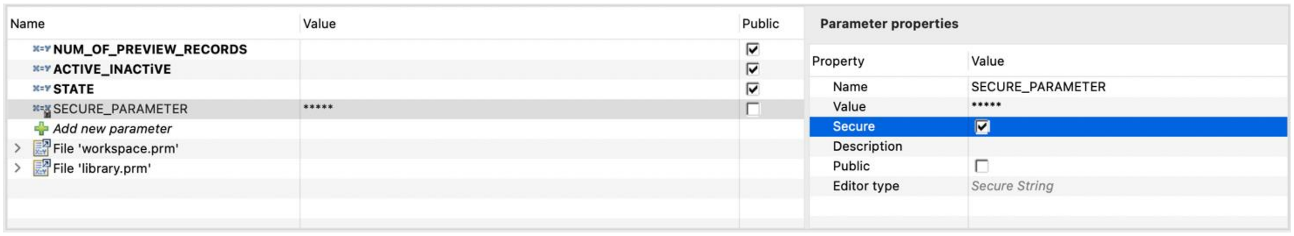
- Error handling with reasonable error messages propagated by Fail component.
- Alternatively, you can call function `raiseError( string message)` directly from CTL2 code. Again, please use descriptive error message.


Error handling
This part collects and concatenates all record-level errors (if any)
No need to modify this section.
0
create error output
0
Fail

Part of graph with error handling functionality

Additional recommendations for Data Source Connectors

- Use secure graph parameters for sensitive input parameters such as passwords or tokens. Be aware that secure parameters are encrypted with CloverDX Server master password and won't work on another instance with different master password.



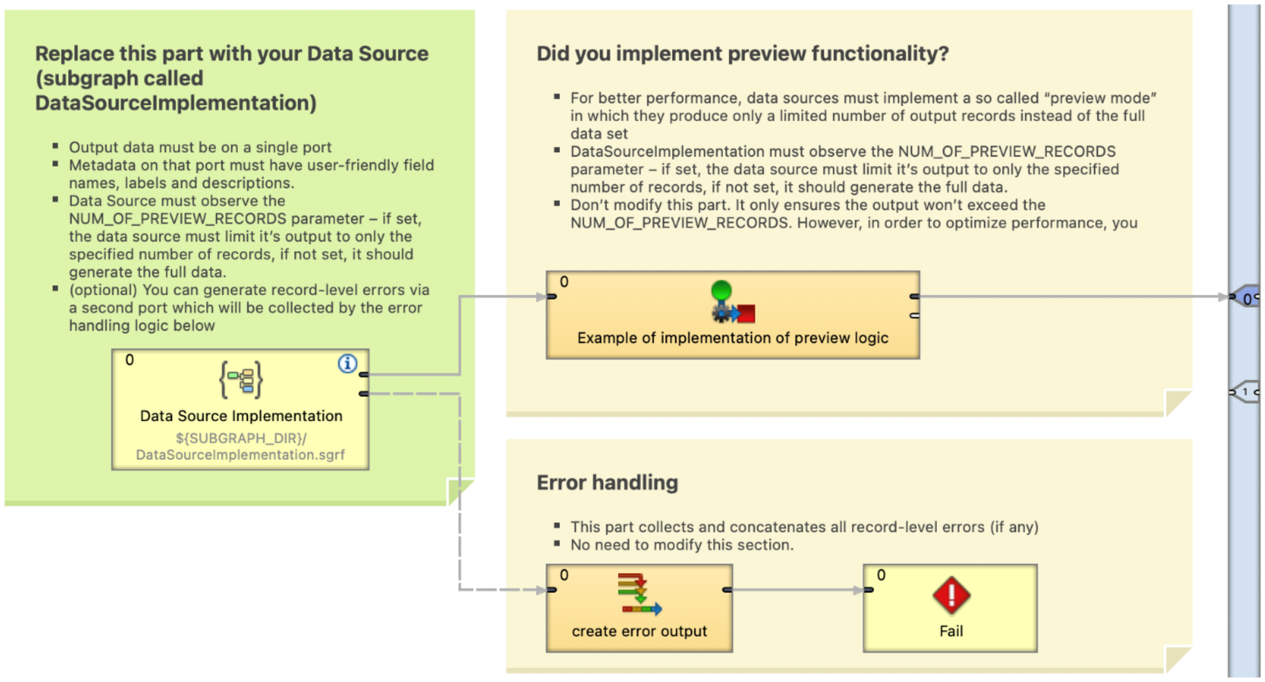
Name	Value	Public
NUM_OF_PREVIEW_RECORDS		✓
ACTIVE_INACTIVE		✓
STATE		✓
SECURE_PARAMETER	•••••	☐
Add new parameter		
File 'workspace.prm'		
File 'library.prm'		

Parameter properties

Property	Value
Name	SECURE_PARAMETER
Value	•••••
Secure	✓
Description	
Public	☐
Editor type	*Secure String*

Parameters dialog – usage of secure property settings

- Keep subgraphs small and focused on accomplishing a single task. Use several layers of subgraphs if needed (e.g. lower level subgraphs for core API calls, higher level subgraphs already specialized with business logic to simplify usage and eliminate complex or excessive configuration options). Don't try to create too universal connectors – create several simple ones instead. The CloverDX Data Catalog helps end users to navigate among options to find the one most suited to their need.



Replace this part with your Data Source (subgraph called DataSourceImplementation)

- Output data must be on a single port
- Metadata on that port must have user-friendly field names, labels and descriptions.
- Data Source must observe the NUM_OF_PREVIEW_RECORDS parameter – if set, the data source must limit it's output to only the specified number of records, if not set, it should generate the full data.
- (optional) You can generate record-level errors via a second port which will be collected by the error handling logic below

Did you implement preview functionality?

- For better performance, data sources must implement a so called "preview mode" in which they produce only a limited number of output records instead of the full data set
- DataSourceImplementation must observe the NUM_OF_PREVIEW_RECORDS parameter – if set, the data source must limit it's output to only the specified number of records, if not set, it should generate the full data.
- Don't modify this part. It only ensures the output won't exceed the NUM_OF_PREVIEW_RECORDS. However, in order to optimize performance, you

Error handling

- This part collects and concatenates all record-level errors (if any)
- No need to modify this section.

0
Data Source Implementation
${SUBGRAPH_DIR}/
DataSourceImplementation.sgrf
0
Example of implementation of preview logic
0
create error output
0
Fail
0
1

Example of a template connector

- Test thoroughly before publishing them for users.

Job Documentation Tips

Job documentation is optional but highly recommended for any public job in your library project. Job documentation helps users understand what your jobs do and how they work.

Here are some tips for writing good job documentation:

- All documentation should be written in English unless otherwise specified by your target audience or market requirements.
- Every public job in a library should have a Note with short explanation of what the job does. This note should be separate (not encapsulate any components) and be visible when the job is opened (i.e., not to far down or to the right - usually top-left corner of the job is best location).
- Difficult and non-obvious parts of each job should be explained in notes as well. Keep in mind that even for libraries users can look inside in Job Inspector or via Designer when debugging.
- Each public subgraph in a library project *must* have a description. This is configured via Description property of the job (please note, this is not a Note). This description is shown in Data catalogue and in the Server UI when browsing libraries and therefore it should be short (one or two sentences). Do not include too much detail here and place the most important info at the beginning of description.
- Design the Data Source Connector with Wrangler's join limitations in mind. Prepare data that meets the needs of Business users.
- For entities such as Customers, Contacts, Invoices, etc., include relevant references in the Data Source Connector.
  - For example, for an Invoice entity, join Customer and Contact information to provide more context in your Data Source Connector.
- Consult with business users about their requirements and offer custom solutions for their business cases.
- Use parameters for the Data Source Connector if there are filters (e.g., valid/invalid records). This allows business users to easily adjust the data set returned by the connector.
- Prefer enums for parameters when possible. This helps users avoid input errors.