TIBCO® Data Virtualization

Extensibility Guide

Version 8.0

Last Updated: *November 7, 2018*

Two-Second Advantage®

TIBCO®

Important Information

Contents

Preface

Documentation for this and other TIBCO products is available on the TIBCO Documentation site. This site is updated more frequently than any documentation that might be included with the product. To ensure that you are accessing the latest available help topics, please visit:

- htps://docs.tibco.com

Product-Specific Documentation

The following documents form the TIBCO® Data Virtualization(TDV) documentation set:

- *TIBCO TDV and Business Directory Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.
- TDV Installation and Upgrade Guide
- TDV Administration Guide
- TDV Reference Guide
- TDV User Guide
- TDV Security Features Guide
- Business Directory Guide
- TDV Application Programming Interface Guide
- TDV Tutorial Guide
- TDV Extensibility Guide
- TDV Getting Started Guide
- TDV Client Interfaces Guide
- TDV Adapter Guide
- TDV Discovery Guide
- TDV Active Cluster Guide
- TDV Monitor Guide
- TDV Northbay Example

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website mainly in the HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit https://docs.tibco.com.

Documentation for TIBCO Data Virtualization is available on https://docs.tibco.com/products/tibco-data-virtualization-server.

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit https://www.tibco.com/services/support.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at https://support.tibco.com.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to https://support.tibco.com. If you do not have a user name, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to https://community.tibco.com.

Introduction

This topic presents an overview of the data source extensibility features and how to use them:

- What Is the TDV Data Source Toolkit?, page 5
- Workflow, page 5
- Basic Concepts, page 5

What Is the TDV Data Source Toolkit?

The TDV Data Source Toolkit is a framework and a set of features that let you create, deploy and configure extension adapters. You can then use these adapters to connect to and work with data sources that TDV, as shipped, does not support.

Workflow

The TDV Data Source Toolkit and this document together support the steps needed to customize an adapter and data source:

- How to Prepare a Data Source Extension Adapter, page 7, describes how to build your own adapter from scratch or by modifying an existing sample adapter.
- Adapter Package Deployment, page 15, explains how to deploy your extension adapter JAR file using a command-line utility.
- Extension Adapter Configuration, page 19, explains how to customize the adapter and determine what is visible in the user interface for data sources that use the adapter.
- Data Source Configuration, page 31, describes how to create and configure a data source instance that uses the extension adapter.

Basic Concepts

To use TDV extensibility effectively, it is important to know its basic concepts and terminology.

- *Adapter* refers to the connective software for a particular brand of data source (such as Oracle or PostgreSQL). An adapter can be configured to use a certain set of properties from the collection available for the adapter. The adapter can be configured to make some or all of these properties visible, required, and overrideable in the configuration window for each data source instance.

  Adapters support relational data sources, and data sources whose intended JDBC driver allows applications to see them as relational, such as Hive via the Hive JDBC driver.

  Extension adapters cannot inherit configurations from other extension adapters.

- A *data source instance* is a physical data source to which TDV connects. The instance is identified by its name and connection URL (for example, ds-ora-1:11521/dev1).

- *Data source properties* are defined by the adapter. Their values can be specified at the adapter level, or at the individual data source level, or both.

  Data source properties fall into two groups:

  — Properties that are characteristic of the kind of data source, such as data types, function support, and support for various language elements.

  — Properties of individual data source instances, such as name, connection URL, login credentials, and settings that may differ from one instance to the next, such as case sensitivity and connection pool settings.

How to Prepare a Data Source Extension Adapter

TDV comes with a set of built-in adapters that connect TDV with various data sources.

The Data Source Toolkit lets you develop, deploy, and manage data source *extension* adapters for TDV—self-contained packages that can be deployed as add-on adapters.

You need to have a general understanding of how TDV works to write an extension adapter. This chapter documents the steps required to develop and manage the life cycle of an extension adapter.

- How to Write an Extension Adapter, page 7
- Directory Contents and Build Steps, page 12

How to Write an Extension Adapter

Following a discussion of TDV and Adapters, page 7 and the TDV Extension API, page 8, this section tells you how to code the extension adapter to:

- Retrieve Adapter Configuration, page 10
- Customize Connection Handling, page 10
- Customize Metadata Retrieval (Introspection), page 10
- Customize Retrieval of Results, page 12

TDV and Adapters

TDV connects to data sources using an adapter. More than two dozen adapters are built into TDV. To see a list of these adapters, you can right-click on an appropriate folder in the Studio resource tree—for example, My Home—and select New Data Source. The New Physical Data Source window appears, with a list of available adapters.

If the adapter you need is not listed in the New Physical Data Source window, or if you need a custom version of an adapter in the list, the Data Source Toolkit provides you the tools to create that adapter. The adapter can then be built, packaged and deployed as an add-on to TDV's collection of adapters. After successful deployment, it appears in the list in the New Physical Data Source window, and is available in the <host_name>/packages/<package_name> folder of the Studio resource tree so you can open and customize it.

The Data Source Toolkit includes Java APIs and default implementations for most aspects of the adapter components. Where required, an alternate implementation can be provided to customize behavior.

At a minimum, an adapter needs to provide a main Java class with DSAdapterProvider and CisExtensionProvider annotations.

The following can be customized using the API:

- Retrieve adapter configuration settings
- Connect to a data source
- Introspect the data source to retrieve its metadata
- Get data from the data source

TDV Extension API

The TDV Extension API can be found at:
`<TDV_install_dir>/apps/extension/docs/com/compositesw/extension/ds`

The four packages in this API are:

- Main Data Source Package, page 8
- Configuration Package, page 9
- Implementation Package, page 9
- Introspection Package, page 10

To view Javadoc descriptions of TDV Extension API components, start with the index file:
`<TDV_install_dir>/apps/extension/docs/index.html`

Main Data Source Package

The main data source package is `com.compositesw.extension.ds`. It contains:

- DataSourceContext—A utility class to interact with TDV.

- DataSourceFactory—A factory for specifying customized implementation classes.
- Logger—An interface for adapter providers to produce debug, error, info, and warning log messages.
- Poolable—An interface to activate and deactivate poolable objects; add, remove, and retrieve a list of registered pool listeners; and set the data source context object.
- PoolListener—A listener that gets callbacks when an object is in and out of the pool.
- RelationalConnectionFactory—A factory for connections to the physical data source.
- RelationalConnectionListener—A connection listener interface to get callbacks upon creation or destruction of a connection.
- RelationalDataSourceFactory—A factory for specifying customized implementation classes for relational adapters.
- RelationalShunt—A shunt interface to customize the way values are retrieved from the result set.
- RelationalShuntFactory—An interface to create custom relational shunts.
- StatementListener—Reserved for future use.

Configuration Package

The configuration package is `com.compositesw.extension.ds.conf`. It contains:

- AdapterConfig—Interface for a data source adapter configuration; can be used to return the current value of adapter properties.
- DataSourceConfig—Interface for a data source configuration; can be used to return the current value of data source properties (encrypted or plain).

Implementation Package

The implementation package is `com.compositesw.extension.ds.impl`. It contains:

- AbstractRelationalMetaData—Contains an abstract relational metadata implementation class, a base implementation that inherits dozens of fields from the java.sql.DatabaseMetaData interface.

Introspection Package

The introspection package is `com.compositesw.extension.ds.introspect`. It contains:

- IntrospectionFilter—An interface that can filter out unneeded metadata items.
- RelationalMetaData—An interface for customizing metadata introspection. This interface is extended from DatabaseMetaData. DatabaseMetaData contains comprehensive information about the database as a whole.
- ResultSetToMetadataConverter—An interface to extract metadata information from the result set of a custom SQL query.

Retrieve Adapter Configuration

AdapterConfig and DataSourceConfig interfaces let you retrieve the properties and values set in the adapter configuration and data source configuration. (See How to Prepare a Data Source Extension Adapter, page 7 and Data Source Configuration, page 31.)

Customize Connection Handling

After setup and configuration, the first task required of an adapter is to instantiate a data source and connect to it. If the default behavior is not sufficient for some reason, an alternate implementation can be provided.

Customize Metadata Retrieval (Introspection)

The next task required of an adapter is to introspect the data source to retrieve its metadata. Set the boolean properties (Adapter Introspection Properties, page 47) according to the metadata available in the data source.

Once the introspection is completed, TDV can query the exposed metadata.

The default implementation uses JDBC DatabaseMetadata to retrieve the metadata. If that does not suffice or perform well, an alternate implementation can be provided.

For relational adapters, the standard interface for the introspector is RelationalMetaData. RelationalMetaData extends DatabaseMetadata. A custom implementation can extend AbstractRelationalMetadata to reduce the burden of implementing all of the DatabaseMetadata methods.

You can use ResultSetToMetadataConverter to translate the metadata result set into a form that the toolkit introspection API can consume.

You can use the IntrospectionFilter interface to filter out unneeded metadata items.

Note: It is important to make sure your adapter retrieves precision, scale, and length from data sources that use nonstandard names for columns in their metadata. See Determination of Precision, Scale, and Length, page 11.

Determination of Precision, Scale, and Length

Adapters use Native-to-TDV data type mappings set on the adapter Configuration tab to determine how to map from column types encountered during introspection to generic TDV data types.

When you introspect a table, you might see that the data type of a table column has unexpected attribute values, such as incorrect precision or scale or length. If this happens, the adapter might be using the wrong field in your database metadata to fetch information about those attribute values.

For example, the default implementation obtains precision from metadata this way:

```
resultSet.getInt("COLUMN_SIZE")
```

Although this works for many data sources, your data source may be different. For instance, your custom data source may return precision in a field called NUMERIC_PRETDVION rather than COLUMN_SIZE. In this case you would need to retrieve it from the result set as rs.getInt(“NUMERIC_PRETDVION”).

You can solve this in one of several ways.

First Option

In Adapter configuration, under Introspection properties, find the property that specifies which field to use for precision, scale, or whatever you are trying to fix. For example, for precision the property is:

Column name for column size in java.sql.DatabaseMetaData.getColumns()

You can change the value of the property to use the appropriate field name for your data source. Also, you can use the Overrides tab in Data Source UI to override this behavior for individual data source instances.

When you change the property to specify a different field name, make sure that the metadata result set actually contains that field. If it does not, use second option and write a SQL query to fetch precision, scale, and length instead of getting it from result set metadata. For example, you may know that a system table contains precision, scale, and length in this field, but the field is not returned as part of metadata.

Second Option

In your adapter extension implementation, you can extend the class AbstractRelationalMetaData and override method getColumns(), where you would place the custom logic that finds and sets precision, scale, length, or other data type details, including the data type name itself.

This is the preferred option if you want to be able to set type details in different ways depending on other conditions.

Third Option

If you want to always map a specific native type to a certain fixed precision/scale/length, regardless of what comes from the database metadata, you can use Data Type Mappings in the adapter configuration.

For example, you can map native type MONEY to a TDV type DECIMAL(19, 2). However, to make TDV honor the numbers in brackets after the type in Native-to-TDV type mappings, you must go to the adapter configuration and set this Introspection property to True:

> Data type details in type mappings override result set implementation

Also, you can use the Overrides tab in Data Source UI to override this behavior for individual data source instances.

Customize Retrieval of Results

The next task required of an adapter is to retrieve query results from the data source.

You can use RelationalShunt, or create a custom shunt with RelationalShuntFactory, to customize the way values are retrieved from the result set.

Directory Contents and Build Steps

TDV 7.0 is shipped with two sample extension adapters, which you can copy and modify as one way to create your own adapter. Each sample adapter comes with a Readme.txt file that describes the following:

- The software required to build and test extension adapters.
- The directory structure and content of the sample adapter source, which illustrate the organization and files needed to create your extension adapter package.

- The steps to build (compile and create) your extension adapter package.

The two sample extension adapters and their readme files can be found at:

- <TDV_install_dir>/apps/extension/examples/adapters/postgres-example-adapter-<version>
- <TDV_install_dir>/apps/extension/examples/adapters/redshift-example-adapter-<version>

For example, for TDV version 7.0.2 the sample Postgres adapter file name is postgres-example-adapter-**7.0.2**.

Adapter Package Deployment

This chapter describes how to deploy and undeploy an extension adapter package, and troubleshoot these procedures.

Once the adapter is loaded, the adapter name appears in its alphabetical place in the list in the New Physical Data Source window. Select it and configure a data source in the same way as any standard data source.

Deploying the Package

You can deploy the extension adapter package from a command line.

To deploy the extension adapter package from a command line

1. Open a command window.
2. Go to the bin directory:

```
cd <TDV_install_dir>/bin
```

3. Run the server utility (server_util.bat on Windows; server_util.sh on UNIX/Linux) with the deploy subcommand.

   For example:

```
./server_util.bat -server <hostname> [ -port <port> ] [ -encrypt ]
  -user <username> -password <password> [ -domain <domain> ]
  -deploy -package my_adapter.jar [-verbose]
```

   If deployment is not successful, see Troubleshoot Package Deployment, page 16.

For a full description of server_util syntax and its options, refer to the "TDV Command-Line Utilities" chapter of the *TDV Administration Guide*.

Troubleshoot Package Deployment

You can check to see if deployment was successful in various ways:

- Use the -verbose option and look for a confirmation message.
- You can select New Data Source from a Studio menu and scroll to the new extension adapter in its alphabetical place in the list of physical data sources.
- You can click Refresh All In the Studio icon bar and look for the adapter under the <host_name>/packages/<package_name> folder in the resource tree.

Validation of Configuration at Deployment

Deployment validation can have various outcomes:

- Failure:
    - If the contents of the JAR do not conform to the expected structure, or if the YAML document representing the adapter configuration is malformed, deployment of the extension adapter package is aborted.
    - If validation of the adapter configuration file produces errors, deployment of the extension adapter package is aborted. See Validation of Configuration When Adapter Configuration Is Saved, page 20.
    - The package duplicates a package of the same type and version as one that has already been deployed.
    - The package is not compatible with the current version of the server.
- Success with warnings:
    - If validation produces only warnings, the deployment succeeds, and the validation warning messages are written to the server log.
- Success

Resource Tree Structure of a Deployed Package

After the package has been deployed, it is placed in the Studio resource tree under <host_name>/packages. The package occupies four levels:

- Metapackage:
  <host_name>/packages/<package_name>
- Package upload (package version):
  <host_name>/packages/<package_name>/<package_name>:1

- Adapter extension package:
  <host_name>/packages/<package_name>/<package_name>:1/<cis_extension_provider_name>
- Adapter:
  /<host_name>/packages/<package_name>/<package_name>:1/<cis_extension_provider_name>/
  <ds_adapter_provider_name>

The names are derived as follows:

- <package_name> comes from MANIFEST.MF entry cisext-name (for example, cisext-name: PostgresAdapterExtension)
- <cis_extension_provider_name> comes from the name specified in the CisExtensionProvider annotation (example @CisExtensionProvider(name=”PostgresAdapterExtension”))
- <ds_adapter_provider_name> comes from the adapters specified in the DSAdapterProvider annotation (for example, @DSAdapterProvider(adapters={"shortName=postgresExtension:longName=Postgres Extension Adapter”})

Undeploying a Package

You can undeploy the extension adapter package from a command line.

To undeploy the extension adapter package from a command line

1. Open a command window.
2. Go to the bin directory:
```
cd <TDV_install_dir>/bin
```
3. Run the server utility with the undeploy subcommand.

   For example:
```
./server_util -server <hostname> [ -port <port> ] [ -encrypt ]
  -user <username> -password <password> [ -domain <domain> ]
  -undeploy -name <adapter_name> -version 1 [-verbose]
```

   Use version number 1.

   If deployment is not successful, see Troubleshoot Package Undeployment, page 18.

For a full description of server_util syntax and its options, refer to the "TDV Command-Line Utilities" chapter of the *TDV Administration Guide*.

Troubleshoot Package Undeployment

Common causes of undeployment failure that are indicated by exception messages are:

- java.lang.NumberFormatException: The version number was not specified.
- java.lang.Exception:
  - — The package was never deployed successfully.
  - — The package has already been removed.
  - — The package name is misspelled, or is not the name assigned to the installed package hierarchy by an annotation in the main class within the JAR file
  - — You specify an incorrect version number.
  - — You included the ".jar" suffix, which should be omitted when undeploying a package.

Extension Adapter Configuration

This chapter describes how to configure an extension adapter:

Overview of Extension Adapter Configuration

When you assemble a deployable package, you have a choice of what to do about the configuration. If several adapters are in the package, you can make different choices for each adapter.

Completeness of Original Configuration

What TDV does with a newly deployed adapter depends on the state of its original configuration:

- If an adapter configuration file is *not* provided for an adapter, then during package deployment the adapter is assigned a default configuration that contains the most common settings for relational data sources.

  Because TDV does not know anything about the new data source, some assigned property values may be incorrect for this new data source. Therefore, after deployment you need to open the adapter configuration in Studio and change property values systematically so they accurately describe the data source's characteristics. Also, it is crucial that you provide information about native data types.

- If an adapter configuration file is provided for an adapter, but the file does not contain the minimum set of properties and property values that TDV requires, the adapter configuration file is first validated. Upon successful validation, the adapter configuration file is merged with the collection of all required and optional properties.

  After deployment, you need to open the adapter configuration file in Studio to make sure all the properties have appropriate values for the data source at hand. Make sure the adapter configuration file specifies information about native data types.

- If an adapter configuration file contains the minimum required set of properties and property values, then upon successful validation the adapter is fully enabled and can create data source instances.

  If the adapter configuration file does not define at least ten functions, the adapter is merged with the full list of all functions known to TDV; otherwise, to save developer time, the merge is not done.

  It is still advisable to go through the adapter configuration in Studio and make sure the merged result is appropriate to the intended data source. Make sure the adapter configuration file specifies information about native data types.

Configuration Planning

When you are setting up the adapter configuration in Studio, these are the main questions to consider:

- Which properties do you want to expose in the data source UI, and which do you not want to expose? Set visibility flags accordingly.
- Which properties should be required? Set visibility flags accordingly.
- Which properties do you want to allow data sources to override? Set visibility flags accordingly.
- What are all the native types and how should they map to Composite data types? How should Composite data types map to the native data types? Set data type mappings in both directions, defining additional data types as needed.
- Which properties are relevant? Set default values for them and delete the others.
- What is unsupported in SQL clauses? Specify those data types.
- Which operators and functions are supported? Set native expressions and arguments appropriately and delete the others.
- Are additional operators or functions needed? If so, define custom operators and functions.

Validation of Configuration When Adapter Configuration Is Saved

When you save changes made on either the Configuration tab or the Text tab, a validity check is made:

- If the adapter configuration file (a YAML document) is malformed or produces errors, it is not saved.

- If you edited the adapter configuration file directly (on the Text tab), the file could be invalid because of:
  - — Schema errors in the structure of the document (for example, the wrong number of space characters before terms)
  - — Use of characters that are not allowed (for example, tab characters; use spaces)
  - — Misspellings, or uppercase where lowercase is required
  - — Unexpected elements
- If validation produces only warnings, the deployment succeeds, and the validation warning messages are written to the server log.

As part of a successful validity check, some changes may be made automatically. For example:

- If Required for data source is set to true but Display in UI is set to false, the validity checker changes Display in UI to true so that the person configuring the data source can set the value.
- If a data type prefix of "cis." or "native." is used for a valid data type but the prefix is not needed, it is removed before the data type is saved.

Extension Adapter User Interface

Once you have deployed an extension adapter package, you can configure the adapter and determine what can be configured for data source instances that use the adapter. When you configure an adapter in Studio and save it, the configuration is automatically checked for validity.

The extension adapter user interface has three main tabs (at the bottom):

- Configuration—This is the tab on which you configure the adapter and make properties available for data source configuration. See Extension Adapter Configuration Tab, page 22.
- Text—This tab lists the text equivalent of settings made and saved on the Configuration tab. If you edit any text on this tab, any unsaved changes made on the Configuration tab are discarded. The Configuration tab reappears the next time the user interface is opened.
- Info—This is the standard information tab, with the adapter name, owners, dates, state of the lock, and so on.

Extension Adapter Configuration Tab

An extension adapter manages much of the housekeeping for data sources attached to it, and determines what is available to set or change in the configuration interface for those data sources. The Configuration tab is where you make these choices.

Many aspects of the user interface operate in the same way for most or all of the configuration activities you can perform. These are explained in the introductory sections:

- Flags, page 22
- Add Any Property, page 25
- Delete Any Property, page 25

Because you configure the same fields for properties on the first several Configuration subtabs, these subtabs are grouped and described together in Configure Simple Properties, page 26.

Properties on the other tabs are configured in somewhat different ways:

- Configure Data Type Mappings, page 26
- Configure Clauses, page 28
- Configure Operators or Functions, page 28

Flags

On the first several Configuration subtabs, each property group and individual property can have three flags associated with it. The names of the flags are also listed as they appear on the Text tab.

- Data sources must specify value—(On Text tab, value_required_for_datasource) This flag specifies whether to require that individual data sources specify a value for this property. If the value is True, the property will appear in the data source UI on the Info tab and will be required.
- Data sources may override value—(On Text tab, value_overridable_by_datasource) If this flag is True, this means that the adapter sets the value of this property, but individual data sources can optionally override that value for themselves.

- Display property in data source UI—(On Text tab, display_in_ui) This flag controls whether properties can be seen in the data source UI, and on which tab:
  - — For required properties (Data sources must specify value is True) this flag will always be True, because required properties must be visible.
  - — For properties that are neither required nor overrideable by data sources, this flag will be assumed to be False, because read-only properties should not appear in the data source UI.
  - — If a property is overrideable (Data sources may override value = True) and this flag is True, the property will appear in the data source UI on the Info tab (along with required properties), but the value will not be required.
  - — If a property is overrideable and this flag is False, the property will not appear on the Info tab, but it will be available for selection in the search box on the Overrides tab of the data source UI.

The display is different for group-level flags and individual property flags. Group-level flags appear as check boxes at the top of each Configuration subtab.

Individual property flags are set from drop-down lists. Setting for these flags to Group default causes them to take their values (true/checked or false/unchecked) from the corresponding group-level flags at the top of the subtab.

Example

On the String Comparison subtab, the “String comparison is case-insensitive” property starts out with these flag values:

- Data sources must specify value—True
- Data sources may override value—Group default
- Display property in data source UI—True

The group-level flags (above the collection of properties) for the String Comparison group have these values:

- Data sources must specify value—False (unchecked)
- Data sources may override value—True (checked)
- Display property in data source UI—False (unchecked)

Because individual property flags override group-level flags, the resulting flag values for the “String comparison is case-insensitive” property are:

- Data sources must specify value—True
- Data sources may override value—True

- Display property in data source UI—True

On the Info tab for a data source instance that uses this extension adapter, the “String comparison is case-insensitive property” is available on the Basic subtab, where the user can override the adapter’s value.

Flag Combinations

The comments describe what happens for each combination of flag settings for a given property.

Data sources...			
Must specify a value	**May override the value**	**Display in UI**	**Comments**
False	False	False	Property value can only be specified by the adapter. Does not appear anywhere in data source UI.
False	False	True	Same as the previous combination; the value of the third flag is ignored. Saving an adapter configuration with this combination results in a message warning that this property will not appear in data source UI.
False	True	False	Property is available for selection from the search box on the Overrides tab of data source UI.
False	True	True	Appears on the Info tab of data source UI as an optional property.
True	False	False	Saving an adapter configuration with this combination results in a message warning that “display in UI” flag for this property will be changed to True. Property will appear on the Info tab as a required property.
True	False	True	Appears on the Info tab as a required property.
True	True	False	Saving an adapter configuration with this combination results in a message warning that "display in UI" flag for this property will be changed to True. Property will appear on the Info tab as a required property.
True	True	True	Appears on the Info tab as a required property.

Add Any Property

The Add buttons let you create custom properties, mappings, operators, and functions. Add buttons are available in several places:

- An Add custom property button appears on all tabs that contain simple properties (that is, properties that designate a single value).
- An Add new mapping button appears below each pair of mapping columns in the Data Type Mappings tab.
- An Add custom operator button appears near the top of the Operators tab.
- An Add custom function button appears near the top of the Functions tab.

The Clauses tab has no Add button.

Add a Custom Property, Operator, or Function

When you click an Add custom property button (available on the first several subtabs), a dialog box opens in which to type a short name for system use, and a more descriptive name to display in the current property tab (and in the data source UI). When you click OK, the new custom property is added to the top of the list on the current property tab, with fields appropriate to the property type. (See Configure Simple Properties, page 26.)

The same actions take place when you click the Add custom operator button on the Operators tab, or click the Add custom function button on the Functions tab. (See Configure Operators or Functions, page 28.)

Add a Data Type Mapping

When you click the Add new mapping button, a pair of fields is added to the list so you can specify new source-destination data type pairs. (See Configure Data Type Mappings, page 26.)

Delete Any Property

A Delete button is displayed next to:

- Optional properties, operators, and functions
- Argument data types
- Data type mapping pairs

The Clauses tab has no Delete button. To signify that a clause is not supported by a data source type, leave the Native Expression field empty.

Configure Data Type Mappings

This subtab lets you define the mappings in both directions between TDV and data sources that use the adapter. You can use the default mappings, change the mappings, or add new mappings.

TDV to Native Mappings

These mappings indicate what each TDV data type (on the left) becomes in the data source (on the right) when TDV issues DDL to create or modify objects in those data sources.

Native to TDV Mappings

These mappings indicate what each data source data type (on the left) becomes in TDV (on the right) when TDV introspects those data sources.

Notation

As pointed out at the top of this subtab, "When needed, use &p for precision, &s for scale, &l for length, or the actual number if known." However, except for timestamps and interval types, specify precision, scale, and length *only on the right side* of the mapping (the target data type).

Examples:

- On the target side of a mapping: NUMBER(12,2) indicates a NUMBER data type with a precision (&p; the total number of significant digits) of 12. The 2 (&s, the scale) indicates 2 digits after the decimal point and 10 digits before the decimal point.
- On the target side of a mapping: CHAR(&l) indicates a CHAR data type and its length.
- On either side of a mapping: TIMESTAMP(3).

Configure Simple Properties

Properties on the first several Configuration subtabs are collectively called "simple properties" because they have just one value. They let you specify a value data type, allowable values, and an adapter-specified value.

The fields for data type and allowed values are grayed out for properties defined by TDV because they should not be modified by the user. When you add a custom property, such fields are enabled for that custom property.

Value Data Type

When you add a custom property, you can assign it one of the following data types:

- — STRING
- — PASSWORD_STRING
- — BOOLEAN—implicit allowable values are true or false
- — DOUBLE
- — FLOAT
- — LONG
- — INTEGER
- — FILE_PATH_STRING
- — PATH_STRING
- — FOLDER

Allowed Values

Type a comma-separated list of the allowable values for this property.

Property Value

Type a value for this property. This becomes the default value for any data source instance that uses this adapter.

Some properties have a value already specified, some do not. If the group-level flag, or the flag for this property overriding it, has a value of True for Data source must specify value, the property value needs to be specified in the data source.

When a data source is created, it inherits the current value of the property as defined by the adapter. At that point the inheritance ends: if the adapter changes the value of this property later, the change is not propagated to the data source. Even if the data source sets the value to empty or NULL, the effective value for that data source will be empty or NULL.

If you specified allowable values, the value you type for this property must be one of those values. If this is not the case and you try to save the adapter configuration, an error message box appears and the configuration is not saved.

Configure Clauses

On the Clauses subtab, you can change the native expression and designate unsupported native and TDV data types for each type of SQL clause. Clauses can be marked as unsupported by clearing the native expression field for it, either in the adapter configuration or in the data source overrides.

Native Expression

Type a representative syntax for the expression as it is to be pushed to the data source. In most cases the native expression already filled in does not need to change.

Unsupported Native Types

Type a comma-separated list of native (to the data source) data types that are not supported for this clause.

Unsupported TDV Types

Type a comma-separated list of unsupported TDV data types. A column or value of such types cannot appear in this clause. For example, the data source cannot do a UNION between two CLOB values.

Configure Operators or Functions

On the Operators and Functions subtabs, you can add, remove, define and modify the expressions (signatures) and data types of functions or operators (either existing or custom) for new data source instances.

Throughout these subtabs, use the notation $1, $2, and so on, to indicate the first (arg1), second (arg2), and subsequent arguments.

Default Native Expression

Type a representative syntax for the expression as it is to be pushed to the data source. This syntax is a combination of argument notation and the characters that make up the operator or function name. For examples, examine some existing native expressions (like “$1 > $2”) in the user interface.

Add a Variation

Right below Default native expression is a field labeled Add <property_name> variation with number of arguments. Specify the number of arguments (up to 100) and click the Add button. A new argument combination (argument data types and native expression) is displayed at the top of the Argument combinations list for this property.

Argument Data Types

Type a data type, or a comma-separated list of data types, for each argument. You must use the prefix cis. or native. before each data type name to remove ambiguity. For a list all TDV data type names, see Data Type Naming, page 38.

Arguments are numbered by their position in each specific argument combination. For example, $2 in one native expression is not necessarily the same as $2 in another native expression.

Native Expression

You need to specify a new native expression if:

- An argument combination uses a different number or ordering of arguments than the default native expression.
- Behavior differs from the SQL standard, such as for a NULL argument, and so you need to map the function combination to something that would give an expected result. That result could be a literal, a CASE IF ELSE statement, or a different function altogether.

Extension Adapter Text Tab

This tab displays the configuration file that represents the most recently saved properties and settings for the extension adapter. Changes that you make on the Configuration tab do not show up on the Text tab until you save the changes and reopen the adapter UI.

This YAML file is displayed for your convenience. It is best not to edit this file directly. If you do, you risk making the extension adapter unusable (for example by using tab characters instead of spaces for indentation).

On the Text tab you can, among other things:

- Search using Ctrl-F
- Make quick edits (if you are familiar with the structure of the document)

- Copy and paste fragments or whole configurations from another source
- Insert from a file

Flag Names on Configuration and Text Tabs

The flags that control properties that can be made available for customization have different names on the Configuration tab and the Text tab (the YAML file).

The following table lists the names used on each tab.

Name on Configuration Tab	Name on Text Tab
Data sources must specify value	value_required_for_datasource
Data sources may override value	value_overridable_by_datasource
Display property in data source UI	display_in_ui

Fourth Flag (value_required_for_adapter)

On the Text tab, the YAML file listing includes a fourth flag, which is handled by the system: value_required_for_adapter. This flag does not appears on the Adapter Configuration tab, because it is not configurable.

System logic enforces the value of this flag:

- Custom property—The flag must be set to False; otherwise, the users would not be able to delete or modify the property. Validation makes sure this flag is set to False for each custom property.
- System property—If the JDBC template requires a property, validation makes sure this flag is set to True, whether for the property itself or for its category. Users cannot change or delete system properties.

Data Source Configuration

After you have deployed an extension adapter package, you can access it from Studio and use it to create and customize a new data source instance.

- Create a New Custom Data Source, page 31
- Configure a New Custom Data Source, page 31
- Configuring Extension Adapters for Maven, page 34

Create a New Custom Data Source

To create a new custom data source that uses an extension adapter, begin in the usual way by clicking New Data Source, selecting the name of the extension adapter from its alphabetical place in the New Physical Data Source window, assigning it a name, and so on. See "Working with Data Sources" of the *TDV User Guide* for details.

The set of properties available for data source instances connected using a given extension adapter depend on the configuration of that adapter. (Extension Adapter Configuration, page 19, describes this.)

Configure a New Custom Data Source

When you open a custom data source, the display on the right of the Studio window includes several tabs:

- Configuration—The contents of this tab are different from the Configuration tab for built-in data sources.
- Overrides—This tab is not present in the display for built-in data sources.
- Cardinality Statistics—The contents of this tab are the same as for a built-in data source.
- Re-Introspection—The contents of this tab are the same as for a built-in data source.
- Introspection Report—The contents of this tab are the same as for a built-in data source.

- Relationship Discovery—The contents of this tab are the same as for a built-in data source.
- Info—The contents of this tab are the same as for a built-in data source, except the tab has no Cluster Health Monitoring or Resource Capabilities section.

For a description of the tabs for built-in data sources, refer to the three Configuring Data Sources chapters in the *TDV User Guide*.

Info Tab

The Info tab is where you specify the properties for which the adapter requires values from the data source, such as connection information. It has two subtabs, Basic and Advanced.

This is sometimes called the *main data source UI* or simply the *data source UI*.

Overrides Tab

The Overrides tab appears whenever an adapter is enabled.

NOTE: Under certain rare circumstances a property might display different values on the Info and Overrides tabs. The value displayed on the Overrides tab is the one that will be used.

In the Search for properties to override section, you can search in one of two ways:

- Begin typing the name of a property. Once you have typed at least two characters, a drop-down list displays all properties with names that contain the string you typed.
- Type * (an asterisk) for the full list of available properties.

When you see the property you want to override, you can select it from the drop-down list; it then appears on the subtab for the group to which it belongs, and disappears from the main list.

After adding a property, you can redisplay the list of properties still available by again placing the cursor in the search field next to the asterisk.

The Overrides tab contains several subtabs, each with content that varies with the way the extension adapter was configured:

- Simple Properties Tabs, page 33
- Data Type Mappings Tab, page 33
- Clauses Tab, page 33
- Operators Tab, page 33

- Functions Tab, page 34

Adapter configuration defines which properties are available for selection on the Overrides tab, as explained in these five sections.

Simple Properties Tabs

The simple properties tabs lets you set properties that involve simple yes-no decisions, or single values, or comma-separated lists.

The properties available on these tabs are those for which adapter configuration has these settings:

- overridable_by_datasource is True
- display_in_ui is False

For a description of adapter-side configuration of simple properties, see Configure Simple Properties, page 26.

Data Type Mappings Tab

The Data Type Mappings tab has two lists of mappings:

- From TDV to native data types that are to be used for DDL generation in the data source. The list of TDV data types is grayed out because you cannot change them when configuring a data source.
- From native to TDV for introspection of the data source.

For a description of the what you can do with data type mappings, refer to Configure Data Type Mappings, page 26.

Clauses Tab

The Clauses tab lets you set properties of clauses.

The clauses available on this tab are determined by the configuration of the extension adapter that connects this data source instance.

For a description of the values and syntax of custom properties for clauses, refer to Configure Clauses, page 28.

Operators Tab

The Operators tab lets you customize operators.

The operators available on this tab are determined by the configuration of the extension adapter that connects this data source instance.

For a description of the values and syntax of custom properties for operators, refer to Configure Operators or Functions, page 28.

Functions Tab

The Functions tab lets you set properties of functions.

The functions available on this tab are determined by the configuration of the extension adapter that connects this data source instance.

For a description of the values and syntax of custom properties for functions, refer to Configure Operators or Functions, page 28.

Configuring Extension Adapters for Maven

For Maven to compile successfully, you need to modify an XML file and Maven command arguments to match the release and patch level in which the extension adapter is being created.

The example shown with the mvn command below is for 7.0.2. In this scenario, a file named csext-0104.jar is present under <TDV_install_dir>\apps\extension\lib.

To configure extension adapters for Maven

1. Go to this directory:
```
\apps\extension\examples\adapters\postgres-example-adapter-<versio
n>\
```

2. If necessary, replace 0.0.0-SNAPSHOT with the TDV version number (for example, 7.0.2) in three places in the pom.xml file file, as highlighted in bold below:
```
<?xml version="1.0" encoding="UTF-8"?>
...
- <project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://maven.apache.org/POM/4.0.0">
   <modelVersion>4.0.0</modelVersion>
   <name>Modules/Server - Postgres Extension Adapter
Example</name>
   <artifactId>postgres-example-adapter-7.0.2</artifactId>
   <groupId>com.cisco.cdvs</groupId>
   <version>7.0.2-SNAPSHOT</version>
```

```
 - <build>
      . . .

      . . .
   </build>
 - <dependencies>
    - <dependency>
         <groupId>com.cisco.cdvs</groupId>
         <artifactId>server-extension</artifactId>
         <version>7.0.2-SNAPSHOT</version>
      </dependency>
   </dependencies>
</project>
```

3. Run the following command to install the jar file and create the adapter package:

```
mvn install:install-file -Dfile=../../../lib/csext-0104.jar
-DgroupId=com.cisco.cdvs -DartifactId=server-extension
-Dversion=7.0.1-SNAPSHOT -Dpackaging=jar
mvn clean package
```

Configuration Files

TDV defines configurations of its built-in adapters in multiple places:

- Properties, defined in code (for example, connection URL, login, password, and so on)
- Capabilities (supports DDL, WITH clause, and so on; data type mappings; supported functions)
- Two XML files: config_defs.xml and values.xml

TDV 7.0 continues to define its built-in adapters in the same way, but it defines *new* extension adapters in a single YAML configuration file.

The following topics are discussed:

- YAML File Organization, page 37
- Data Type Naming, page 38

YAML File Organization

Adapter configuration is structured as a YAML document that goes into a deployable package.

Items to be aware of when creating or modifying the YAML document are:

- Indentation is meaningful. The number of leading spaces in a line determines the nesting level of each element within the document structure.
- Tab characters are not allowed in a YAML file. If your editor does not convert tabs into spaces automatically, do not to use tabs.

A prototype YAML file contains:

- YAML version
- Adapter name
- Property categories, such as connection, clause support, operator support, function support, and data type mappings.

Data Type Naming

This topic discusses data types in configurations for extension adapters and data sources.

TDV Types

For extension adapters, TDV supertypes are named in such a way that no leaf type and supertype have the same name, even when prefix characters like ~ and @ have been removed. For example, supertype ~date is now referred to as dates, to distinguish it from the leaf type date (formerly denoted @date). The new type names appear in YAML configuration files and in Studio.

The table shows the source hierarchy of supertypes (in bold) and leaf types. The right column lists the names that are used for extension adapters.

Source (capabilities files for built-in adapters)				Target (YAML files)
`~any`				`any`
	`~number`			`numbers`
		`~whole_number`		`integers`
			`@bit`	`bit`
			`@integer`	`integer`
			`@smallint`	`smalling`
			`@tinyint`	`tinyint`
			`@bigint`	`bigint`
		`~floating_point`		`floats`
			`@float`	`float`
			`@real`	`real`
			`@double`	`double`
			`@decimal`	`decimal`
			`@numeric`	`numeric`

Source (capabilities files for built-in adapters)		Target (YAML files)
`~string`		`strings`
	`@char`	`char`
	`@varchar`	`varchar`
	`@longvarchar`	`longvarchar`
`~binary`		`binaries`
	`@binary`	`binary`
	`@varbinary`	`varbinary`
`~date`		`dates`
	`@date`	`date`
	`@time`	`time`
	`@timestamp`	`timestamp`
`~lob`		`lobs`
	`@clob`	`clob`
	`@blob`	`blob`
`@boolean`		`boolean`
`@null`		`null`
`~interval_year`		`year_intervals`
	`@interval_year_to_month`	`interval_year_to_month`
	`@interval_year`	`interval_year`
	`@interval_month`	`interval_month`
`~interval_day`		`day_intervals`
	`@interval_day`	`interval_day`

Source (capabilities files for built-in adapters)	Target (YAML files)
@interval_hour	interval_hour
@interval_minute	interval_minute
@interval_second	interval_second
@interval_day_to_hour	interval_day_to_hour
@interval_day_to_minute	interval_day_to_minute
@interval_day_to_second	interval_day_to_second
@interval_hour_to_minute	interval_hour_to_minute
@interval_hour_to_second	interval_hour_to_second
@interval_minute_to_second	interval_minute_to_second
@xml	xml

All Types

In extension adapter configurations and data source overrides, TDV and native data type names appear in these areas:

- Data type mappings
- Clause support
- Operator mappings
- Function mappings

Native and TDV types often have the same name—for example, TDV's date and Oracle's date. In capabilities files for built-in adapters, the two are differentiated using prefixes. A TDV supertype name would start with ~ (for example, ~date), a TDV leaf type would begin with @
(for example, @date) and a native type would have no prefix (for example, Oracle's native type called date).

For extension adapters, the @ and ~ prefixes are no longer used. Whole-word prefixes, followed by a dot, now distinguish between native and TDV types:

- cis.date—TDV leaf type
- cis.dates—TDV super type
- native.date—Oracle’s native type

It is recommended that you use the prefixes only in those parts of the configuration file where both TDV and native types can appear; that is, function and operator mappings. For instance, the same function can list both native and TDV types in the same argument combination.

Because data type mappings and clause support specifically request either TDV or native types, no ambiguity can occur and so no prefixes are required:

- The data type mapping configuration section specifies TDV-to-native mappings and native-to-TDV mappings separately.
- A configuration item describing a particular clause specifies unsupported native types and TDV native types separately.

For these sections, prefixes are unnecessary and discouraged for the sake of brevity.

Here are examples of how to refer to types for data type mappings and clauses:

- date—TDV leaf type
- dates—TDV supertype
- date—Oracle's native type

Metadata Introspection Method Properties

DatabaseMetaData interface methods let you retrieve comprehensive information about the database. Interface methods like getColumns return a result set; values of capabilities like "introspect.column.column_name" represent fields in that result set; and one would do resultSet.getString(capability_value) to retrieve the value of a field.

The field value can be retrieved as a STRING argument, a field number, or a field position to get the same information.

Method	Result Set Field	Description
getCatalogs	introspect.catalog.table_cat	The catalog name of the catalog table
getColumns	introspect.column.column_def	The column default
	introspect.column.column_name	The column name
	introspect.column.column_size	The column size
	introspect.column.data_type	The data type of the column
	introspect.column.decimal_digits	Decimal digits of the column
	introspect.column.is_nullable	The is_nullable flag of the column
	introspect.column.nullable	The nullable flag of the column
	introspect.column.ordinal_position	The ordinal position of the column
	introspect.column.table_cat	The catalog name of the column table
	introspect.column.table_name	The table name of the column table
	introspect.column.table_schem	The schema name of the column table
	introspect.column.type_name	The type name of the column

Method	Result Set Field	Description
getImportedKeys	introspect.fkey.deferrability	The deferability flag for the foreign key
	introspect.fkey.delete_rule	The delete rule for the foreign key
	introspect.fkey.fk_name	The foreign key name
	introspect.fkey.fkcolumn_name	The foreign key column name
	introspect.fkey.fktable_cat	The catalog name for the foreign key table
	introspect.fkey.fktable_name	The foreign key table name
	introspect.fkey.fktable_schem	The schema name for the foreign key table
	introspect.fkey.key_seq	The foreign-key's key sequence
	introspect.fkey.pk_name	The primary key name for the foreign key
	introspect.fkey.pkcolumn_name	The primary key column name for the foreign key
	introspect.fkey.pktable_cat	Foreign key's primary key table's catalog name
	introspect.fkey.pktable_name	The primary key table name for the foreign key
	introspect.fkey.pktable_schem	The primary key table for the foreign key
	introspect.fkey.update_rule	The update rule for the foreign key

Method	Result Set Field	Description
getIndexInfo	introspect.index.asc_or_desc	Ascending or descending (asc_or_desc) index
	introspect.index.cardinality	The cardinality of the index
	introspect.index.column_name	The column name of the index
	introspect.index.index_name	The name of the index
	introspect.index.non_unique	The non-unique flag of the index
	introspect.index.ordinal_position	The ordinal position of the index
	introspect.index.table_cat	The catalog name of the index table
	introspect.index.table_name	The table name of the index table
	introspect.index.table_schem	The schema name of the index table
	introspect.index.type	The type of the index
getPrimaryKeys	introspect.pkey.column_name	The column name for the primary key
	introspect.pkey.key_seq	The key sequence for the primary key
	introspect.pkey.pk_name	The primary key’s name
	introspect.pkey.table_cat	The catalog name for the primary key table
	introspect.pkey.table_name	The table name for the primary key table
	introspect.pkey.table_schem	The schema name for the primary key table

Method	Result Set Field	Description
getProcedureColumns	introspect.parameter.column_name	The column name of the parameter
	introspect.parameter.column_type	The column type of the parameter
	introspect.parameter.data_type	The data type of the parameter
	introspect.parameter.length	The length of the parameter
	introspect.parameter.nullable	The nullable flag for the parameter
	introspect.parameter.precision	The precision of the parameter
	introspect.parameter.procedure_cat	The catalog name of the parameter's procedure
	introspect.parameter.procedure_name	The parameter's procedure name
	introspect.parameter.procedure_schem	The schema name of the parameter's procedure
	introspect.parameter.scale	The scale of the parameter
	introspect.parameter.type_name	The type name of the parameter
getProcedures	introspect.procedure.procedure_cat	The catalog name of the procedure
	introspect.procedure.procedure_name	The procedure name of the procedure
	introspect.procedure.procedure_schem	The schema name of the procedure
getSchemas	introspect.schema.table_cat	The catalog name of the schema's table
	introspect.schema.table_schem	The schema name of the schema's table
getTables	introspect.table.table_cat	The catalog name of the table's table
	introspect.table.table_name	The table name of the table's table
	introspect.table.table_schem	The schema name of the table's table
	introspect.table.table_type	The table type of the table's table

Adapter Introspection Properties

To guide the TDV introspection process, set the following BOOLEAN properties to match the structure of your custom data source.

Property	Description
introspect.relational.ignore_catalogs	Ignores catalogs even if the metadata supports them.
introspect.relational.include_catalog	Supports catalogs
introspect.relational.include_procedure	Supports procedures
introspect.relational.include_schema	Supports schemas
introspect.relational.include_table	Supports tables
introspect.relational.include_table.foreign_keys	Supports foreign keys in tables
introspect.relational.include_table.foreign_keys.exported	Supports exported foreign keys in tables
introspect.relational.include_table.foreign_keys.imported	Supports imported foreign keys in tables
introspect.relational.include_table.indexes	Supports indexes in tables
introspect.relational.include_table.primary_keys	Supports primary keys in tables
introspect.relational.use_catalogs_for_schemas	Tables and procedures are directly under catalogs