TIBCO EBX® Data Model and Data Visualization Add-on

1.4.12
March 2022

TIBCO®

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

ANY SOFTWARE ITEM IDENTIFIED AS THIRD PARTY LIBRARY IS AVAILABLE UNDER SEPARATE SOFTWARE LICENSE TERMS AND IS NOT PART OF A TIBCO PRODUCT. AS SUCH, THESE SOFTWARE ITEMS ARE NOT COVERED BY THE TERMS OF YOUR AGREEMENT WITH TIBCO, INCLUDING ANY TERMS CONCERNING SUPPORT, MAINTENANCE, WARRANTIES, AND INDEMNITIES. DOWNLOAD AND USE OF THESE ITEMS IS SOLELY AT YOUR OWN DISCRETION AND SUBJECT TO THE LICENSE TERMS APPLICABLE TO THEM. BY PROCEEDING TO DOWNLOAD, INSTALL OR USE ANY OF THESE ITEMS, YOU ACKNOWLEDGE THE FOREGOING DISTINCTIONS BETWEEN THESE ITEMS AND TIBCO PRODUCTS.

TIBCO and TIBCO EBX are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. Please see the readme.txt file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (*https://www.tibco.com/patents*) for details.

Table of contents

User Guide

Reference Guide

Developer Guide

Customizing graph nodes

Java API Reference

Release Notes

User Guide

CHAPTER 1

About the add-on

This chapter contains the following topics:

1.1 Model and data visualization

Effective communication and understanding are essential to making the best use of your data. One tried and true communication method is the use of visual aids. To facilitate this type of communication, the TIBCO EBX® Data Model and Data Visualization Add-on generates interactive graphs of your data, relationships, and data structure.

The add-on allows you to:

- View *data structure* by rendering data model graphs.
- View *data values and relationships* by rendering graphs that show selected values and their relationships.

Data Model Graphs

By default, no configuration is required to generate data model graphs. If you have access to a data model, dataset, or table in TIBCO EBX®, you can use the add-on to render a model. Of course, options are available to customize appearance and behavior. Saved graphs can highlight changes to models. This is a convenient way for you to visualize how changes will impact the model.

Data model graphs are not limited to EBX® data sources. Using the API, graphs can be generated from external sources. See Creating and editing graph configurations [p 41] for more information on customization.

Data value graphs

The add-on does not require configuration to display data value and relationship graphs. However, administrators can create custom data value graph configurations. The default data value graphs display values in boxes and use lines to represent relationships between values. When an administrator creates a custom configuration, they can specify:

- The relationships included in a graph and how they display—as lines, or as containers. When set to lines, the add-on draws arrows to connect related values and indicate relationship direction. Container style relationships display child nodes inside of larger parent nodes.

- The tables included in a graph configuration. This determines the availability of data values and relationships for a given graph.
- Limit the availability of graph rendering to specific datasets.
- Use Java classes to filter out graph components and create custom node templates.

1.2 How this guide is organized

This guide contains the following sections:

Section	Contents
Getting started tutorial	The Getting Started Tutorial [p 14] leverages a basic use case to demonstrate product capabilities and setup options. The tutorial should take approximately 20 minutes to complete.
Data model graphs	The topics in this section cover: Generating and loading data model graphs [p 28], Interacting with data model graphs [p 33], and Changing display and configuration options [p 41].
Data value graphs	The topics in this section cover: Configuring graphs [p 49], Generating data value graphs [p 65], and Cloning and deleting graph configurations [p 63].
Appendix	The appendix contains sample data for use with the tutorial.

Getting started tutorial

CHAPTER 2

Introduction

This chapter contains the following topics:

2.1 Overview

This sample tutorial will help you get acquainted with the add-on by demonstrating one use case—product traceability. You can complete the tutorial in approximately 20 minutes. The Appendix [p 79] in this user guide contains the required data model and data to follow along. Prerequisites and setup [p 15] describes how to set-up your environment before starting.

2.2 Use case overview

Imagine that a retail store starts to receive an above average number of computer returns. Technicians traced the problem to a manufacturing defect that causes processors to overheat. To avoid liability, the company needs to track down the source of the problem and share the findings with management. The add-on can help them view and understand the data model. Then they will be able to create a graph to follow the relationships between data values to determine which:

- assembly plant(s) produced the impacted machines.
- factory sent faulty parts to the plants.

2.3 What's next?

Proceed to the Prerequisites and setup [p 15] for this tutorial.

CHAPTER 3

Prerequisites and setup

This chapter contains the following topics:

3.1 Overview

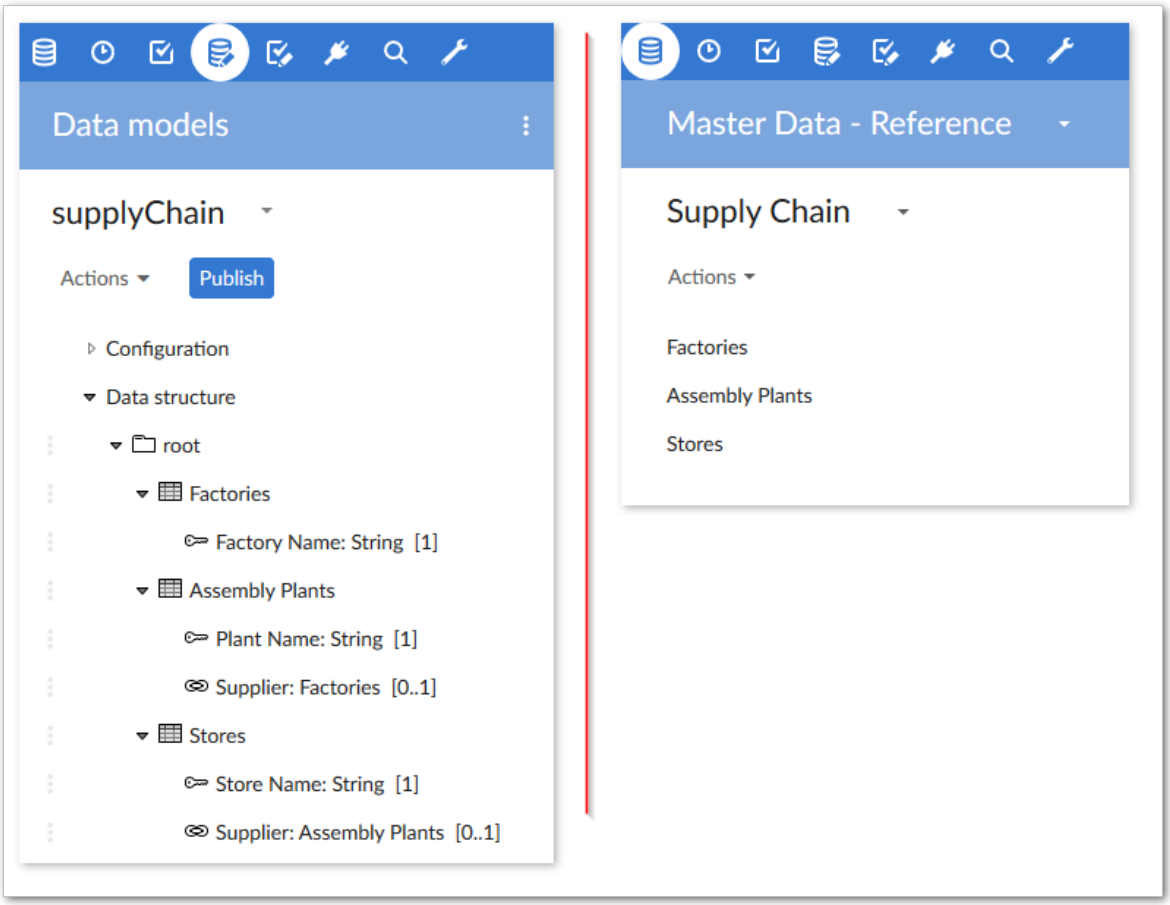
To follow this tutorial you should know how to use a text editor, be familiar with the EBX® Data Modeler Assistant (DMA), and know how to import data into EBX® tables. You can refer to the *EBX® Documentation* for additional information.

3.2 Steps

To setup your environment:

1. Copy the data model code from the Data model [p 79] section in the appendix to a file and save as XSD format.
2. The Table data [p 81] section in the appendix contains the table data required for the tutorial. Copy the data for each table to separate files and save them in XML format using the supplied table name.
3. In EBX®, create a new semantic data model and name it **`supplyChain`**.
4. From the data model's **Actions** menu, select **Import XSD** and follow the wizard to import the data model file you created in step 1.
5. Publish the data model and create a dataset called **`Supply Chain`**.
6. Open the dataset and use each table's **Actions** menu to import the data from the corresponding XML files created in step 2. You can use the Data Exchange Add-on, or the EBX® import XML option.

Your data model and dataset should resemble the following:



3.3 What's next?

After finishing the setup, continue following the tutorial in the Viewing the data model structure [p 17] section.

CHAPTER 4

Viewing the data model structure

This chapter contains the following topics:

4.1 Overview

We can begin working through the use case by generating a data model graph to get familiar with the data structure. This graph will also provide information needed during later configuration steps.

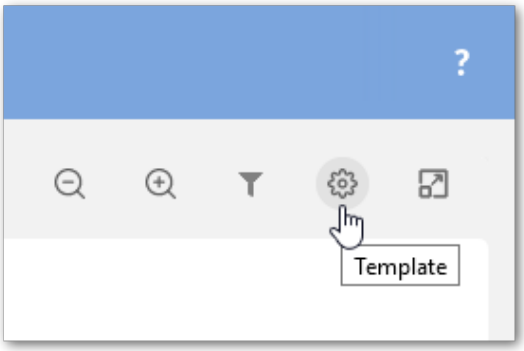
> **Note**
>
> The add-on requires no configuration to generate a data model graph. However, you can customize several options. See Changing display and configuration options [p 41] for more information on customization. The steps below cover some data model graph features and functionality. See Interacting with data model graphs [p 33] for descriptions of additional options.

4.2 Steps

To view the data model graph:

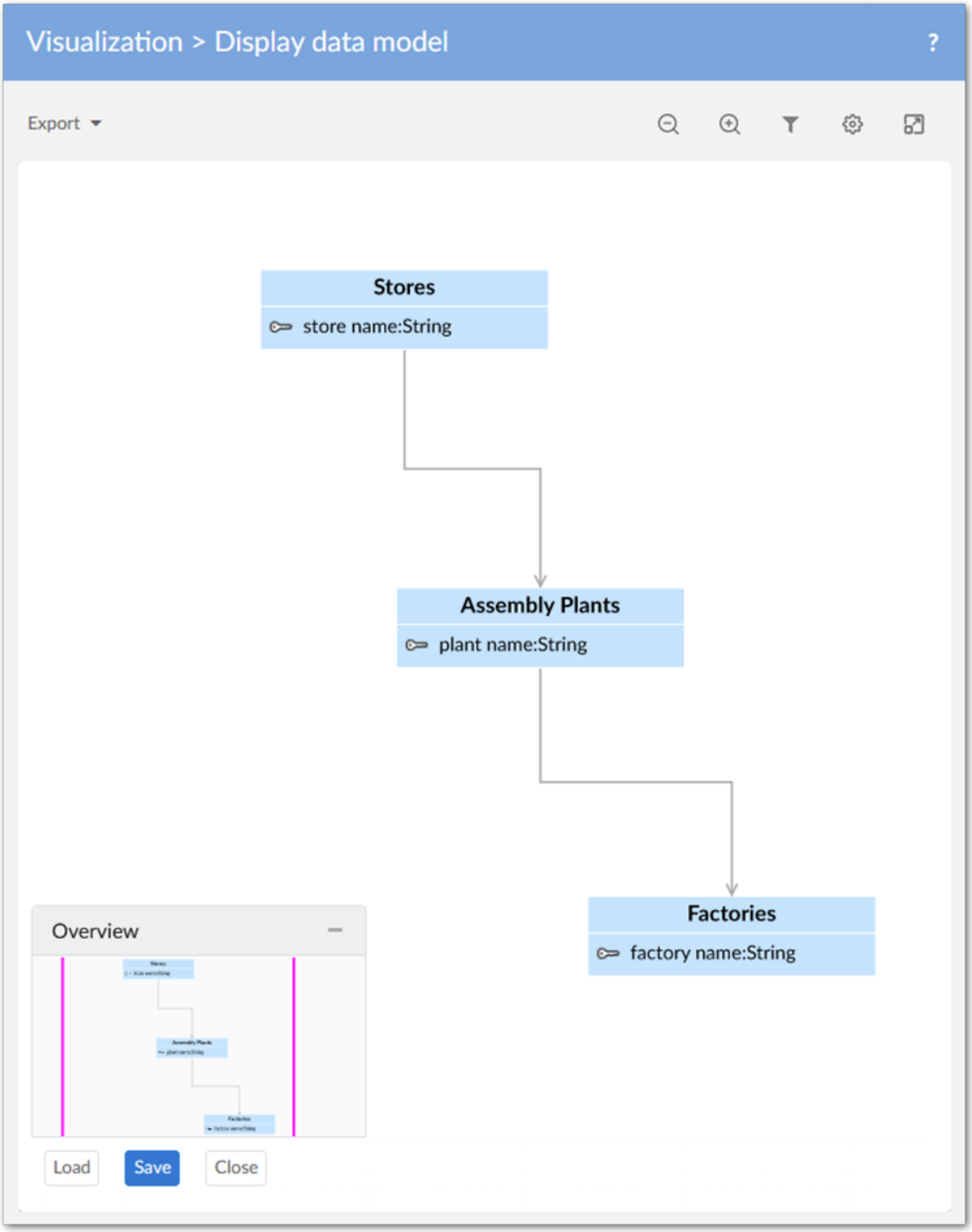
1. Navigate to the Supply Chain dataset that we created during set-up.
2. We can generate a graph using the **Display data model** service. This service is accessible from a dataset or table **Actions** menu under **Visualization** services. Feel free to run the service from both to observe the differences.
3. When viewing the graph:

- Select the template settings icon and in the **Display options** group tick the **Select all** box (or just a few options), save and close. The graph now displays the selected components.



- Drag the tables to re-arrange.
- Zoom in a bit to change the perspective.
- Select **Save** at the bottom of the screen and provide a name.

As shown below, the graph provides a clear view of how data flows. In this use case, data and physical product flow coincide. So, the Supply Chain model graph tells us that: Factories supply Assembly Plants which in turn supply Stores.



We have solved one part of the problem! Remember, the technicians said the issue stemmed from a manufacturing defect and not improper assembly. The defective parts must originate in one of the factories. But, which one? To determine this, we can generate a graph of data values and

relationships and follow the data trail. Before generating this type of graph we need to configure the add-on. To help with the configuration, let's export a copy of the current data model graph for reference purposes.

4. Click **Export** at the top of the graph. Choose the desired format and action—either open or save the exported file.

4.3 What's next?

Complete the final piece of the puzzle by following along in the next section. Solving with a data value graph [p 21] describes how to create a graph that will show data values and provide a report to management.

CHAPTER 5

Solving with a data value graph

This chapter contains the following topics:

5.1 Overview

This part of the tutorial will give you hands on experience creating a data value graph configuration. It demonstrates how add-on configuration options translate to generated graphs. If you would prefer more generic, high-level instructions, see Configuring graphs [p 49].

We will use our generated graph to prove which factory is producing the defective parts. Additionally, we will export a graph showing our findings to share with management.

5.2 Steps

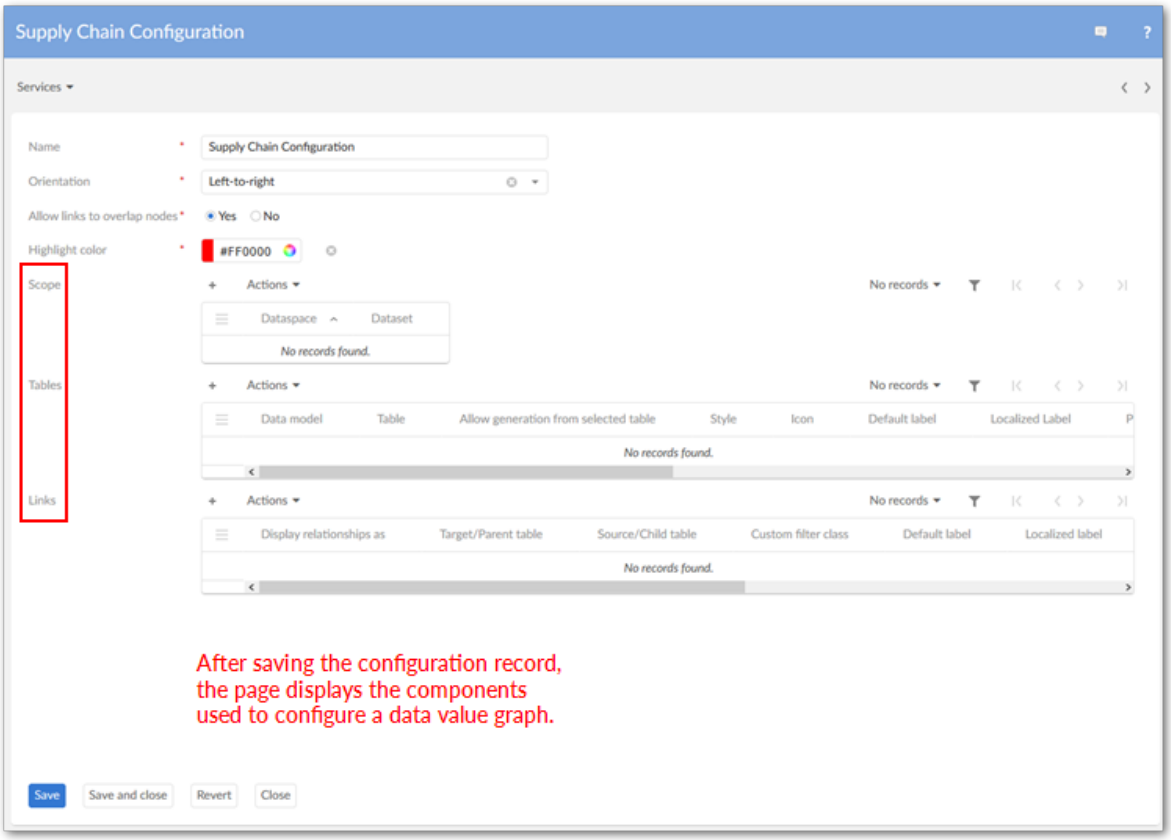
The following sections describe how to create a data value graph configuration and export a generated graph:

Creating a graph configuration

To begin, let's create a configuration that determines the data and relationships included in the graph, and how this information displays.

Navigate to *Administration > User interface > TIBCO EBX® Data Model and Data Visualization Add-on > Value and relationship graphs > Configuration* and create a new record. Leave the

Orientation property at its default of **Left-to-right**. After saving, but not closing, the page displays the following configuration sections required for graph generation: **Scope**, **Tables**, and **Links**.



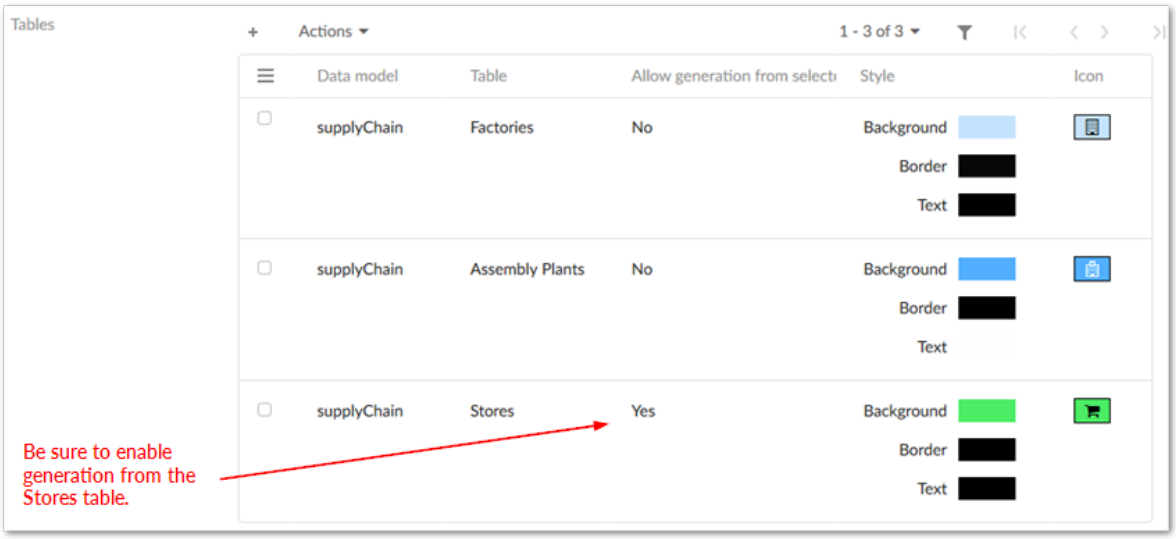
More about graph configurations: Data value graph configurations act as a container for graph settings. The settings determine accessibility, content, and how content displays. When multiple graph configurations include the same table, users can choose the configuration used to generate a graph. Although not demonstrated in this tutorial, a *Scope* restricts the dataset from which users can access the configuration.

Creating table configurations

We now need to decide which tables contain the data relevant to the supply chain issue (described in the tutorial's Prerequisites and setup [p 15]), and link them with our graph configuration. To accomplish this, we will create *table configurations* that associate tables with our graph configuration. To create the table configurations:

1. On the configuration page, click the ✚ icon in the **Tables** group to create a new record.
2. Choose the table using the **Data model** and **Table** fields. When creating the Store table, be sure to set the **Allow generation from selected table** property to **Yes**. This setting determines whether we can generate a data value graph from the table's **Actions** menu. When disabled, the option does not display as a menu option.
3. Optionally, change each table's style and icon. These features make it easier to differentiate between elements in the generated graphs.

4. Save and close. We will repeat the process for each table in the tutorial's sample data model. When finished, your **Tables** section should resemble the following:



Adding link configurations

The data model graph generated in the previous chapter of the tutorial shows us which relationships we need to drill into to find the origin of the problem. By adding these foreign key relationships to the graph configuration, we determine which data values display. This will allow us to trace the supply chain to find the source of the defective parts. All we have to do is use *link configurations* to tell the add-on which relationships to include.

What are link configurations? Each link configuration:

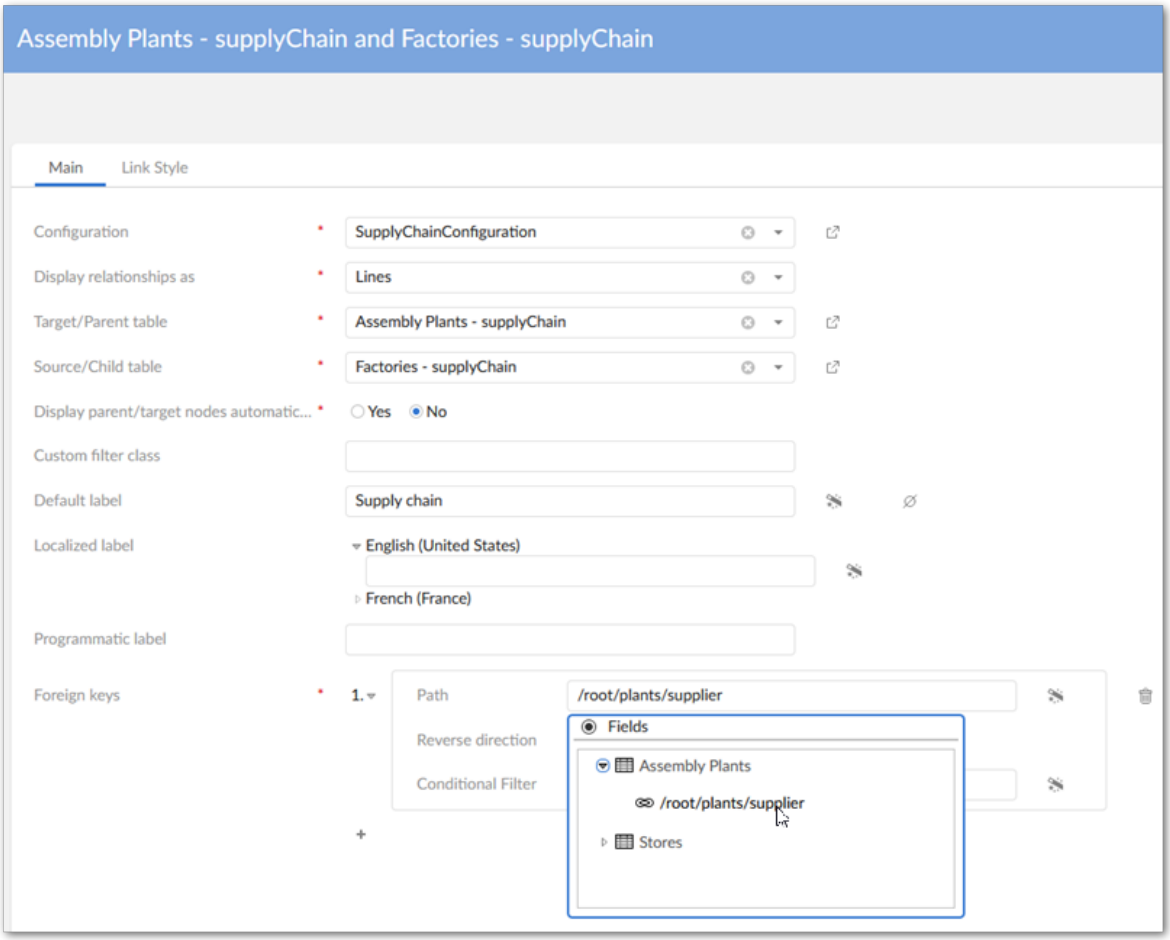
- uses paths in the data model to identify one or more foreign key relationships.
- defines the type of component the add-on uses to show this relationship when users generate a graph. The settings allow relationships to display as lines, or as parent and child containers (more on this below).
- sets the context for the relationship by indicating which table—linked by the foreign key—is the:
  - Parent or child: when setting the display type to *container*, nodes containing data values are nested within each other—parent nodes are rendered as the larger, outer nodes and children nodes within these.
  - Target or source: when using *line* for the display type, nodes containing data values are linked by lines with an arrow that indicates relationship direction.
- specifies the data value graph configuration to which these settings apply. This setting also determines which tables are available to link.

By referring to the data model graph generated earlier, we quickly see which relationships are needed. The following table summarizes the link configuration settings required to trace the defective parts to their originating factories.

Relationship type	Relationship and label	Foreign Key Path
Lines	• **Target/Parent table**: Assembly Plants • **Source/Child table**: Factories • **Label**: Supplied By	/root/plants/supplier
Containers	• **Target/Parent table**: Assembly Plants • **Source/Child table**: Stores • **Label**: Supplied By	/root/store/supplier

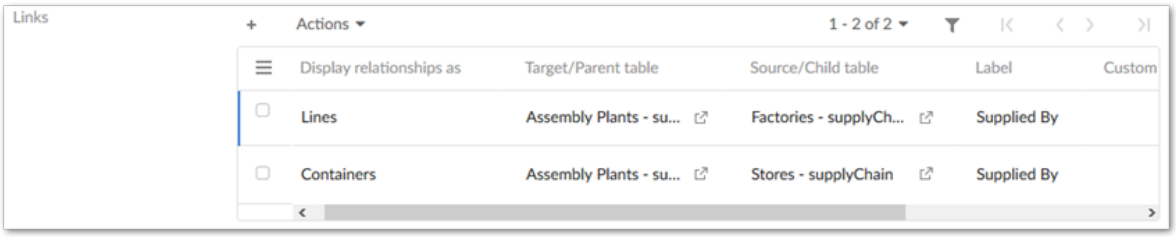
For each row in the table above, we will create a link configuration record populated with the row's values:

1. Create a new record in the **Links** group.
2. We can reference the exported data model graph to determine which links should be used. Alternatively, the table above lists the configuration settings. When entering the foreign key path, use the wizard as shown in the following image:



3. Repeat the first two steps to add the second link configuration and populate it with the correct table values.
4. Be sure to save and close all configuration records.

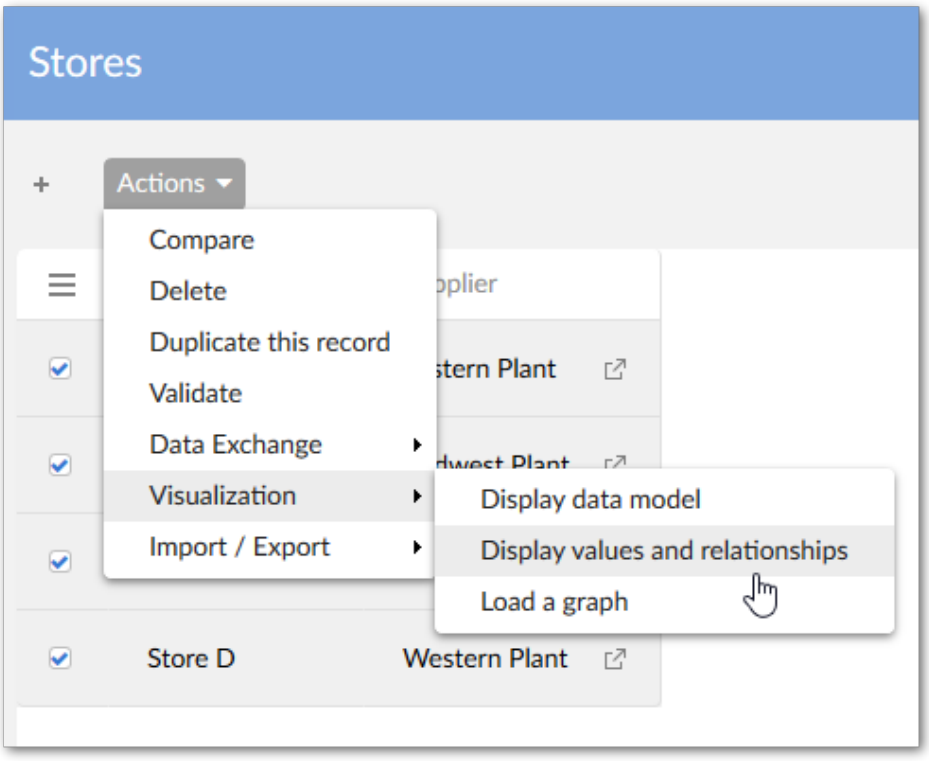
When finished, your **Links** configuration section should look like the following:



Locating the problem and sharing results

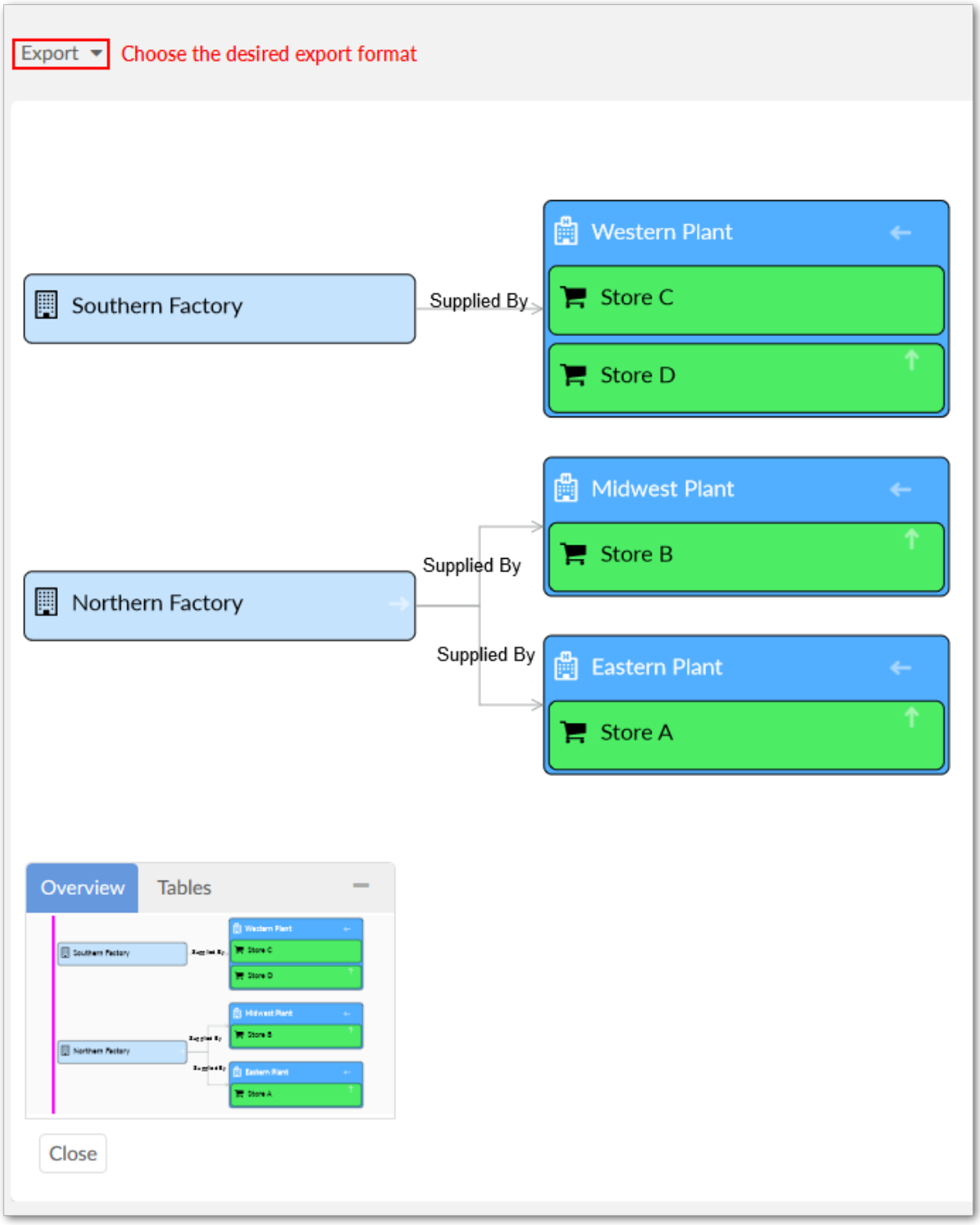
Now, it's time to follow the data trail. We can generate and share our graph by navigating to the Supply Chain dataset in EBX® and:

- Opening the Store table and selecting the records to display in the graph. The reports received about the returns have only been coming from stores A and B, but we would like to include all stores in the report to management; just to back up our findings. So, we will select all records in the Store table.
- From the table's **Actions** menu, select *Visualization* > *Display data using configuration*. Note that if multiple configurations existed for this table, this is where we could choose which to display.



Once the graph has rendered, we can follow the data to discover the source of faulty components. Right-click the Store D node and expand Assembly Plants, then click the related assembly plant and expand its factories. We will repeat this process for Store B and A. As the following image shows,

stores A and B are supplied by the Eastern and Midwest plants, respectively. This means the faulty parts must originate in the Northern Factory—the only factory to supply these plants. Congratulations, on discovering the source! Only one thing remains.



We can share our findings by simply clicking **Export** to download a PDF or PNG that we can include in our report to management.

Data model graphs

CHAPTER 6

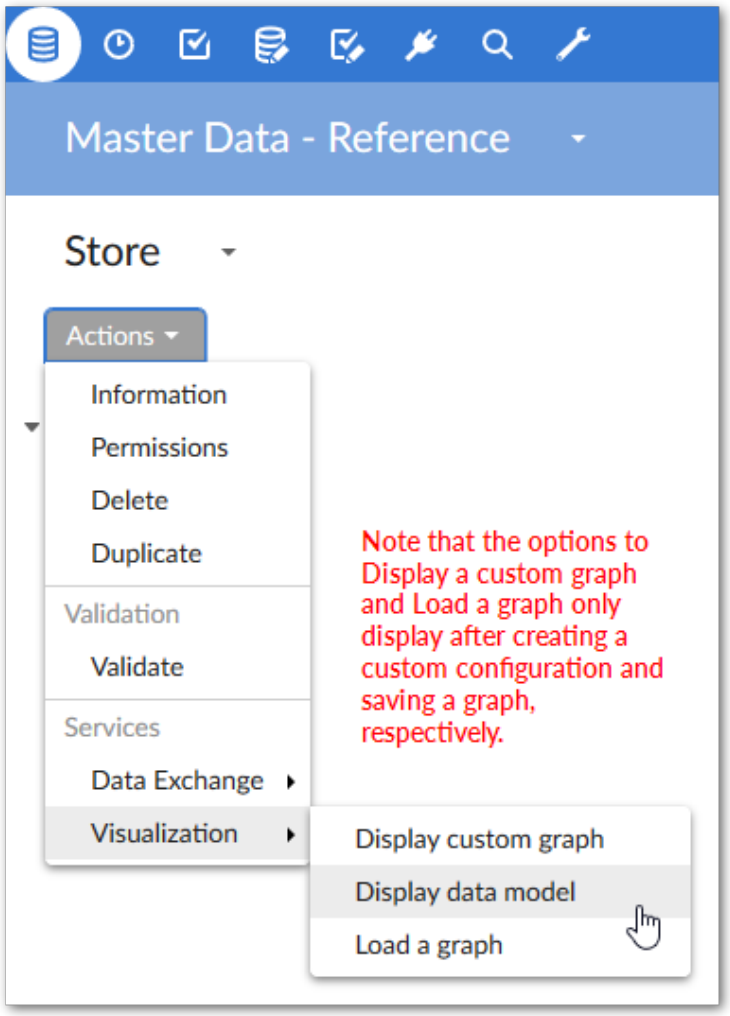
Generating and loading data model graphs

This chapter contains the following topics:

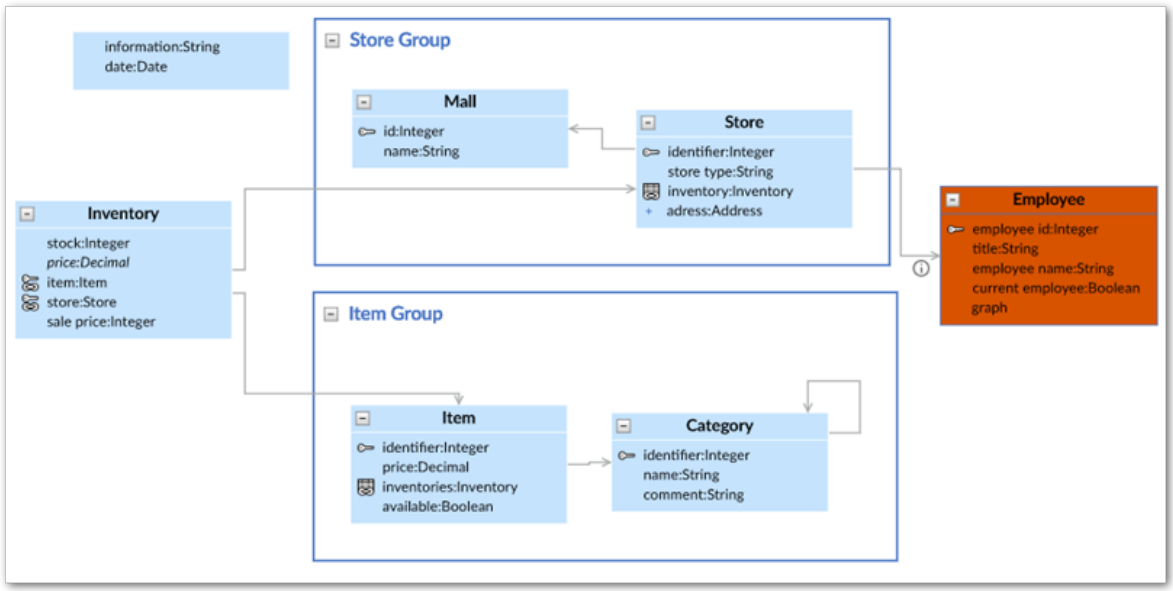
6.1 Generating graphs

Without configuring anything, the add-on can render a new data model graph. Simply open a data model, dataset, or table **Actions** menu and select **Display data model** under **Visualization**. If the option has been enabled by an administrator, you can choose which tables are rendered in the graph. See Changing display and configuration options [p 41] for information on configuring this option. When you run the service from a:

- Data model or Dataset: The table from the current model or dataset with the most links is centered in the graph.
- Table: The table is centered in the graph.



The following image shows an example data model graph:

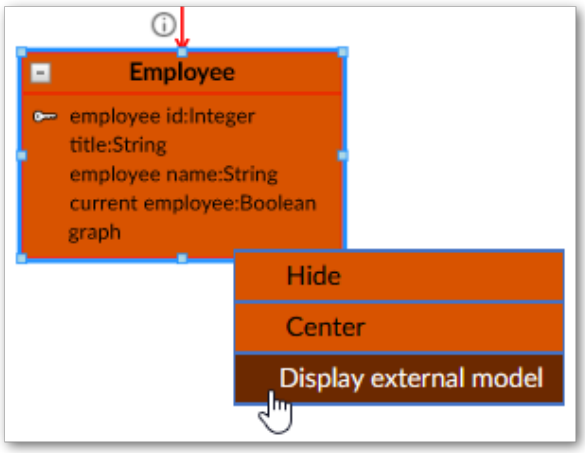


6.2 Displaying external models

If a data model graph contains a table from a different data model, you can enable display of the external model in the current graph. You can do this for any external table and each graph can contain multiple data models.

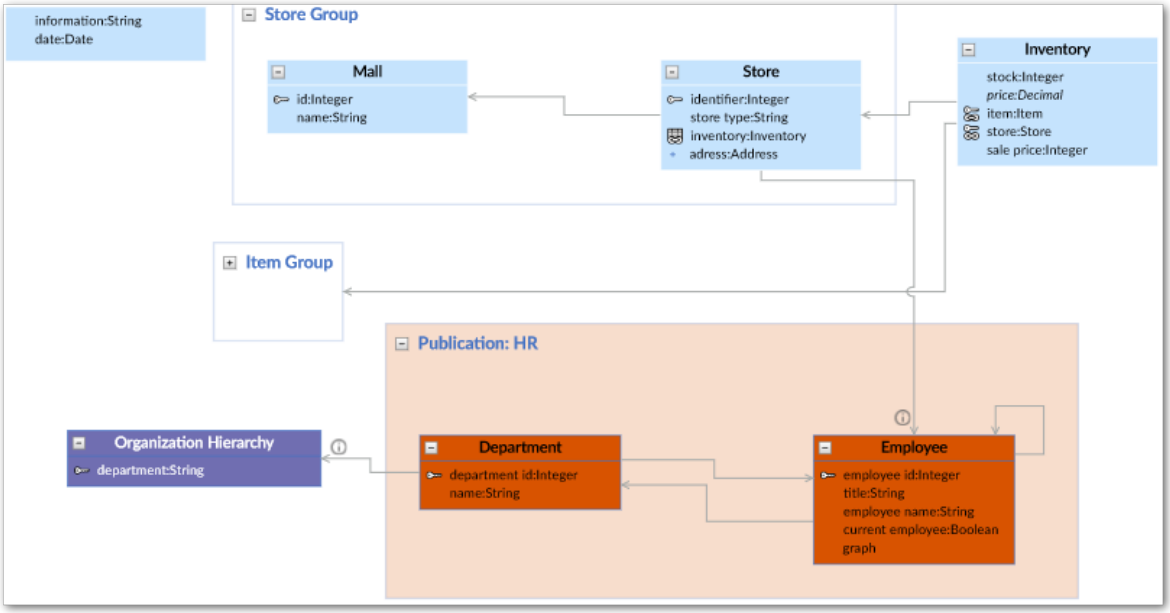
To display an external model in an existing data model graph:

1. Right-click an external table. A graph identifies external tables using the icon and an alternate color.



2. Select **Display external model**.

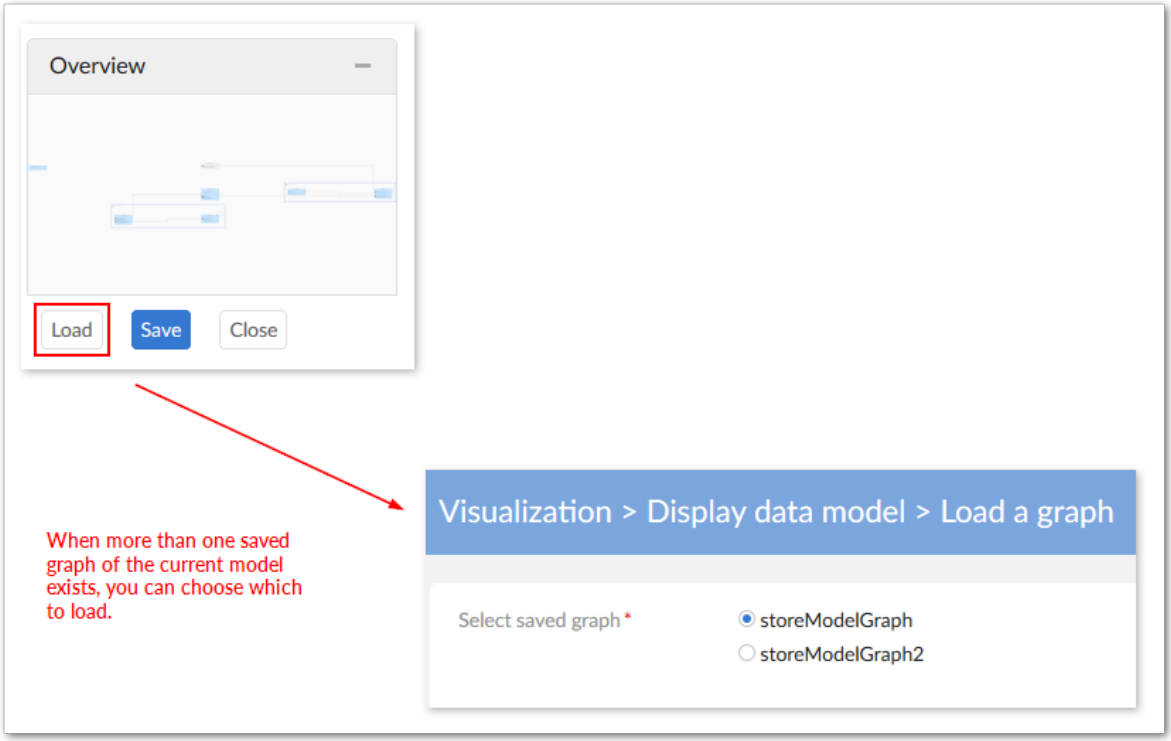
   As shown below the external model now displays in the graph, and the above process was repeated to display additional models. See Interacting with data model graphs [p 33] for additional information on using graph features.



6.3 Saving and loading graphs

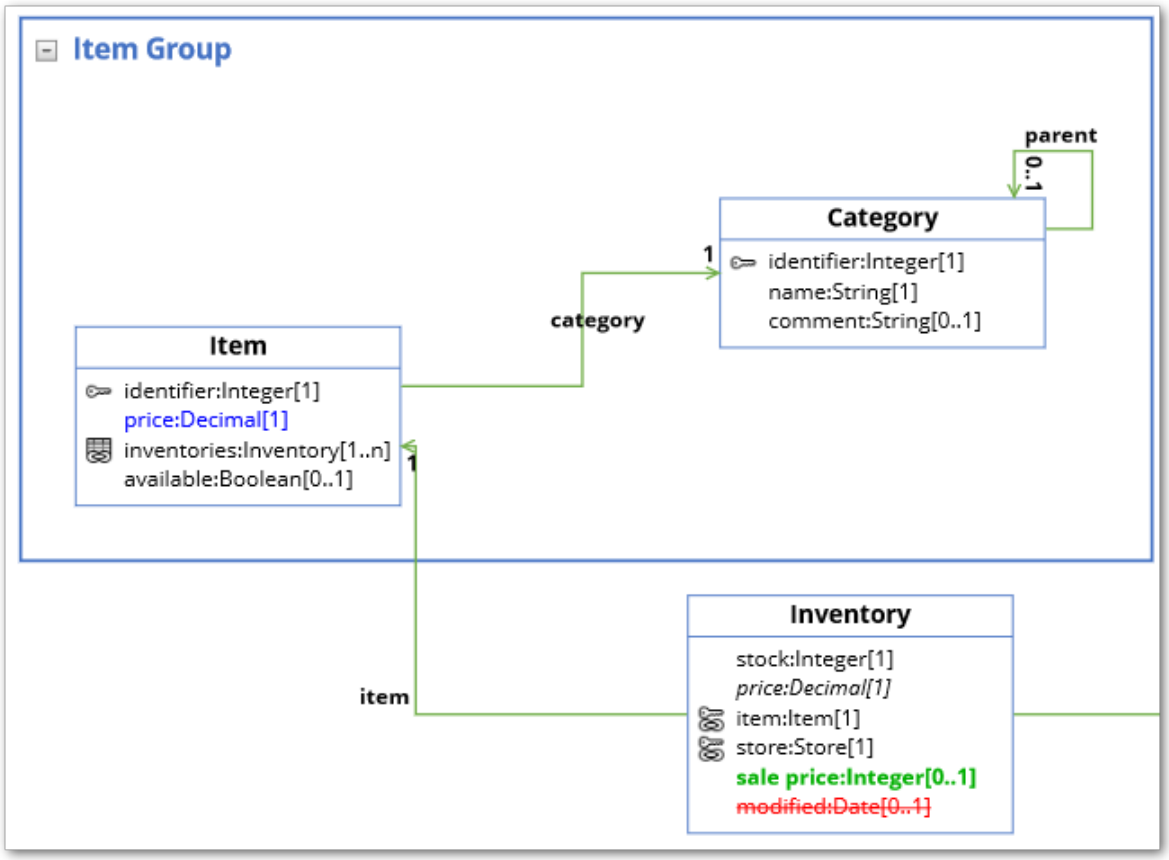
You can expand, collapse, determine display of, and rearrange graph components. Once satisfied with the appearance, use the **Save** button at the bottom of the graph to store the current layout and settings. As shown below, you can use the **Load** button to load a saved graph. When more than one saved graph of the current model exists, select the desired graph and click **Generate** at the bottom-right of the screen.

> **Note**
>
> If the loaded graph's underlying data model contains changes since its last save, the add-on allows you to choose whether to synchronize the graph. See Visualizing changes to models [p 31] for more information on synchronization.



6.4 Visualizing changes to models

If you are loading a graph after a change has been made to the underlying data model, a pop-up dialog displays and allows you to choose whether to update the graph. As shown below, updated graphs highlight new items in green, changed items in blue, and deleted items in red:



See also

Interacting with data model graphs *[p 33]*

Changing display and configuration options *[p 41]*

6.5 Permissions

If you do not have sufficient permission to view any tables in a given dataset, the **Display data model** service will not display. If there are certain tables or fields you are restricted from viewing, they do not display when you generate a data model graph.

See also

Changing display and configuration options *[p 41]*

Interacting with data model graphs *[p 33]*

CHAPTER 7

Interacting with data model graphs

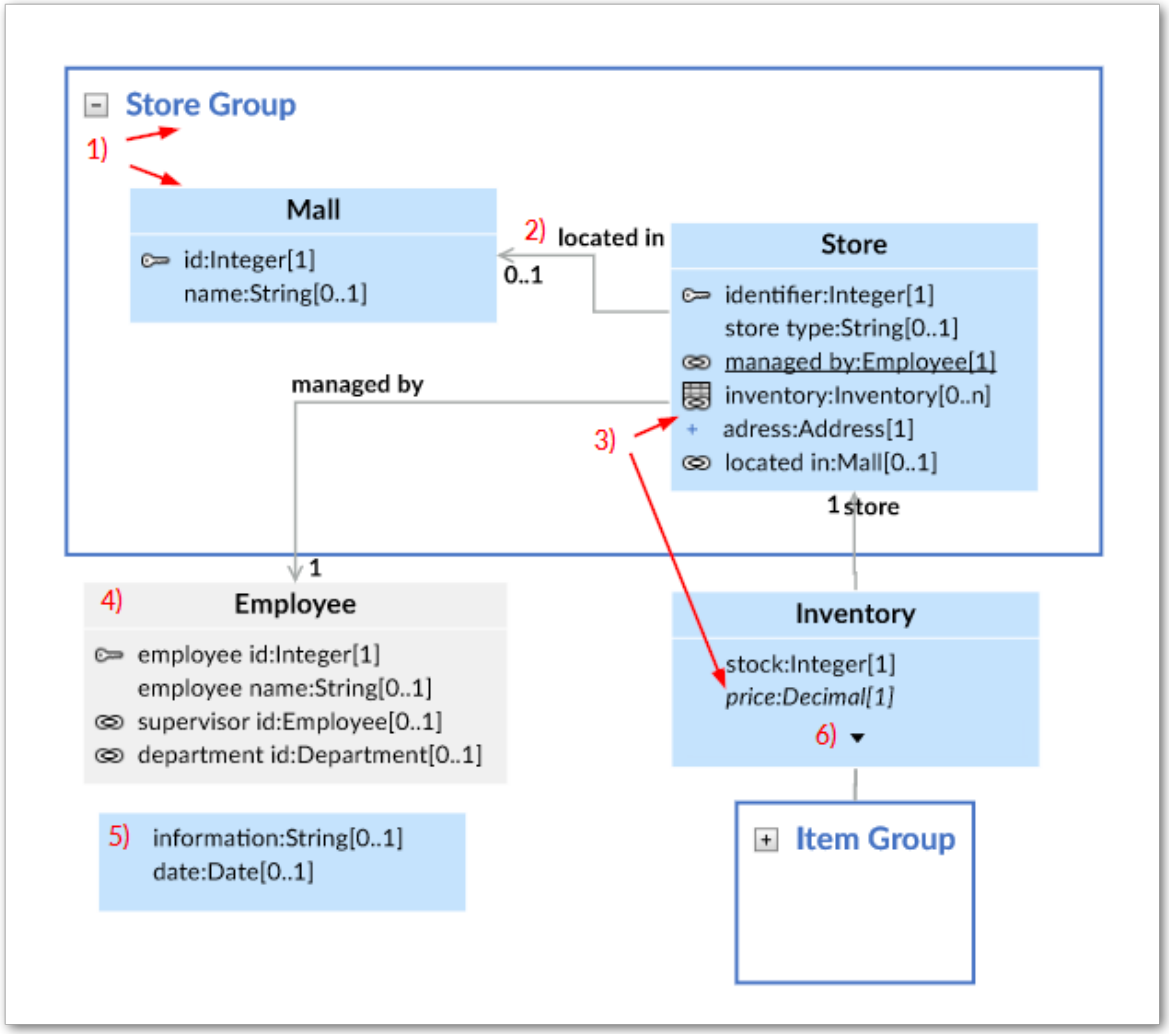
This chapter contains the following topics:

7.1 Overview

The following sections provide descriptions of what displays in a data model graph and available interactions:

7.2 How data model components display in graphs

What displays in a graph depends on the model content from which it was generated and the template settings. The following image and table describe data model graph components.



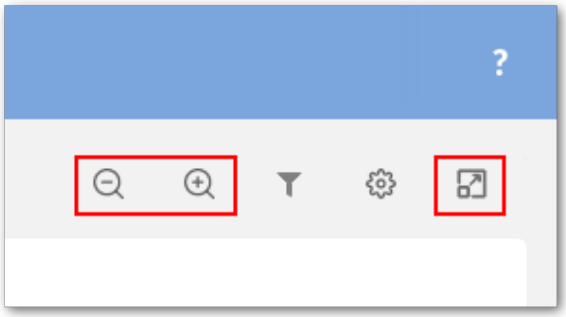
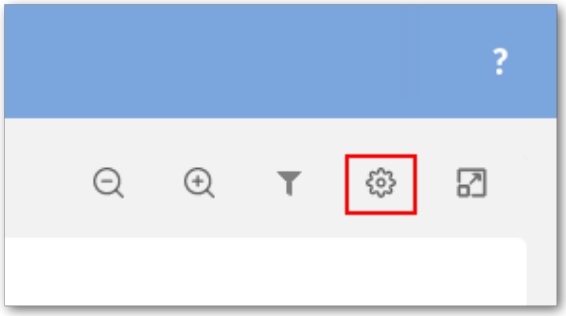
1) Groups and Tables	A box is automatically drawn around components included in a group. You can collapse, expand, re-size and rearrange groups. The bottom-right corner of the image shows a collapsed group. Tables display their labels and fields. Standard DMA icons are used to indicate keys and relationships. When a graph displays multiple models, each model behaves as if it is a group. For example, clicking the white space around elements from the base data model draws a box around all of the model's elements. You can then drag to move all elements at the same time. External models include the additional option of expanding and collapsing to show/hide included tables.
2) Relationships	Arrows indicate foreign key relationships and their direction. The labels correspond to each foreign key field.

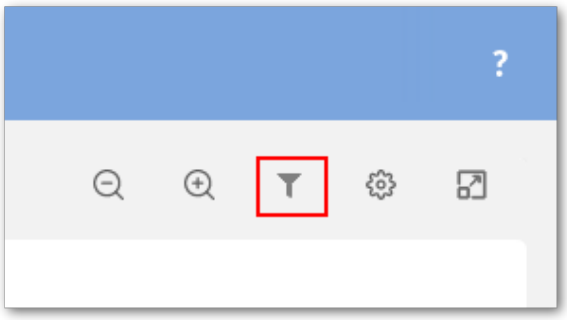
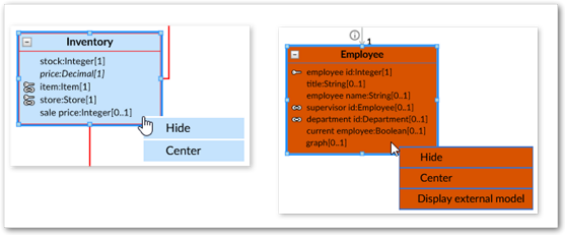
3) Underlined field, plus icon, and italic fields.	An underlined field belongs to a table located outside of the current dataset. The plus icon next to a field indicates you can expand to display more information, such as when fields are included in a complex type. Fields in italics represent inherited fields.
4) Table located outside of the current dataset	When a table is located outside of the current dataset, it will be shaded using a pre-defined color. You can display the external table's data model in the current graph using the right-click **Display external model** menu option.
5) Orphan fields	Fields that are not part of a table display without a title.
6) Display additional fields	When a table holds more fields than can display, use the arrow icon to expand/collapse.

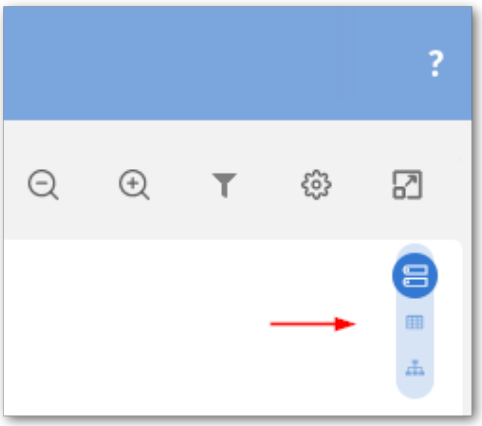
7.3 Available actions when viewing a graph

There are several ways in which you can interact with and customize a data model graph. Whether you decide to alter a graph or leave the defaults in place, you have the option to share it with other

users. See Sharing a graph [p 38] for information on sharing graphs. The following table describes what you can do when viewing a data model graph:

Action	Description
Navigation	You can navigate a graph by: • Clicking and dragging in a graph's whitespace to change the view. • Dragging the highlighted rectangle in the **Overview** map to display the desired portion of the graph. Note that you can also move the **Overview** map to any of the graph's corners by selecting its title bar and dragging. • Using your mouse wheel, or keyboard arrows to scroll.
Zooming and fullscreen	The icons at the top of the graph screen allow you to open the graph in a fullscreen view, zoom, and reset to the original magnification.  Use the combination of the Ctrl key and your mouse wheel to zoom in and out. Alternatively, hold down the Ctrl key and press the + and - keys to zoom.
Arranging graph components	Drag and drop components to arrange them. Once the layout is satisfactory, you can save the graph for re-use.
Showing/hiding components and changing look	Click the **Template** icon to open the **Graph template configuration** window.  Adjust the settings as desired. If you have a question about a property or setting, open its tooltip for more information. Once you are finished, click **Save** to record your changes to the template.
Locating tables and toggling table display	Click the filter icon to choose which tables and data models display.

Action	Description
	 The filter pane allows you to: • Search for tables and external data models shown in the current graph. • Toggle display of tables and external data models. Select **Apply** to update the graph display. Note that the  icon indicates that the graph includes an external table, but its data model is not displayed. • Use the **All**, **Displayed**, and **Hidden** tabs to view the tables and models under these categories.
Exporting a graph	Click **Export** at the top of the screen to export and download the current graph in either PDF, PNG, or SVG format.
Resizing	Group boxes automatically resize when you change the arrangement of its components. You can resize tables by clicking to select and dragging anchor points.
Right-click menu	When you right-click a table, you can hide or center it in the graph (shown left). Right-clicking an external table gives you the additional option of displaying the external table's data model (shown right). 
Level selector	If a graph displays external models, you can change the level of detail shown using the level selector. The levels available to select include: • The default **Field** level, which displays expanded models and tables to show all model components including fields. • The **Table** level displays models and all tables collapsed to hide their fields. This level also hides any orphan fields in the graph. • The **Model** level displays all data models collapsed to hide tables and fields.

Action	Description
	
Load, Save, and Close the graph	The buttons below the graph **Overview** box allow you to save and close the current graph and load a different graph. For information on sharing a saved graph, see the Sharing a graph [p 38] section below.

See also

Changing display and configuration options [p 41]

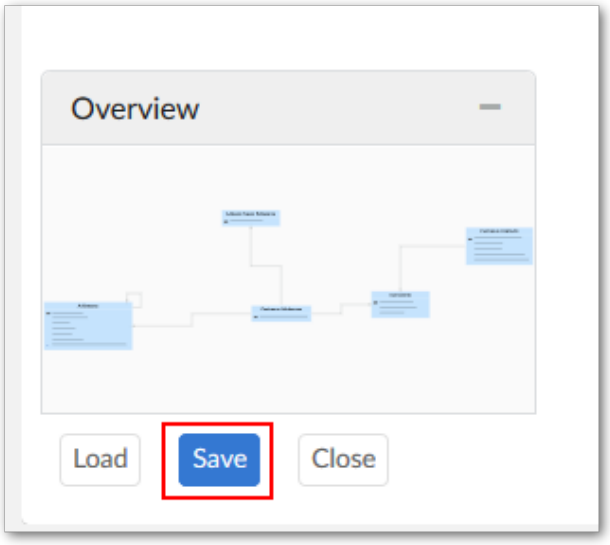
Generating and loading data model graphs [p 28]

Sharing a graph

The add-on makes it simple to share saved graphs with other users. The option to share a graph is available when saving. Once you share a graph with a user, it displays from the list of available graphs when they choose to load a graph. Any tables for which a user does not have sufficient permission to view will not display in the shared graph. Unless explicitly defined as such, recipients of a shared graph are not considered its owners. If you neither are an administrator or owner of a graph, you can only save changes to the graph's description. However, if you make other changes that impact the graph, you can save the graph using a different name to keep your changes.

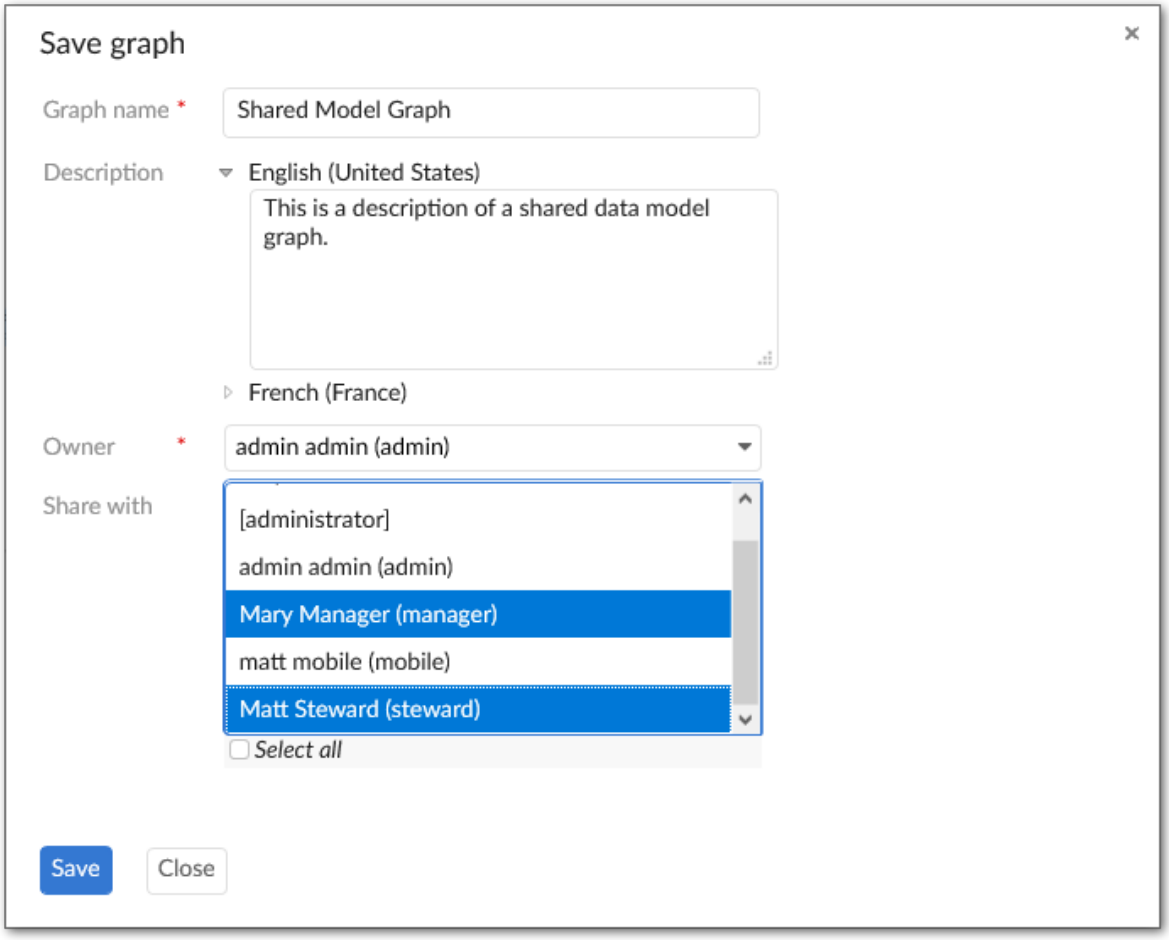
To access the share feature:

1. Select **Save** at the bottom of a data model graph.



2. Provide a name for the graph and optionally include a description.
3. Optionally, set the graph's owner. By default, you are considered the graph's owner. However, you can use the **Owner** property to specify another profile as owner. Note that administrators can also edit graph owners.

4. Use the **Share with** box to select one or more users/roles with whom to share the graph.



Permissions for shared graphs behave as you would probably expect. Without sufficient permissions graph elements do not display. Even if you have permission to view a table that was hidden in a graph shared with you, it does not display. You would have to generate a new graph of the model to display it. If permission to view a graph element is revoked after receiving a shared graph, the impacted element does not display.

CHAPTER 8

Changing display and configuration options

This chapter contains the following topics:

8.1 Overview

If you have access to the EBX® **Administration** area, you can create and edit data model graph configurations and templates. Configurations determine which model the add-on renders when users generate graphs. The add-on can generate graphs of EBX® data models and models from external sources. Templates are applied to configurations to determine the look and feel of generated graphs. These topics are discussed further in the following sections:

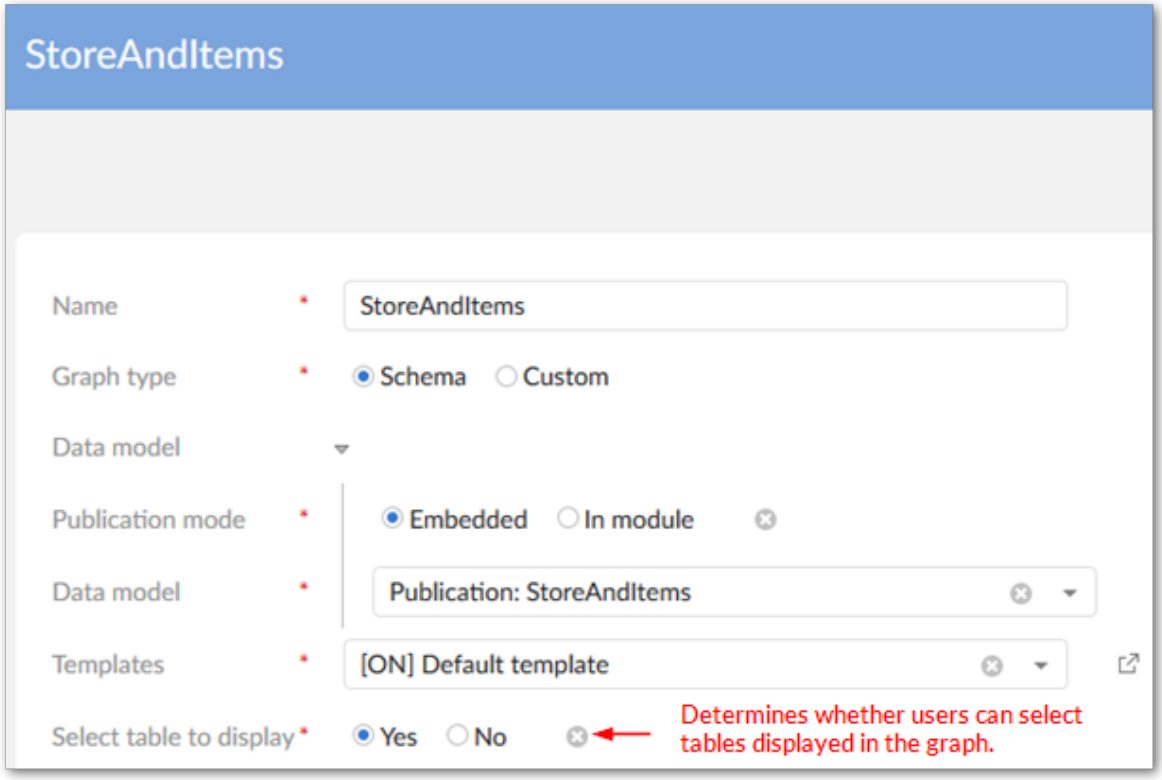
- Creating and editing graph configurations [p 41]
- Working with graph templates [p 42]

8.2 Creating and editing graph configurations

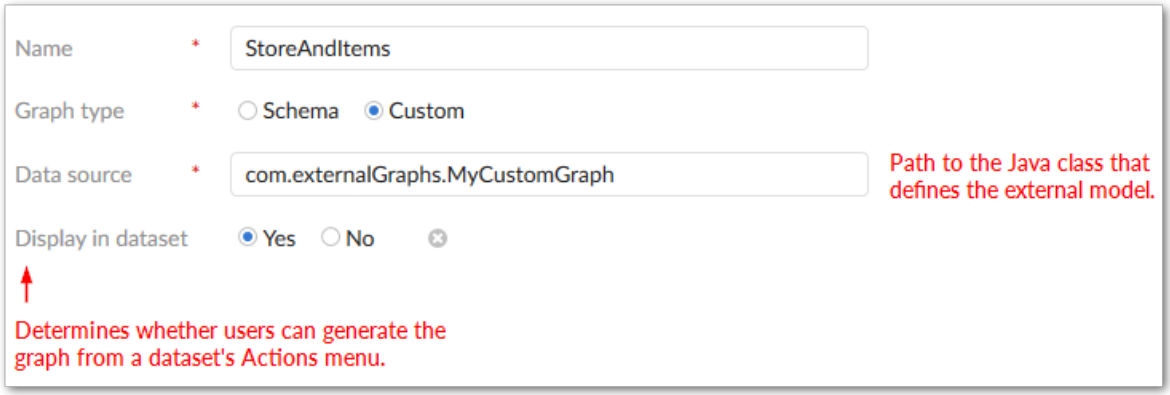
When creating or editing a data model graph configuration, the available options depend on the **Graph type** setting. Select the **Schema** option to base the generated graph on an EBX® model. Choose **Custom** to generate a graph from an external data source.

When set to **Schema**, the **Data model** group displays and allows you to:

- Choose the model associated with this configuration using the **Publication mode** and **Data model** fields.
- Select the template used by the configuration.
- Specify whether users can select which tables display when generating the graph.

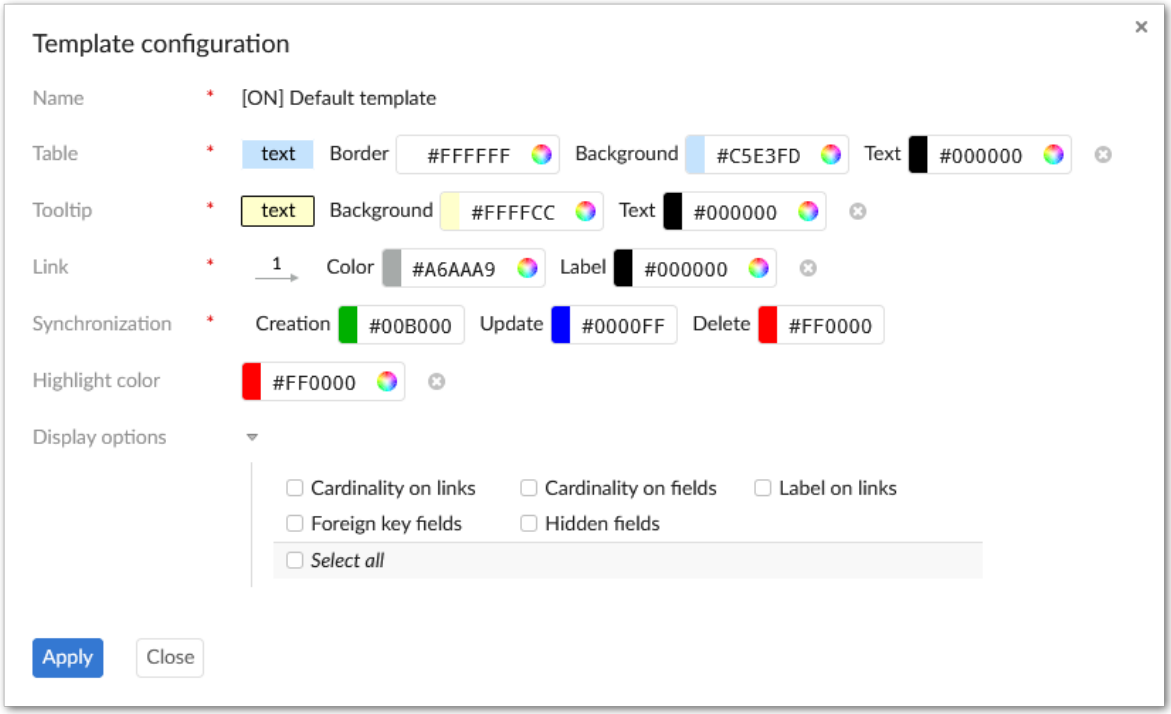


If you set the **Graph type** to **Custom**, you must supply the fully qualified path to a Java class. To make the graph accessible to users from the **Actions** menu, enable the **Display in dataset** property. See Generating a model graph from an external source [p 99] for information.



8.3 **Working with graph templates**

You can create and edit templates that determine how generated graphs display. Editable features include color choices and visibility of graph components. Note that when users view a data model graph, they can edit the template applied to the graph. Any saved changes apply only to the graph they are viewing and not the default template. Additionally, these changes only last for the duration of the user's current session. After logging out, changes are reverted.


Template configuration
Name
[ON] Default template
Table
text
Border
#FFFFFF
Background
#C5E3FD
Text
#000000
Tooltip
text
Background
#FFFFCC
Text
#000000
Link
1
Color
#A6AAA9
Label
#000000
Synchronization
Creation
#00B000
Update
#0000FF
Delete
#FF0000
Highlight color
#FF0000
Display options
Cardinality on links
Cardinality on fields
Label on links
Foreign key fields
Hidden fields
Select all
Apply
Close

Data value graphs

CHAPTER 9

Overview

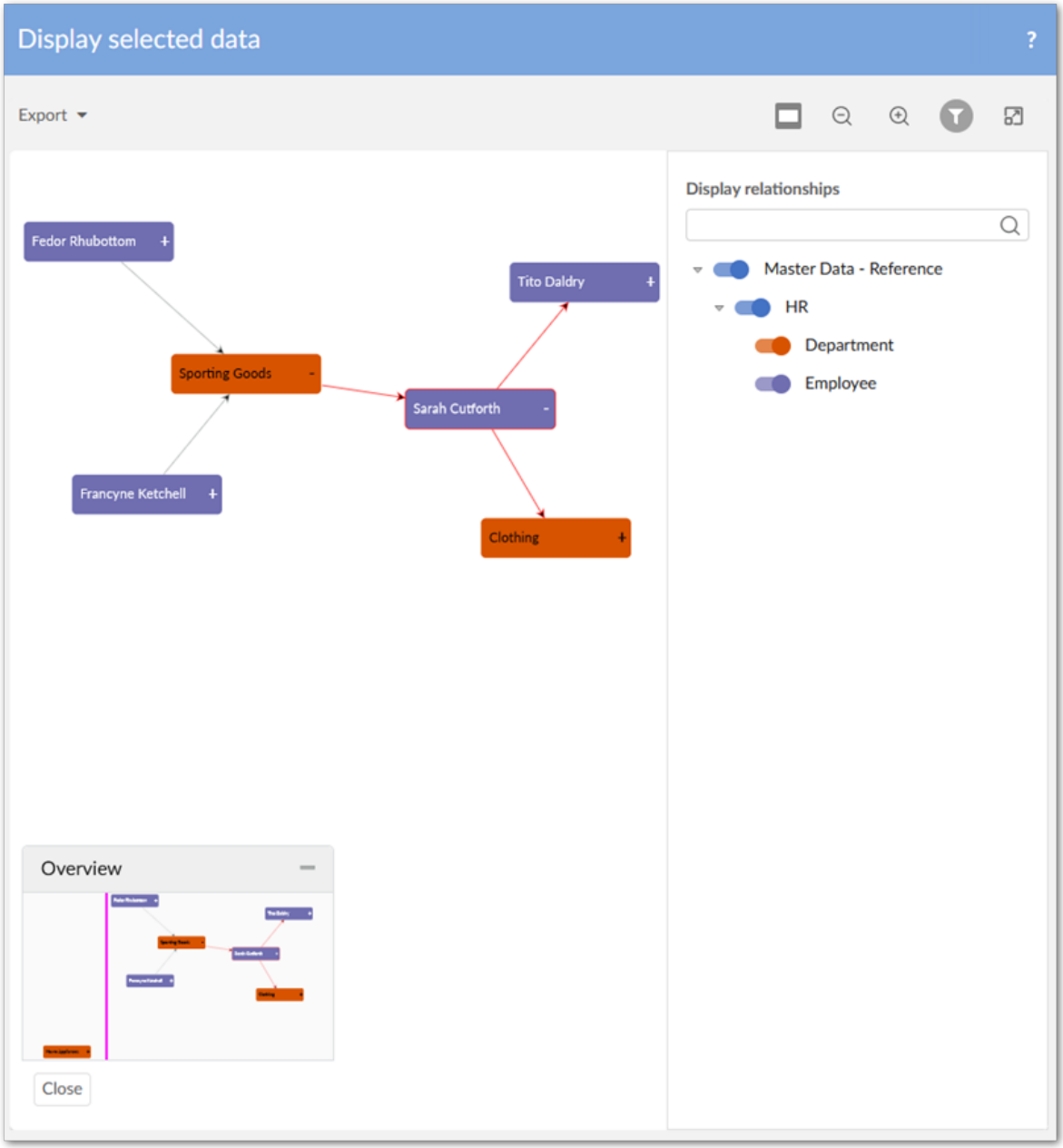
This chapter contains the following topics:

9.1 What are data value graphs?

A data value graph is a visual representation of how data values relate to each other. A graph can display relationships between data in the same table, different tables, and even tables from multiple data models. By default, you can view a graph of selected data values [p 46]. Additionally, you can choose a data value graph configuration [p 47] created by an administrator to display a customized graph of selected data values. See the sections below for more details on each graph type.

Default graphs

A default data value graph requires no advance configuration. Default graphs display data values in colored boxes and use arrows to represent the relationships between the data values. The following image provides an example of a default data value graph:



See also

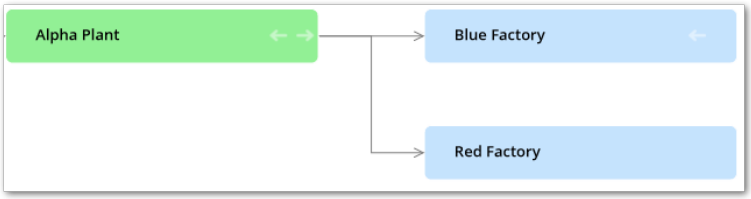
Generating data value graphs *[p 65]*

Using data value graphs *[p 69]*

Configured graphs

Administrators can create configurations to determine which values and relationships can display. They also configure how the add-on renders these relationships by assigning a display type to each one. The table below provides examples of the available display types. To display a graph, users select one or more values in a table and run the table's **Display data using configuration** service. Note that this service is available only from tables included in a configuration.

The following table outlines the available relationship display types:

Type	Example and description
As Lines	As shown in the following image when you set the display to lines, values display in rectangular nodes and lines between nodes represent relationships. 
As Containers	As shown in the following image containers show a parent/child relationship as nodes within nodes. Outer nodes are the parents of the smaller, inner nodes. 

The sample tutorial [p 14] provides detailed configuration instructions. You can follow the tutorial step-by-step, or feel free to use the generalized instructions in the Configuring graphs [p 49] chapter to create your own configuration.

See also

Configuring graphs [p 49]

Generating data value graphs [p 65]

Using configured data value graphs [p 71]

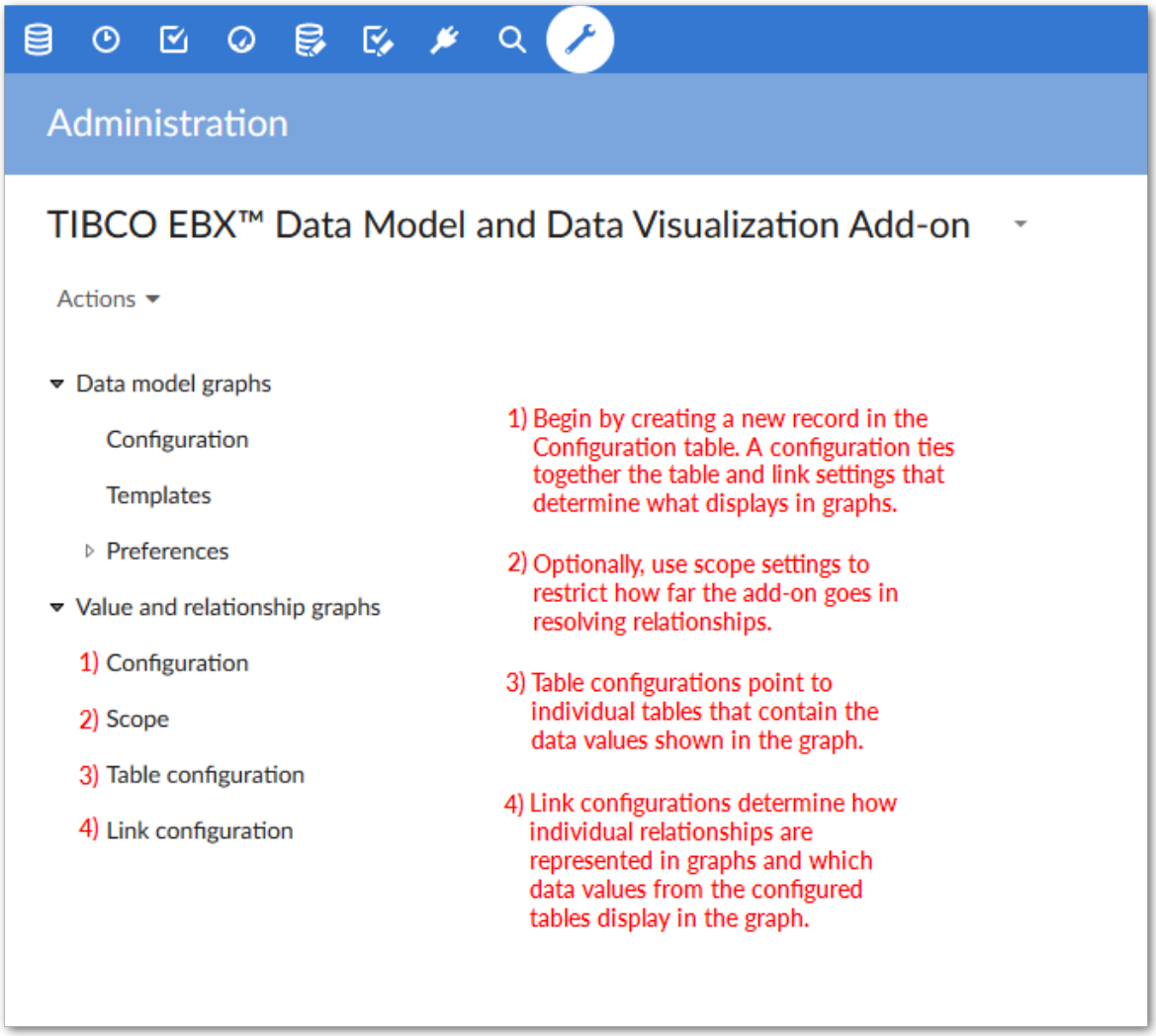
CHAPTER 10

Configuring graphs

This chapter contains the following topics:

10.1 Overview and configuration roadmap

The following image outlines the path to configuring a data value graph. The Options overview [p 50] section describes configuration options and settings in more detail.



10.2 **Options overview**

You perform all of the following configuration actions in the *Administration* > *User interface* > *TIBCO EBX® Data Model and Data Visualization Add-on* > *Value and relationship graphs* group:

- Main configuration page [p 51]
- Graph scope [p 52]
- Graph tables [p 52]
- Link configurations [p 53]

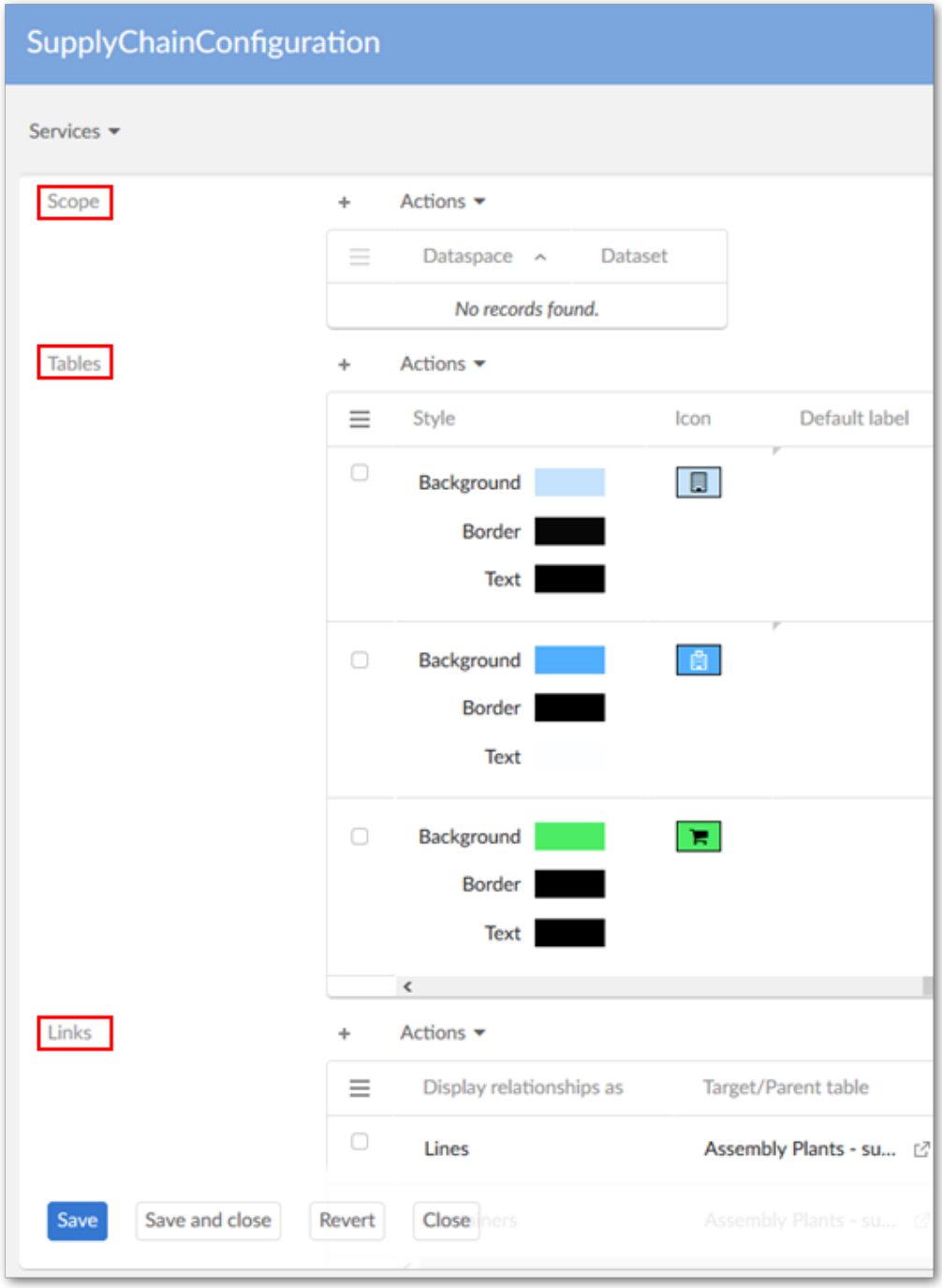
Main configuration page

You can create a graph configuration by adding an entry in the **Configuration** table. Each graph configuration is used to couple all other settings that determine what data displays in a graph and how the data and relationships are rendered.

> **Note**
>
> You might find it helpful to use the main configuration page when changing or updating graph settings. Since each configuration provides easy access to all configuration settings pertaining to a particular data value graph, you wouldn't have to hunt through configured tables, or links to find the desired setting.

When creating a graph configuration, you use the **Orientation** setting to determine how the graph displays line relationships. You can specify Left-to-right, Right-to-left, Bottom-up, or Top-down. The add-on begins with the root node and renders remaining nodes in the specified direction. Users can change display orientation when opened, but this does not effect the configuration setting. Each time the graph is opened, it uses the configured settings.

After filling in required fields, you can save (but not close the configuration) to use it as a quick access point for remaining configuration options (shown below). When you create scopes, table and link configurations directly from this page, they are automatically associated to the main graph configuration.



Graph scope

Optionally, create a scope. The **Scope** table allows you to restrict the dataspaces and datasets used in graph generation.

Graph tables

Use the **Table configuration** table to include tables in this configuration. Users that can view data contained in these tables can then generate graphs once you define the relationships. When adding a

table, use the **Allow generation from selected table** property to specify whether the option to generate a graph displays in the table's **Actions** menu.

You have several options for defining how labels display in the generated graphs (see Customizing labels [p 55] for more detailed information on labels):

- The **Default label** property allows you to enter text to display on a node. Alternatively, you can select the wizard icon to choose from related data model elements.
- The **Localized Label** group allows you to select a data model component from which the component's localized label from the DMA will be used. You can also enter the text to display.
- The **Programmatic Label field** field allows you to enter the path to a Java class that defines labels.

You can use a combination of default and localized labels. However, the order of priority is Programmatic, Localized, and Default. So, if you use a programmatic label, it overrides all other settings.

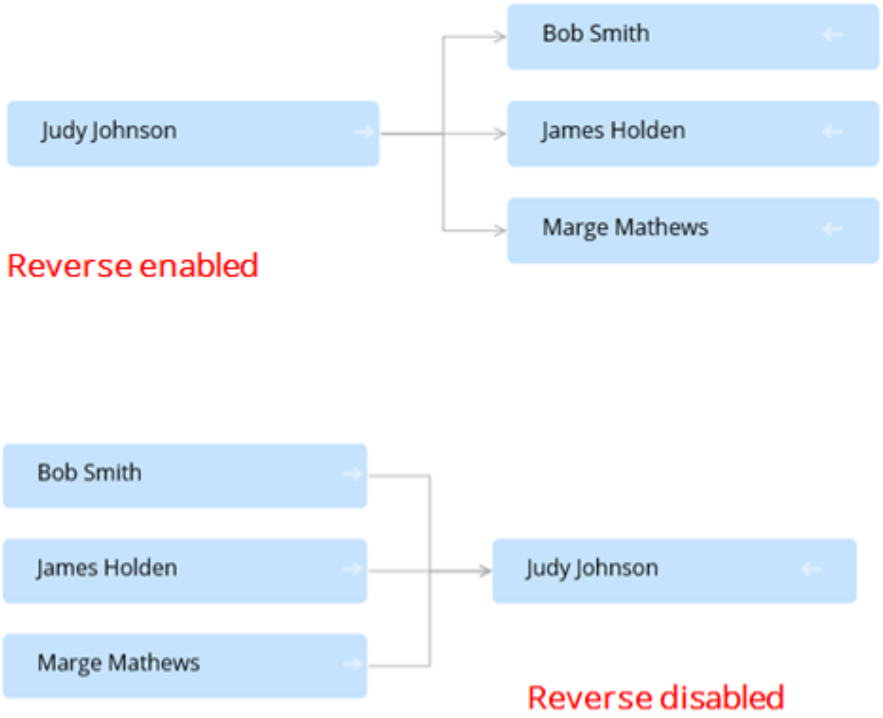
Use the settings in the **Node style** tab to make it easier for users to differentiate between sources when viewing graphs. You can also use the API to create custom nodes. See Customizing styles [p 57] for more information on these settings.

Link configurations

Define the relationships between data and how these relationships display using the **Link** table. Create as many links as required and save and close when finished. Each link is defined by:

- Specifying the link relationship type; one graph can display multiple types. See Overview [p 46] for examples of how each relationship type displays in the graph.
- Setting the **Source**/**Target** when displaying relationships as lines and **Parent**/**Child** for containers. You can also specify whether the first level of parent and target nodes automatically display.
- Entering the foreign key path and direction. The **Path** points to the foreign key and the direction property determines the direction of relationship lines. This field allows you to add multiple paths to account for foreign key references to multiple tables. See Filtering data values [p 61] for information on using the filter.

Note that the add-on automatically sets the direction value when the foreign key references another table. When **Reverse direction** is enabled, the graphs generated using self-referencing foreign keys render the relationships in reverse (shown below).



See also

Generating data value graphs *[p 65]*

Cloning and deleting graph configurations *[p 63]*

CHAPTER 11

Customizing labels

This chapter contains the following topics:

11.1 Overview

The add-on gives you full control over the labels that display on data value graph nodes and relationships. Administrators can configure the following label types for nodes and links:

- Default: You can enter text to use for the default labels. Alternatively, you can select the wizard icon to choose from related data model elements. Labels based on model elements dynamically update based on changes to the associated value.
- Localized: This type of label leverages EBX®'s data model localization feature. As with the default label you can enter text, or specify dynamically updated model elements.
- Programmatic: You can point to a Java class that defines how labels display.

You can use a combination of a fixed label and XPath for default and localized labels. However, programmatic labels override the other label types. So when you define a programmatic label, it is the only label that displays. If you choose not to specify a label, each graph element's default label displays.

> **Note**
>
> When nodes are filtered out or hidden due to permissions, the hidden node labels do not display. For instance, a label configuration may specify that nodes from table A display their own label and the label linked from table B. However, if table B is hidden, only table A's label displays.

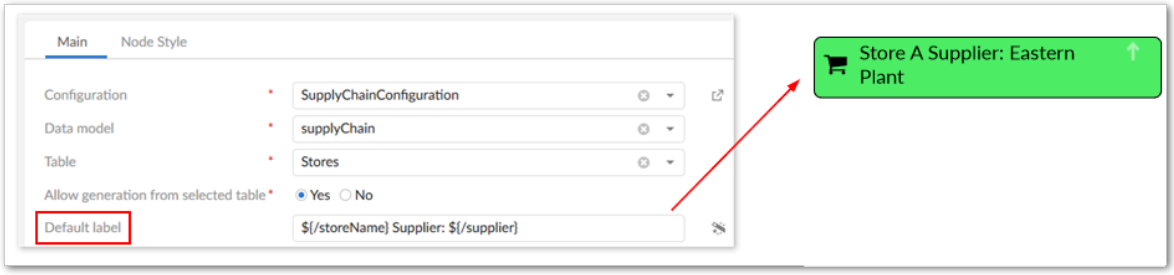
A maximum of three lines display on node labels. Once the maximum is reached, **...** displays and the full label can be viewed by hovering your mouse over the label.

11.2 Where to setup customization and examples

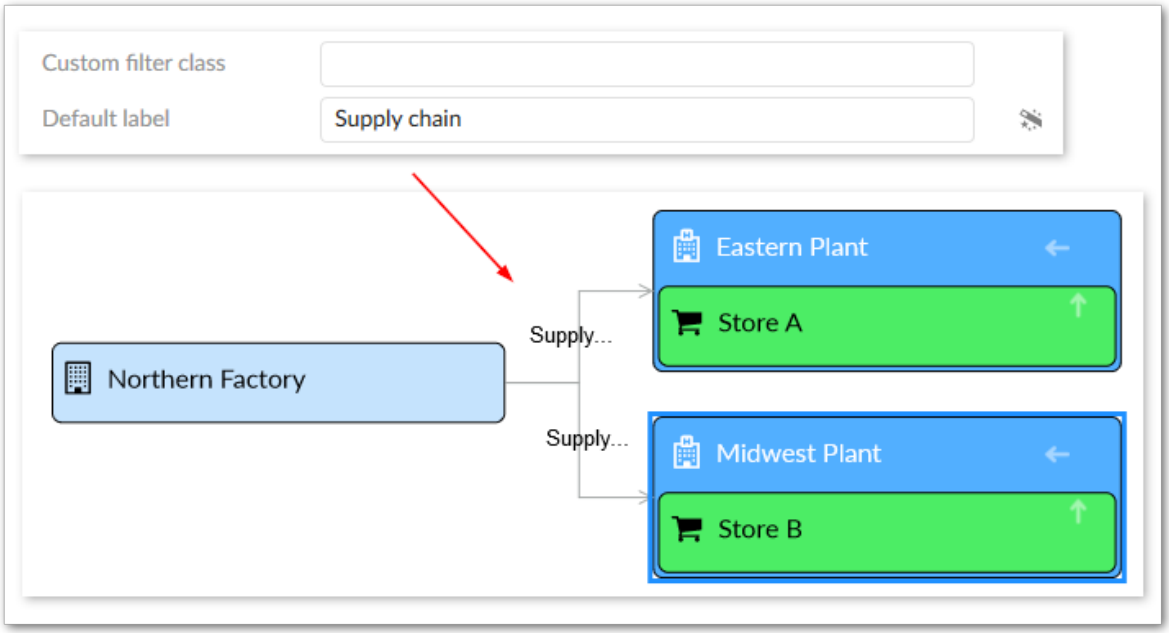
Administrators can configure label settings in the following locations:

- Node labels: *Administration > User interface > TIBCO EBX® Data Model and Data Visualization Add-on > Value and relationship graphs > Table configuration*. The following

image shows an example of combining model elements and text so that the label shows which assembly plant supplies a particular store:



- Link labels: *Administration > User interface > TIBCO EBX® Data Model and Data Visualization Add-on > Value and relationship graphs > Link configuration*. The following example shows a link label defined with plain text that is too long to display. You can hover your mouse over the link to display the full label. By right-clicking the link and selecting **Show link details** you can view the full label along with other details.



> **Note**
>
> When there are multiple links between a source and target, no label displays. However, users can right-click and open the link details to view all labels.

CHAPTER 12

Customizing styles

This chapter contains the following topics:

12.1 Overview

You can customize styles for data value graph nodes and relationships. Customization can be as basic as setting some default options, or as complex as basing style on the outcome of an evaluated XPath expression. You can also implement a Java class:

- to define programmatic styles for links. A Java class allows you to define more complex logic, such as applying a style to only a specific foreign key path in the graph.
- to create custom node templates. This gives you fine-grained control over the appearance of nodes. See Node templates [p 116] for detailed instructions.

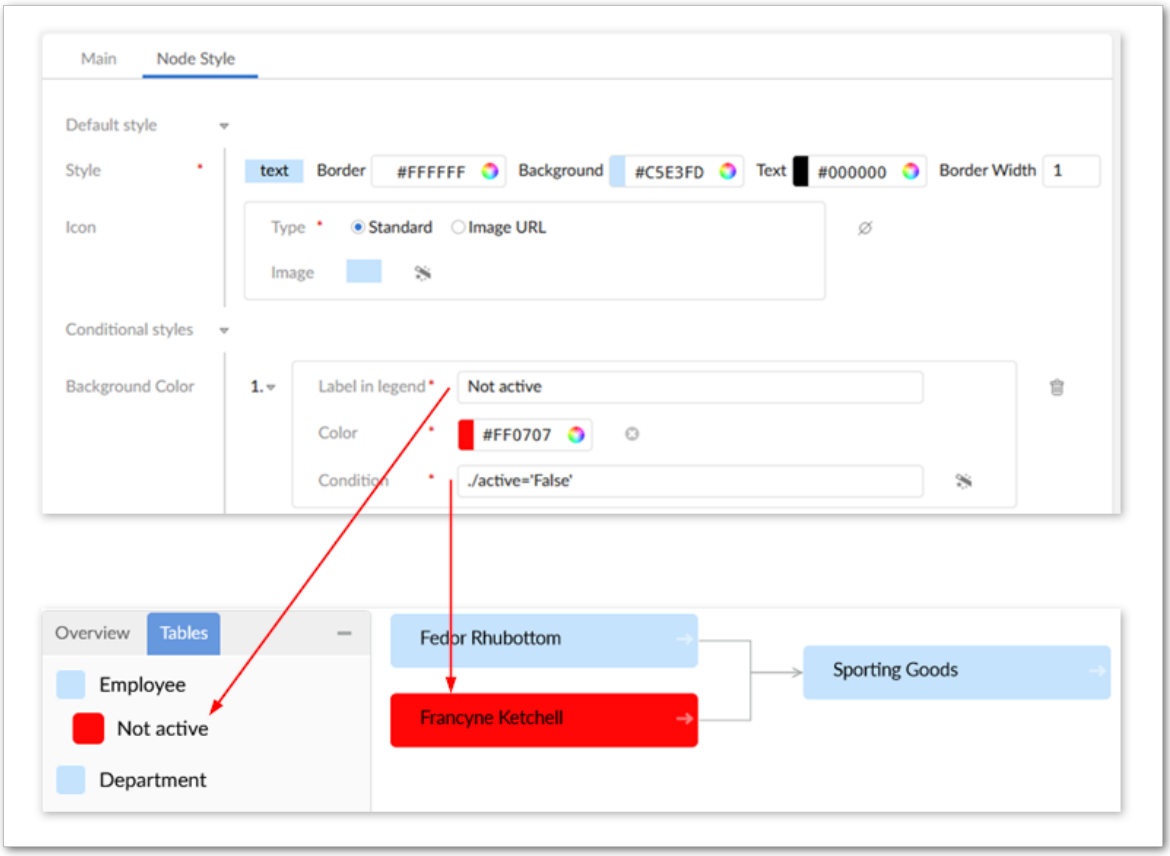
> **Note**
>
> Customization of link styles is only available for relationships set to display as lines and not containers.

12.2 Where and how to customize

Administrators can customize display in the following locations:

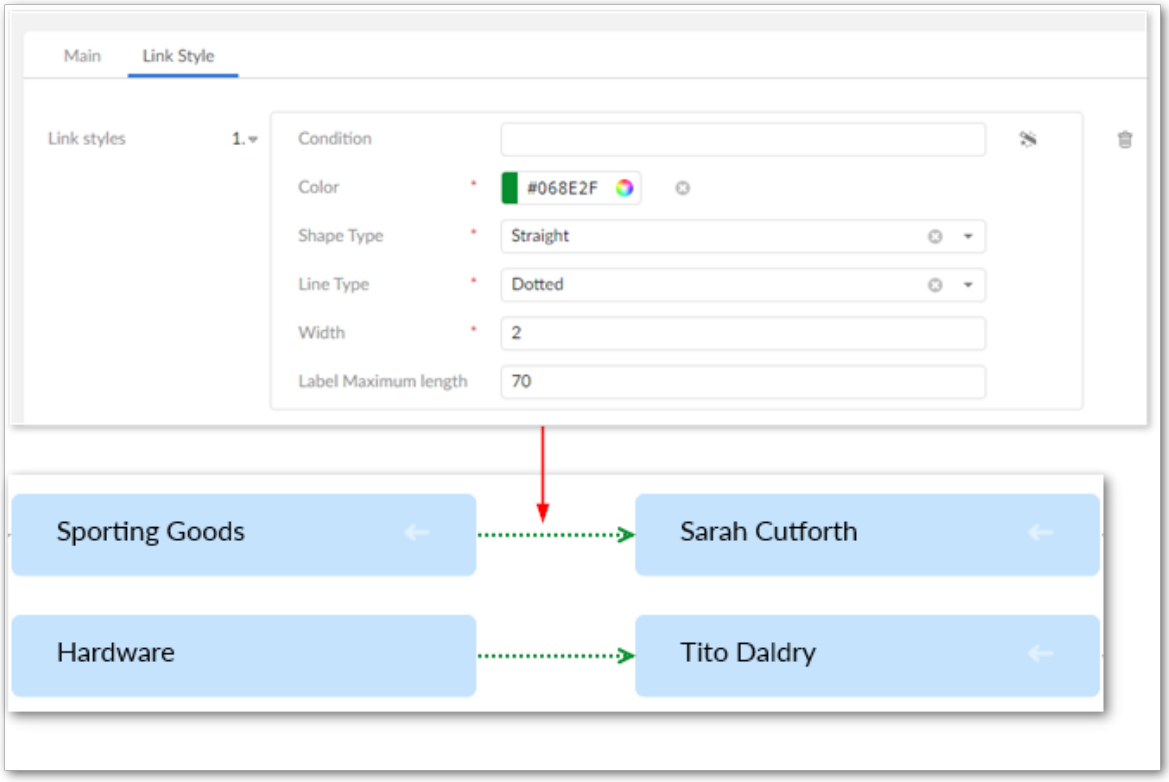
- *Administration > User interface > TIBCO EBX® Data Model and Data Visualization Add-on > Value and relationship graphs > Table configuration > Node Style*. Use the settings in the **Default style** group to determine how nodes display in normal conditions. You can specify that any of these options change display based on a condition. To do this, select the plus icon under **Conditional styles** and fill in the required fields.

In the image below, node color changes for employees that are not active. Also, you can specify a label that will be used in the table legend to differentiate these nodes.



- *Administration > User interface > TIBCO EBX® Data Model and Data Visualization Add-on > Value and relationship graphs > Link configuration > Link Style*. Click the plus icon to edit

link styles via the UI. Alternatively, use the **Programmatic Style** field to enter the fully qualified name to a Java class that defines link styles. The following image shows a customized link style.



CHAPTER 13

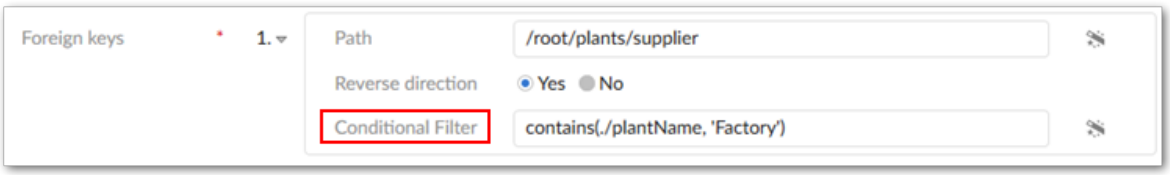
Filtering data values

You can filter values and relationships from data value graphs using an XPath expression, or a programmatic filter implemented in a Java class. The points below describe behavior, advantages and drawbacks to each:

- **Programmatic filter:** When a programmatic filter is used, its logic applies to all foreign keys defined in the link configuration. These types of filters can be very powerful as they allow you to filter links based on individual start and end nodes, any related nodes, or nodes between the start and end node path. The difficulty in using a programmatic filter mainly lies in its implementation, as this requires knowledge of Java. These types of filters can also negatively impact performance in large graphs.
- **XPath filter:** An XPath filter can be configured and applied to individual foreign keys in a link configuration. These types of filters provide the convenience of configuration via the UI and not requiring programming knowledge to configure. However, there are limits to the available functions and operators.

The following demonstrates how to add each type of filter to a link configuration and the outcome of doing so:

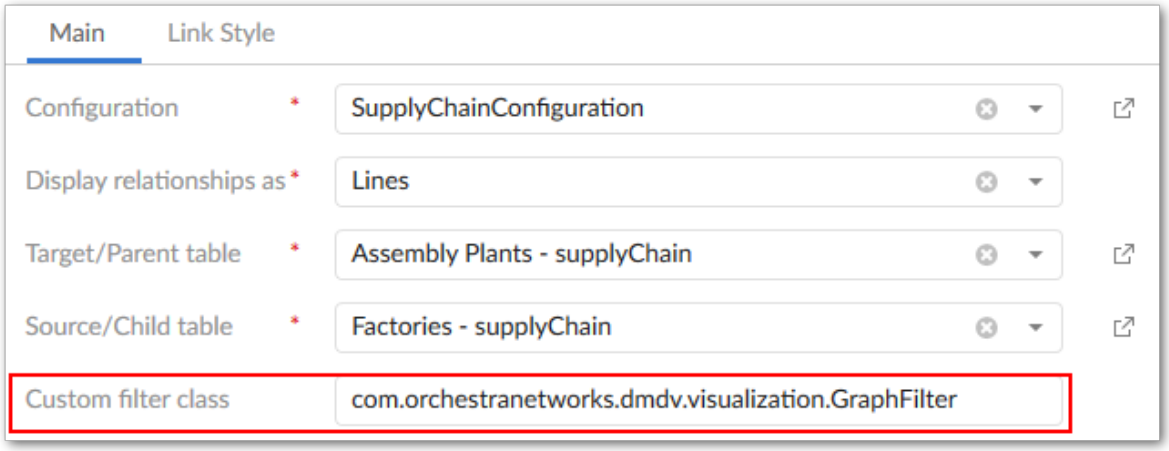
- Add an XPath expression to a link configuration's **Path** property. If conditions in a generated graph satisfy the expression, the add-on removes the corresponding nodes. As shown in the following image, you can use the **Conditional Filter** property to define an XPath filter:



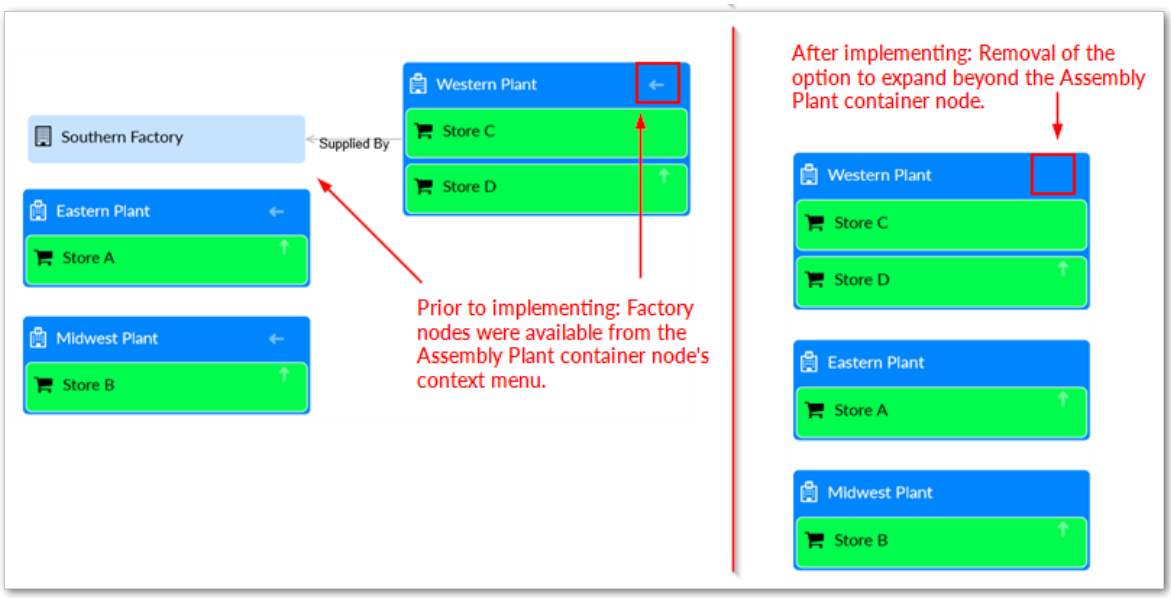
> **Note**
>
> The add-on only supports XPath for fields on the target table, or a combination of the target and current tables. You cannot use the current table only. Additionally, the add-on validates the XPath statement when saving the record and blocks the save action if the statement is invalid.

- Add a programmatic filter that applies to all paths defined in a link configuration. See the Filtering data values and relationships [p 110] section for sample code to help you get started. Once you've

implemented the custom filter class, enter the fully qualified name in the **Custom filter class** field (shown below).



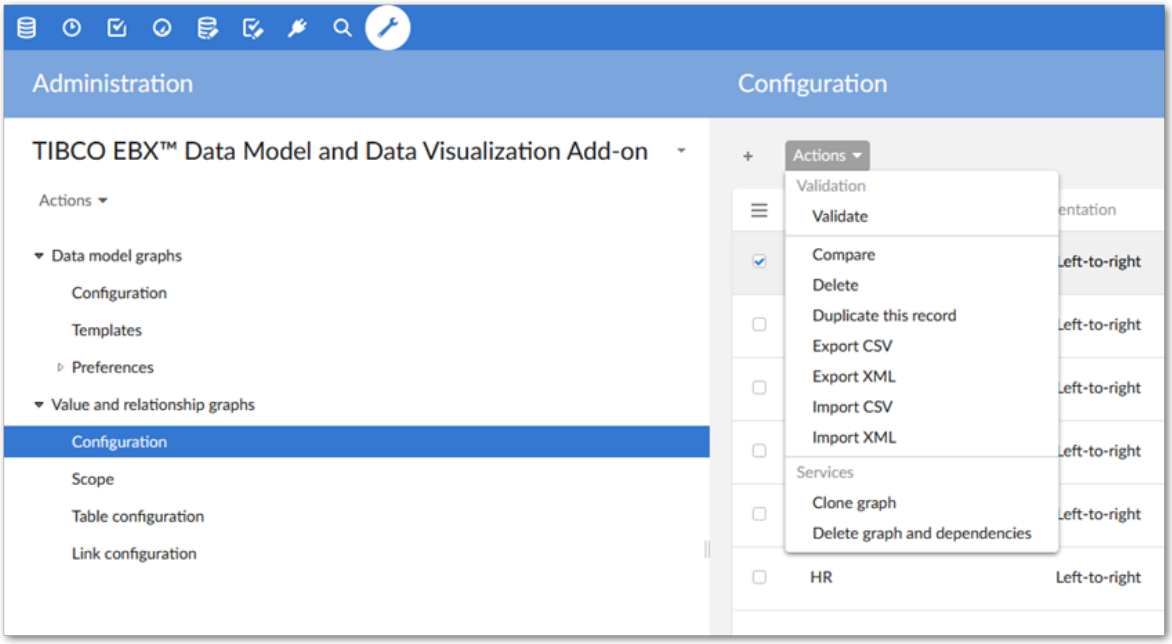
For demonstration purposes, both filters described above remove the same values from the graph. The image below shows the result of applying the filters:



CHAPTER 14

Cloning and deleting graph configurations

The add-on allows you to duplicate and delete graph configurations using the **Clone graph** and **Delete graph and dependencies** services, respectively. The add-on prepends (Copy) to the label of duplicated graphs. When deleting graphs, all related table and link configurations will be deleted.



See also

Configuring graphs [p 49]

Generating data value graphs [p 65]

CHAPTER 15

Generating data value graphs

This chapter contains the following topics:

15.1 Overview

You can generate data value and relationship graphs at anytime using default add-on functionality, or you can generate custom graphs after an administrator has created a configuration within the add-on that determines how graph elements display. When tables have multiple configurations, you may select the configuration to load.

15.2 Generating data value graphs

To generate a graph:

1. Navigate to a table containing the data you want to view in a graph.
2. Select one or more records. Each selection will display in the graph as a node.
3. From the table's **Actions** menu, select *Visualization > Display selected data.*

15.3 Generating configured data value graphs

To generate a graph using a custom configuration:

1. Navigate to a table associated with a data value graph configuration.
2. Select one or more records. Each selection will display in the graph as a node.
3. From the table's **Actions** menu, select *Visualization > Display data using configuration.*
4. If multiple configurations exist, select the desired option and click **Generate**.

15.4 Permissions

The add-on hides any graph nodes that correspond to tables or records for which you do not have sufficient permission to view. This behavior applies to linked nodes. For example, a graph

configuration may link Node A to Node C via Node B. After expanding A, you cannot view C without sufficient permission on B.

See also

Configuring graphs *[p 49]*

Cloning and deleting graph configurations *[p 63]*

CHAPTER 16

Enabling tab display

This chapter contains the following topics:

16.1 Overview

Administrators can enable display of default data value graphs in tabs on EBX® records. When a user opens a record, they are not required to run a service to display a graph. They can simply select the tab and display a graph with the current record in focus.

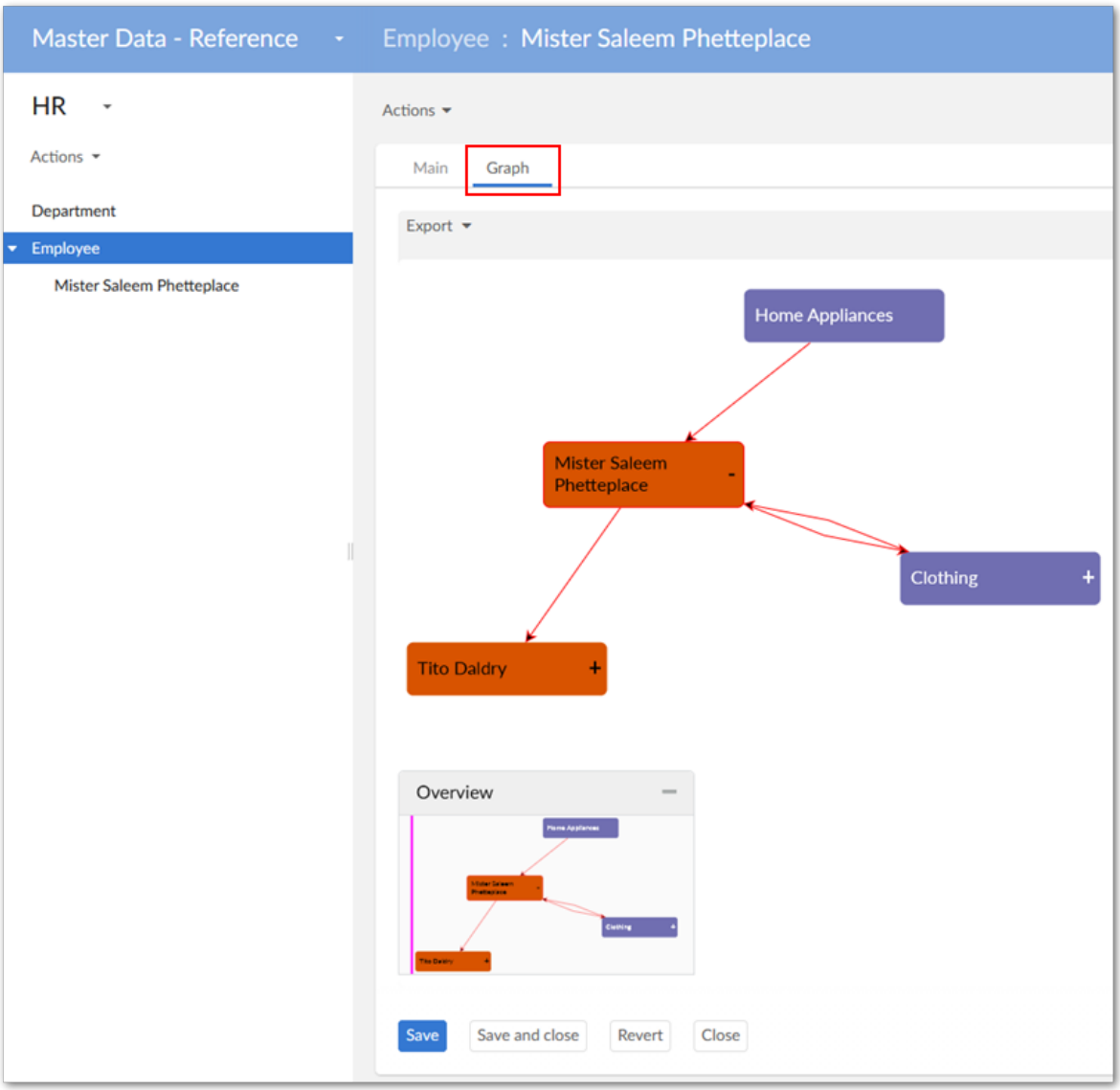
16.2 Configuring tab display

To display a default data value and relationship graph in a record view tab:

1. Open the desired data model in the EBX® Data Modeler Assistant (DMA).
2. Navigate to *Configuration* > *Component library*, create a new record, and enter the following:
   - A name.
   - In the **Configuration** field, enter `com.orchestranetworks.addon.dmdv.data.ui.GraphDataWithoutConfigUIWidgetFactory` and select **Save**.
   - On the **Parameters** tab enable **widgetKey** and save your progress.
3. In the **Navigation pane**, select the table where you want to add the tab and complete the following:
   - Select the table's **Advanced properties** tab.
   - Navigate to *Table* > *Presentation* > *Default rendering for groups in forms* > *Enable rendering* and select the **Enable tabs** option.
   - Save your progress
4. Add a **Group** child element to this table. Optionally, you can set the **Minimum number of values** option to 0. Setting this option prevents validation messages from displaying.
5. Select the **Advanced properties** tab and in the **Default view and tools** group specify the following:
   - Under **Rendering in forms**, enable the **As tab** option.

- Under **Widget**, use the **Component** tab's drop-down menu to select the widget component created in step 2.
- Under **Widget**, use the **Parameters** tab to enter the following **widgetKey** parameter: `com.orchestranetworks.addon.dmdv.data.ui.GraphDataWithoutConfigTabUIWidget`
- Set the **Access properties** option to **Read only**.

6. Save your progress and publish the data model. As shown below, the tab now displays when opening the record.



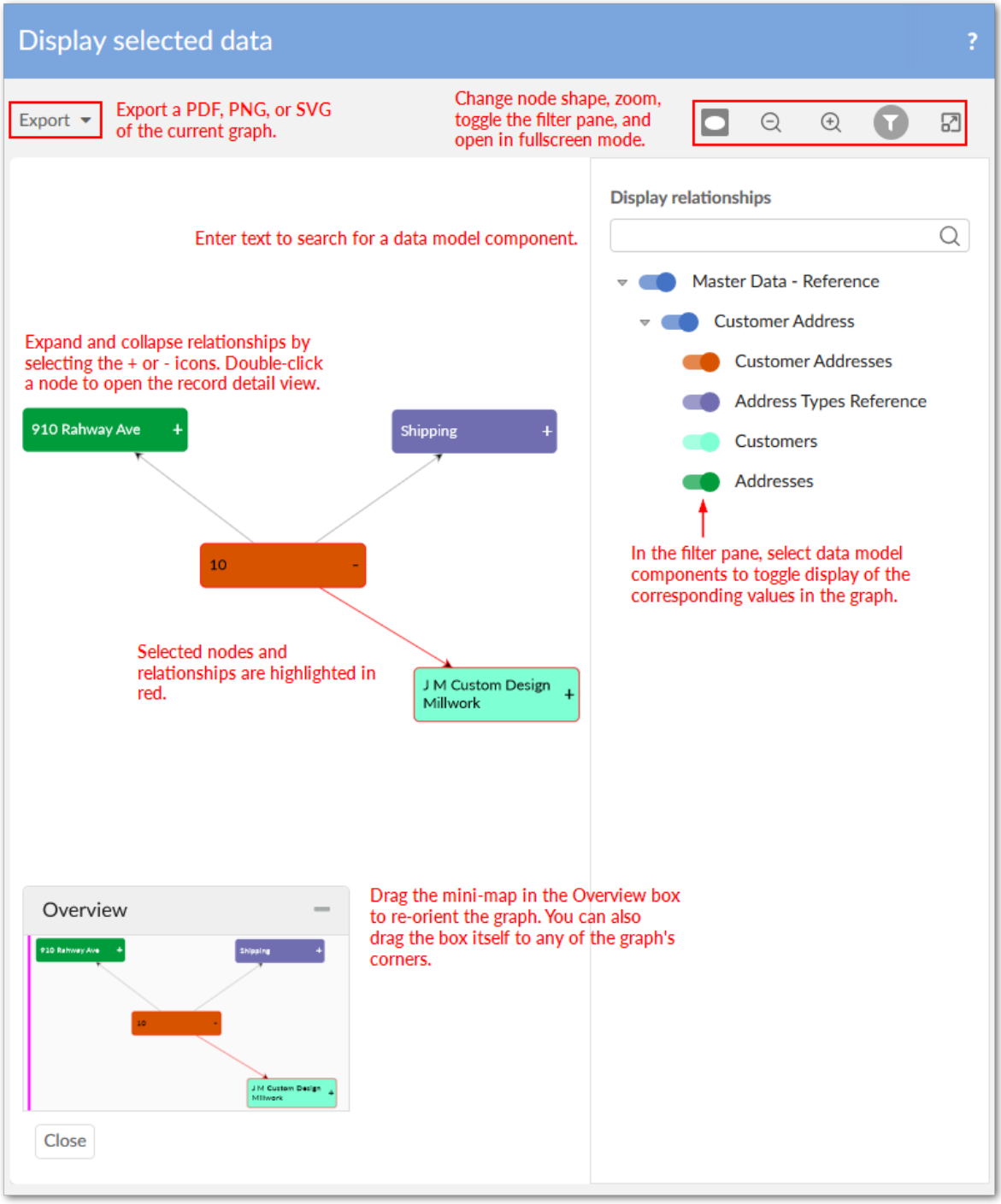
CHAPTER 17

Using data value graphs

When you generate a data value graph using default functionality—a graph not tied to a custom configuration—the add-on displays the selected values as nodes. From the graph, you can:

- Export a PDF, PNG, or SVG of the graph.
- Change the node shape.
- Expand the graph to fullscreen, zoom to change the magnification level.
- Click and drag a node to change its position or graph background to change its orientation.
- Expand/collapse nodes and view node details. To view node details, double-click a node.
- Filter the graph content to display only the data and relationships you want.
- Use the **Overview** mini-map to re-orient the graph. Additionally, you can move the box to a different corner of the graph by selecting its title bar and dragging.

The following image highlights the available features when viewing a data value graph:


Display selected data
Export
Export a PDF, PNG, or SVG of the current graph.
Change node shape, zoom, toggle the filter pane, and open in fullscreen mode.
Enter text to search for a data model component.
Expand and collapse relationships by selecting the + or - icons. Double-click a node to open the record detail view.
910 Rahway Ave
Shipping
10
J M Custom Design Millwork
Selected nodes and relationships are highlighted in red.
Display relationships
Master Data - Reference
Customer Address
Customer Addresses
Address Types Reference
Customers
Addresses
In the filter pane, select data model components to toggle display of the corresponding values in the graph.
Overview
Drag the mini-map in the Overview box to re-orient the graph. You can also drag the box itself to any of the graph's corners.
Close

CHAPTER 18

Using configured data value graphs

This chapter contains the following topics:

18.1 Overview

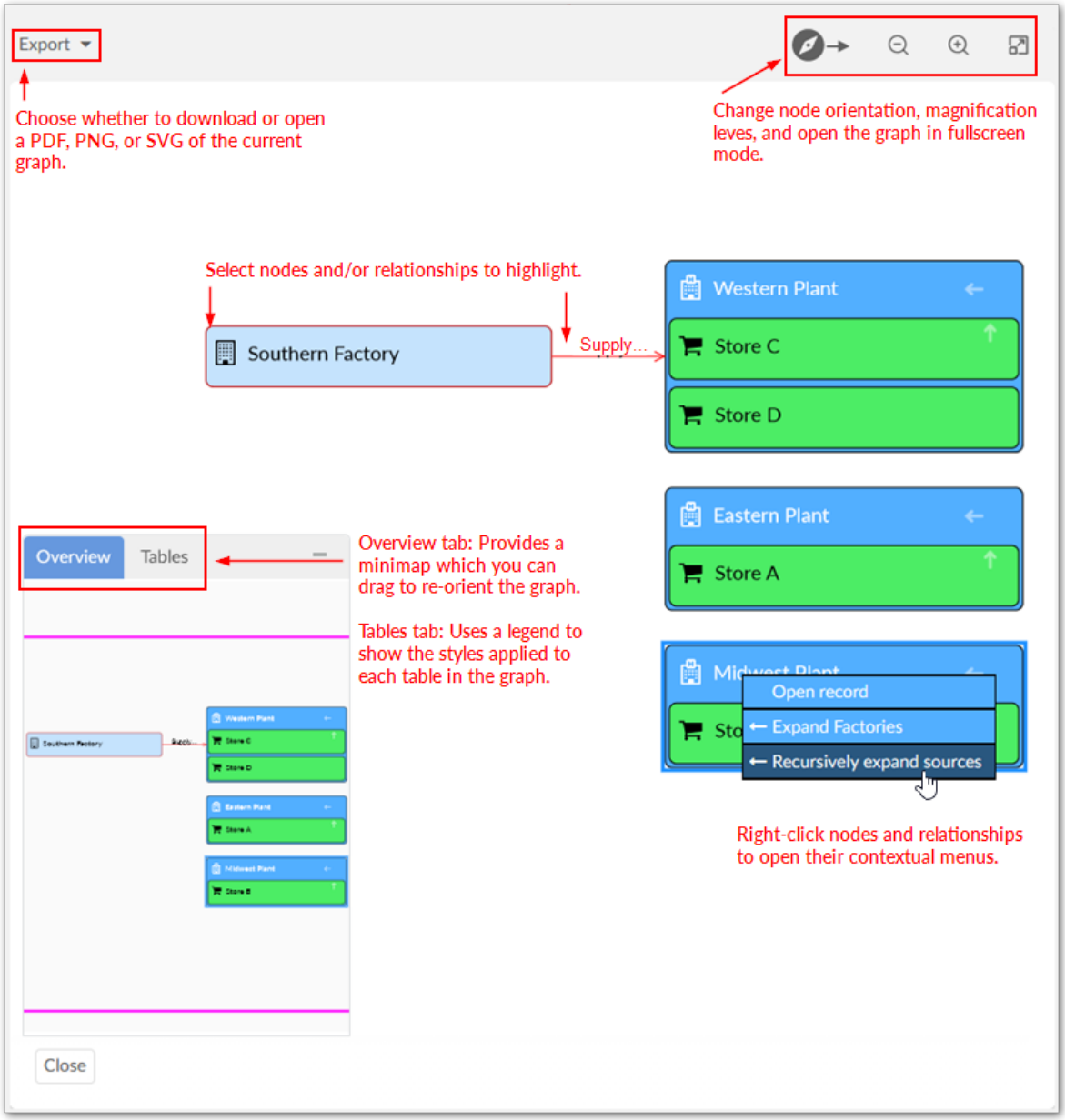
You can generate data value and relationship graphs after an administrator has created a configuration within the add-on that determines how graph elements display. When tables have multiple configurations, you may select the configuration to load.

18.2 Using data value graphs

When viewing graphs, you can:

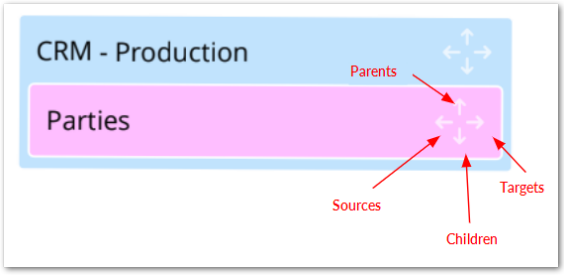
- Export a PDF, PNG, or SVG of the graph.
- Expand the graph to fullscreen and zoom to change the magnification level.
- Choose the orientation of the graph's nodes that use line relationships. The options for node orientation tell the add-on to put the source nodes in the selected position. Then it renders related nodes in the selected direction. For example, the **Top-down** option adds source nodes to the top of the graph and renders related nodes below.
- Click and drag to change the orientation.
- Expand/collapse nodes and view node and relationship details. You can access this option by right clicking on nodes. Depending on the available relationships and configuration, you can expand one node at-a-time, or expand all related nodes.
- Use the **Overview** mini-map and **Tables** tab to re-orient the graph and see how styles are applied to tables, respectively. Additionally, you can move the box to a different corner of the graph by selecting its title bar and dragging.

The following image and table highlight the available features when viewing a data value graph:



> **Note**
>
> It is not shown in the image above, but a double arrow displays to indicate nodes are linked multiple times using the same link.

An arrow icon on a node indicates the availability and display type of related nodes. Administrators set display types during configuration. As shown in the following image, a given node can hold up to four relationship types:



Context menus

When you right-click a node or relationship, its related context menu displays and allows you to:

- Expand any related nodes configured to display in this graph. When a node holds a relationship type to multiple nodes, you can expand them recursively in a given direction. For example, you might see the option to expand all of a node's targets. Nodes can expand recursively up to five levels deep.
- Open the selected record, or show the link details.

CHAPTER 19

Graphs in perspectives and workflows

This chapter contains the following topics:

19.1 Overview

Administrators can configure EBX® perspectives and workflows to display data model and data value graphs. See the EBX® product documentation for more information on creating perspectives and workflows.

19.2 Adding graphs to perspectives

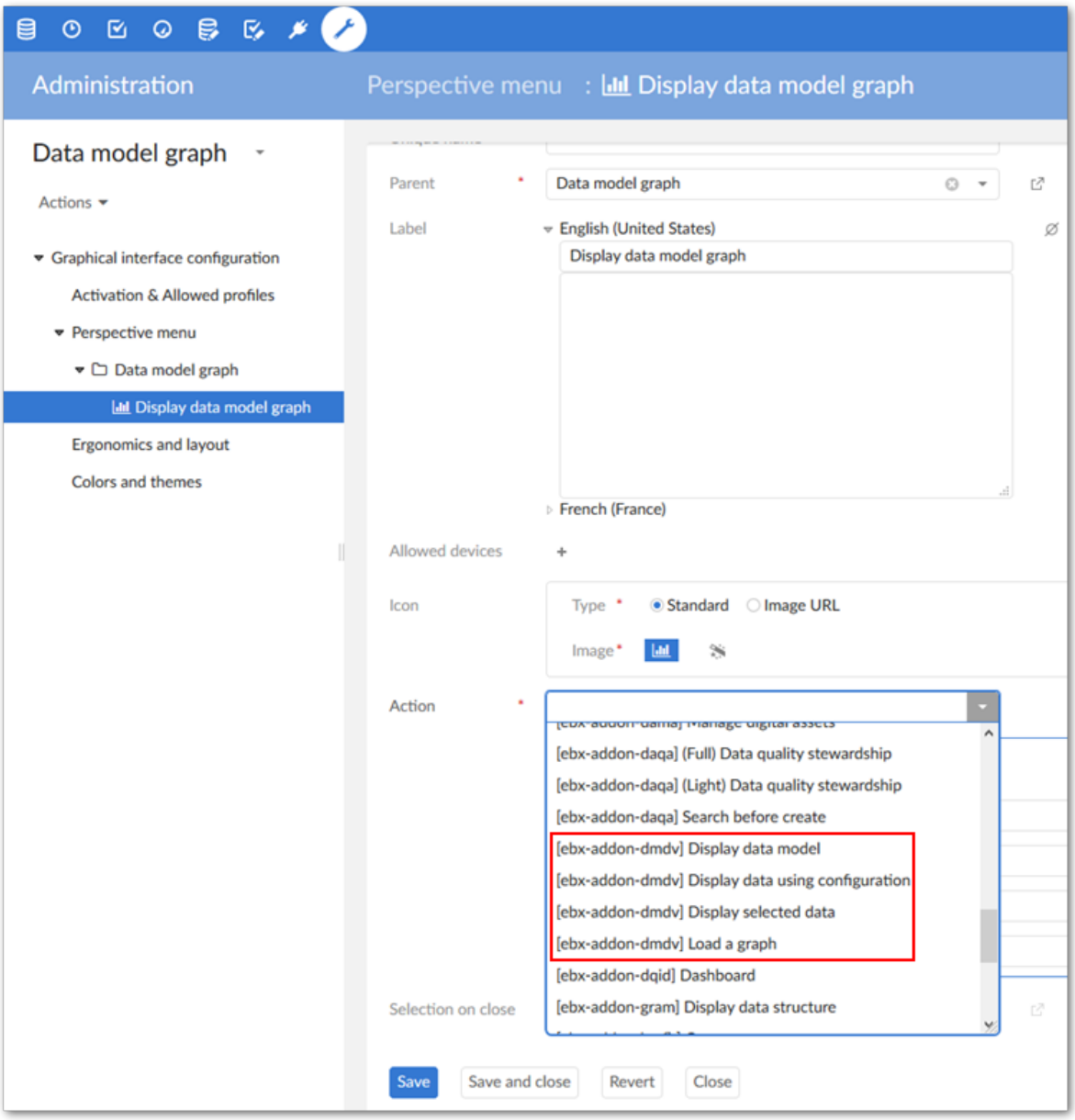
When creating or editing a perspective, you can use the **Action** menu to access the following add-on functionality:

- **Display data model**: displays the data model specified in the **Input parameters** group. The graph uses any existing configuration settings for display. If no settings exist, the default add-on settings apply.
- **Display data using configuration**: displays a configured data value graph. Use the service's **Input parameters** group to enter the dataspace, dataset, and records to display in the graph. The **Expand by default** parameter determines whether the add-on automatically expands the graph's nodes on open. To enable automatic expansion, enter a value of `true`. Leave the field empty or enter `false` to disable the feature.

  Using the **Graph configuration** parameter, you can specify which graph configuration the add-on uses to generate the graph. If you leave the parameter blank:

  - And the selected records are only included in one graph configuration, the add-on automatically uses that configuration.
  - And multiple configurations are linked to the selected records, the add-on will allow the user to choose the configuration to use.

- **Display selected data**: displays a data value graph. Use the service's **Input parameters** group to enter the dataspace, dataset, and records to display in the graph.
- **Load a graph**: allows the user to choose a saved graph to load.



See also

Interacting with data model graphs *[p 33]*

Generating and loading data model graphs *[p 28]*

19.3 Accessing graphs in workflows

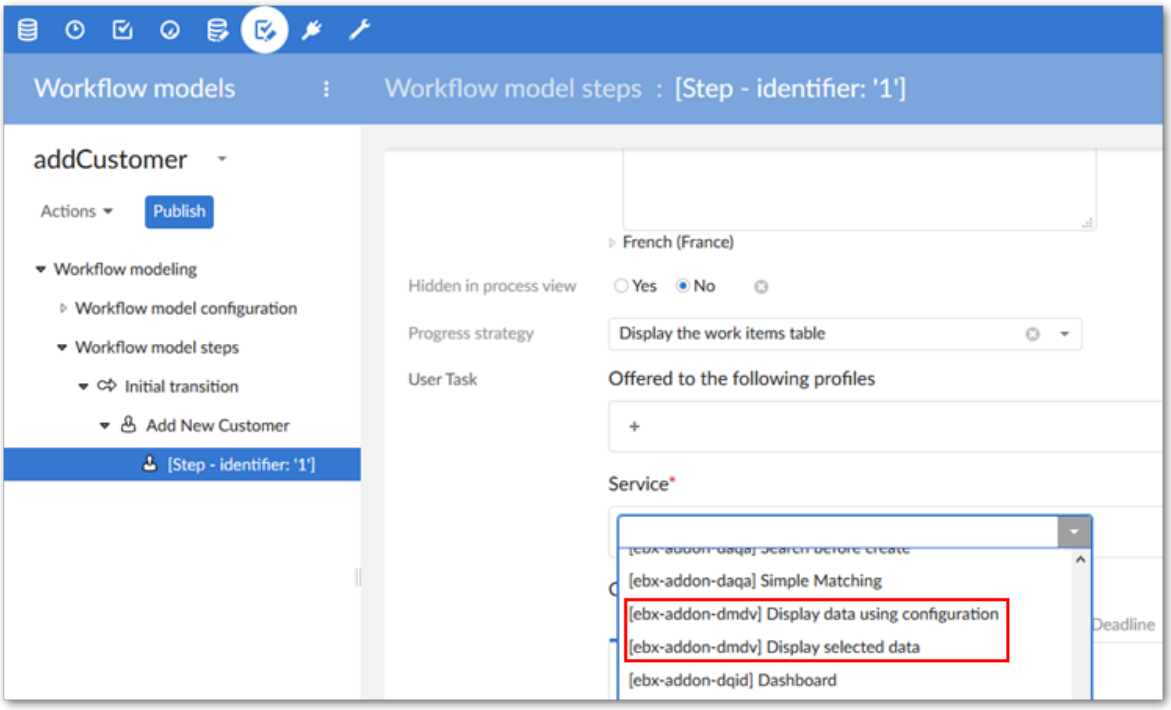
When creating or editing a workflow step you can apply the following add-on services:

- **Display data using configuration**: displays a configured data value graph. Use the service's **Input parameters** group to enter the dataspace, dataset, and records to display in the graph. Using

the **Graph configuration** parameter, you can specify which graph configuration the add-on uses to generate the graph. If you leave the parameter blank:

- And the selected records are only included in one graph configuration, the add-on automatically uses that configuration.
- And multiple configurations are linked to the selected records, the add-on will allow the user to choose the configuration to use.

- **Display selected data**: displays a data value graph. Use the service's **Input parameters** group to enter the dataspace, dataset, and records to display in the graph.



CHAPTER 20

Appendix

This chapter contains the following topics:

20.1 Overview

This appendix contains the file content to follow along with the sample tutorial. The Prerequisites and setup [p 15] chapter provides instructions for setting up your environment to match the sample. Use the links below to jump to the corresponding content:

Data model

The following contains the example code for the sample tutorial data model. Follow this link to return to the Prerequisites and setup [p 15] instructions.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--XML schema generated from EBX™5 DMA instance [reference=basicSupplyChain] on Tue Jun 19 17:23:43 MDT 2018 by
 user [admin].-->
<xs:schema xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0" xmlns:ebxbnd="urn:ebx-
schemas:binding_1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ebxs="urn:ebx-schemas:session_1.0">
<xs:import namespace="urn:ebx-schemas:common_1.0" schemaLocation="http://schema.orchestranetworks.com/
common_1.0.xsd"/>
<xs:import namespace="urn:ebx-schemas:session_1.0" schemaLocation="http://schema.orchestranetworks.com/
session_1.0.xsd"/>
<xs:element name="root" osd:access="--">
<xs:complexType>
<xs:sequence>
<xs:element name="factories" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation xml:lang="en-US">
<osd:label>Factories</osd:label>
</xs:documentation>
<xs:appinfo>
<osd:table>
<primaryKeys>/factory </primaryKeys>
</osd:table>
</xs:appinfo>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element name="factory" type="xs:string" minOccurs="1" maxOccurs="1">
<xs:annotation>
<xs:documentation xml:lang="en-US">
<osd:label>Factory Name</osd:label>
</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
```

```
</xs:complexType>
</xs:element>
<xs:element name="plants" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation xml:lang="en-US">
<osd:label>Assembly Plants</osd:label>
</xs:documentation>
<xs:appinfo>
<osd:table>
<primaryKeys>/plantName </primaryKeys>
</osd:table>
</xs:appinfo>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element name="plantName" type="xs:string" minOccurs="1" maxOccurs="1">
<xs:annotation>
<xs:documentation xml:lang="en-US">
<osd:label>Plant Name</osd:label>
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="supplier" type="xs:string" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation xml:lang="en-US">
<osd:label>Supplier</osd:label>
</xs:documentation>
<xs:appinfo>
<osd:otherFacets>
<osd:tableRef>
<tablePath>/root/factories</tablePath>
</osd:tableRef>
</osd:otherFacets>
</xs:appinfo>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="store" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation xml:lang="en-US">
<osd:label>Stores</osd:label>
</xs:documentation>
<xs:appinfo>
<osd:table>
<primaryKeys>/storeName </primaryKeys>
</osd:table>
</xs:appinfo>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element name="storeName" type="xs:string" minOccurs="1" maxOccurs="1">
<xs:annotation>
<xs:documentation xml:lang="en-US">
<osd:label>Store Name</osd:label>
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="supplier" type="xs:string" minOccurs="0" maxOccurs="1">
<xs:annotation>
<xs:documentation xml:lang="en-US">
<osd:label>Supplier</osd:label>
</xs:documentation>
<xs:appinfo>
<osd:otherFacets>
<osd:tableRef>
<tablePath>/root/plants</tablePath>
</osd:tableRef>
</osd:otherFacets>
</xs:appinfo>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Follow this link to return to the Prerequisites and setup [p 15] instructions.

Table data

The sample table data below is for the Assembly Plants table.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--XML content generated for /root/plants in data set basicSupplyChain in data space Reference on
 2018-07-31T07:38:51.649 by user [admin].-->
<root>
 <plants>
  <plantName>Eastern Plant</plantName>
  <supplier>Northern Factory</supplier>
 </plants>
 <plants>
  <plantName>Midwest Plant</plantName>
  <supplier>Northern Factory</supplier>
 </plants>
 <plants>
  <plantName>Western Plant</plantName>
  <supplier>Southern Factory</supplier>
 </plants>
</root>
```

The sample table data below is for the Factories table.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--XML content generated for /root/factories in data set basicSupplyChain in data space Reference on
 2018-07-31T07:39:18.494 by user [admin].-->
<root>
 <factories>
  <factory>Northern Factory</factory>
 </factories>
 <factories>
  <factory>Southern Factory</factory>
 </factories>
</root>
```

The sample table data below is for the Stores table.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--XML content generated for /root/store in data set basicSupplyChain in data space Reference on
 2018-07-31T07:39:56.654 by user [admin].-->
<root>
 <store>
  <storeName>Store A</storeName>
  <supplier>Eastern Plant</supplier>
 </store>
 <store>
  <storeName>Store B</storeName>
  <supplier>Midwest Plant</supplier>
 </store>
 <store>
  <storeName>Store C</storeName>
  <supplier>Western Plant</supplier>
 </store>
 <store>
  <storeName>Store D</storeName>
  <supplier>Western Plant</supplier>
 </store>
</root>
```

Follow this link to return to the Prerequisites and setup [p 15] instructions.

Reference Guide

CHAPTER 21

Data model graphs configuration group

This chapter contains the following topics:

21.1 Overview

The settings in this group allow you to create a configuration used to display a data model. Additionally, any saved graph configurations are stored here.

21.2 **Configuration table**

Each record in the **Configuration** table specifies options for displaying a visual representation of a data model.

Property	Description
Name	The name used by this configuration.
Graph type	Determines whether this graph will be generated from an TIBCO EBX® data model (Schema), or from another data source (Custom).
Publication mode	When the graph type is set to Schema, use this property to indicate whether the data model is embedded, or in a module. The **Data model** drop-down will then populate with available data models of the corresponding type.
Data model	Specifies the data model the add-on uses to render the graph.
Data source	Displays when graph type is set to **Custom** and must specify the fully qualified path to the Java class that implements this graph.
Display in dataset	Determines whether this configuration is available when users display custom graphs from a dataset's **Actions** menu.
Templates	Specifies the template applied to this configuration. Templates determine a graph's color and display options.
Enable table selection	Determines whether users will have the option of selecting which tables display when generating this graph.

21.3 **Templates table**

You can use the **Templates** table to customize the look and feel of a data model graph.

Property	Description
Name	The name used by this configuration.
Table	Determines border, background, and text color for tables displayed in the graph.
Tooltip	Specifies the background and text color for tooltips.
Link	Sets the color for links between tables and for link labels.
Synchronization	If a data model related to a saved graph has been updated, the add-on can indicate where and what type of changes were made. Updated graphs highlight new items in green, changed items in blue, and deleted items in red.
Highlight color	Determines the color of links and box borders when moused, or hovered over.
Display options	This group allows you to display/hide the following graph components: • Cardinality on links: Toggles display of link cardinality value. • Cardinality on fields: Toggles display of field cardinality value. • Labels on links: Toggles display of foreign key labels. • FK as fields: Toggles display of foreign keys as fields. • Hidden fields: Toggles display of fields that are set as hidden in the DMA.

21.4 **Preferences group**

The **Preferences** group stores information when users save a graph layout. Note that the descriptions for this group are for informational purposes only as records are automatically updated by the add-on after saving a graph. This group includes the **Saved graph** table described in the following section.

Saved graph table

Each record in this table represent a graph that has been saved by a user.

Property	Description
Name	The name used by this saved graph configuration.
Data model	The data model represented by this graph.
User profile	The user profile that created this graph.
Publication mode	Specifies whether the graph was generated from an embedded data model or one from a module.
External model	List of additional data models displayed in this graph.
Graph configuration	The configuration used for this graph.
Shared Policy	List of users or roles that can interact with this graph.
Description	The saved graph's description. Users can provide a description when sharing a graph.

CHAPTER 22

Value and relationship graphs group

This chapter contains the following topics:

22.1 Overview

The **Data configuration** group allows you to define how the add-on renders graphs showing data values. This group contains the following tables:

- Configuration
- Scope
- Table configuration
- Link configuration

22.2 Configuration table

Each configuration specifies general graph behavior. They also act as an anchor point for the settings that determine a graph's scope, displayed tables, and included links. When more than one configuration applies to the same table, the add-on allows users to choose which will display.

Property	Description
Name	The name used by this configuration.
Orientation	Defines the graph's default orientation for relationships displayed using lines. You can choose from the following options: Left-to-right, Right-to-left, Bottom-up, or Top-down.
Allow links to overlap nodes	Determines whether lines drawn for links can overlap nodes. Note that if disabled, this can negatively impact performance.
Highlight color	Sets the color used to highlight relationships and nodes when selected.

22.3 Scope table

The **Scope** table allows you to specify the locations used in graph construction.

Property	Description
Configuration	The configuration to which you want to apply these settings.
Dataspace	Sets the dataspace searched by the add-on when searching for nodes to display in the graph.
Dataset	Sets the dataset(s) searched by the add-on when searching for nodes to display in the graph.

22.4 Table configuration

Each record allows you to associate a table with a data graph configuration. The add-on renders these tables when users generate the specified graph configuration. After adding a table to a configuration, you can configure links to and from the table that determine how graphs display. Two different tabs

display and allow you to configure the main and node style configuration options. The following table provides descriptions for the fields on the **Main** tab:

Property	Description
Configuration	Specifies the configuration to which this table will be added.
Data model	The data model containing the table to include in this configuration.
Table	The table to include in this configuration.
Allow generation from selected table	Indicates whether users can generate the graph configuration from this table's **Actions** menu.
Default label	Text to display on nodes. You can use an XPath expression to define the label. The Default label only applies when the Localized Label and Programmatic Label are undefined.
Localized label	Text to display on nodes according to locales. You can use an XPath expression to define the label. The Localized label only applies if no Programmatic Label is defined.
Programmatic label	Specifies a class to display a custom label on nodes. The value you enter in this field must specify a complete custom class that implements the NodeLabelRenderer interface. For example: `com.orchestranetworks.dmdv.ProgrammaticNodeLabelForItem`

The following table provides descriptions for the options on the **Node style** tab:

Property	Description
Default style group	Defines the default styles for nodes.
Style	Specifies the color for the border, background and text. Additionally, sets the width and style for the border configuration.
Icon	Allows you to associate an icon with the node. By default, the icon displays on the left-hand side of the node. You can use the **Standard** option to select from the icons supplied with the add-on. Use the **Image URL** option to provide a URL to a custom image. The image must be located in the same domain as the running instance of EBX®. Supported filed types include: SVG, PNG, GIF and JPG. Any image larger than 21 by 21 pixels will be automatically resized when displayed in the graph.
Conditional styles group	Defines conditional style for nodes. The add-on applies these styles only certain conditions have been met. You each style component's **Condition** property to set the desired criteria.
Background color	Conditional background color.
Label in legend	Text displayed in the legend for this background color conditional style.
Color	Color of background displayed both in graph and legend if condition is true.
Condition	Condition statement associated with the background color. All of the current table's fields are available through the XPath editor. For example: "./available=true" where "available" is a boolean field of current table)
Label Color	Conditional label color.
Label in legend	Text displayed in the legend for this label color conditional style.
Color	Color of the label displayed both in graph and legend if the condition is true.
Condition	Condition statement associated to the label color. All of the current table's fields are available through the XPath editor. For example: "./available=true" where "available" is a boolean field of current table).
Border Color	Conditional border color.
Label in legend	Text displayed in the legend for this border color conditional style.
Color	Color of the border displayed in the graph and legend if the condition is true.
Condition	Condition statement associated to the border color. All of the current table's fields are available through the XPath editor. For example: "./available=true" where "available" is a boolean field of current table).
Border Style	Conditional border style.

Property	Description
Label in legend	Text displayed in the legend for this conditional border style.
Style	The border style displayed in the graph and legend if the condition is true.
Condition	Condition statement associated to the Border Style. All of the current table's fields are available through the XPath editor. For example: "./available=true" where "available" is a boolean field of current table).
Border Width	Conditional border width.
Label in legend	Text displayed in the legend for this conditional border width conditional style.
Width	The border width displayed in the graph and legend if the condition is true. Borders can be a width of 1 to 4.
Condition	Condition statement associated to the border width. All of the current table's fields are available through the XPath editor. For example: "./available=true" where "available" is a boolean field of current table).
Node template	Specifies a class to customize node appearance. The value you enter in this field must specify a complete custom class that implements the NodeTemplateFactory interface. For example: `com.orchestranetworks.dmdv.NodeTemplateForItem`

22.5 **Link configuration**

This table determines how the add-on displays relationships in the graph.

Property	Description
Configuration	The configuration these link display options apply to.
Display relationships as	Specifies whether the graph displays this relationship using lines to connect nodes, or containers with nested nodes.
Target/Parent table	Sets the table used as the target if the relationship type is set to **Line**, or the parent table if the relationship type is **Container**.
Source/Child table	Sets the table used as the source if the relationship type is set to **Line**, or the child table if the relationship type is **Container**.
Display parent/target nodes automatically	Determines whether the child or source nodes defined in this link configuration automatically expand to display their respective parent and target relationships. When enabled, the add-on automatically expands this relationship when a user opens a graph displaying these child/source nodes, or when a user expands an existing node to display these child/source nodes.
Custom filter class	Specifies a class that implements a custom filter. The value you enter in this field must specify a complete custom class that implements the DisplayFilter interface. For example: `com.orchestranetworks.dmdv.DisplayFilter`
Default label	Text that will be displayed as default. This applies only if no Localized Label and no Programmatic Label are defined.
Localized label	Text that will be displayed according to locales. This applies only if no Programmatic Label is defined.
Programmatic label	Specifies a class used to display a custom label on a link. The value you enter in this field must specify a complete custom class that implements the LinkLabelRenderer interface. For example: `com.orchestranetworks.dmdv.LinkLabelRenderer`
Foreign keys	Each path entry defines a foreign key relationship between the source/target, or parent/child table to display in the graph. This group's fields are described below: • The **Path** field sets the FK's path. Click the wizard icon to display and select the desired FK. • The add-on automatically sets the value of **Reverse direction** when the defined foreign key references another table. If the defined foreign key is self-referencing, this property's setting determines the direction in which the add-on follows the foreign key. The direction determines how the generated graph renders the relationship. • Use the **Conditional Filter** field to define an XPath predicate and apply a specific filter on a link. This limits the number of displayed nodes in the graph.

The following table provides descriptions for the options on the **Link Style** tab:

Action	Description
Link styles group	Contains the link style definition.
Condition	If no condition is defined, the style is considered always valid and will be applied.
Color	Defines the link's color.
Shape Type	Defines the link's shape.
Line Type	Defines the type of line used for the link.
Width	Defines the width of the link. The width can be set from 1 to 4.
Label Maximum Length	Defines the maximum length of the link label in pixels. If the label value is longer than this number, the value is truncated automatically. Hover your cursor on the label to see the entire value.
Programmatic Style	Specifies a class used to display a custom link style. The value you enter in this field must specify a complete custom class that implements the LinkStyleFactory interface. For example: `com.orchestranetworks.dmdv.LinkStyleFactory`

Developer Guide

CHAPTER 23

API Overview

The TIBCO EBX® Data Model and Data Visualization Add-on's API provides extensive access to features and functionality. Using the API, you can generate graphs from sources outside of TIBCO EBX®, customize graph look and feel, provide user access to graphs from custom services, and implement custom filters. The following table shows you where to locate samples that can assist your own implementation:

Topic	Sample information and location
Generating custom data model graphs from external sources and editing templates to customize look and feel.	After an administrator uses the add-on's UI to associate the custom graph with a configuration record, users can select the graph when they run the add-on's **Generate custom graph** service. See Generating a model graph from an external source [p 99] for more details.
Writing a custom service from which users can generate a custom data model graph.	See Generating from a service [p 103] for more details.
Generating an existing data value and relationship graph for display in a UI tab.	When users open a record included in the graph's configuration, they can view the graph by selecting the corresponding tab. See Data value and relationship graph options [p 105] for more details.
Writing a custom service from which users can generate an existing data value and relationship graph.	Users can select the records to display in the graph and run the custom service from the table's **Actions** menu. See Displaying a graph from a custom service [p 108] for more information.
Creating a custom filter for data value and relationship graphs.	The filter can remove nodes that should not display by removing the corresponding relationships. See Filtering data values and relationships [p 110] for more details.
Implementing node templates to customize data value graph nodes.	You can create templates that determine the look and feel of data value graph nodes. The functionality becomes available to users after an administrator links the custom node template implementation with a graph configuration. See Node templates [p 116] for more details.
Writing a service that displays a default data value and relationship graph.	A default data value and relationship graph requires no configuration. Users can select one or more records and generate the graph. This sample code demonstrates how to write your own service to accomplish this and how to make some minor customizations.

CHAPTER 24

Generating a model graph from an external source

This chapter contains the following topics:

24.1 Overview

This section shows how to generate data model graphs from external sources and customize look and feel. You can define all data model graph components in a Java class. This is similar to using the DMA to build out a model, except model components are created using the API. You can allow users access to custom graphs in the following ways:

- By creating a configuration in the add-on that points to the custom graph. This makes the graph selectable when users run the add-on's **Generate custom graph** service.
- By making the graph available from a custom service you write using EBX®'s API.

24.2 Defining a model with CustomGraphModelFactory

You can use a Java class that implements the `CustomGraphModelFactory` interface to create a custom data model. The interface's `build()` returns a `Diagram`, which corresponds to a data model. The getGraphModelTemplate() returns the template to determine look and feel.

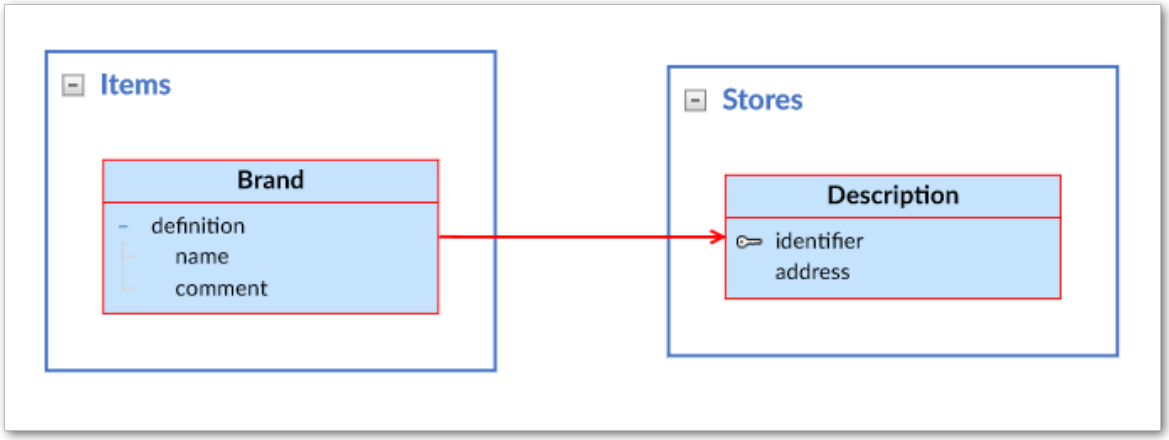
The `DemoCustomGraphModelFactory` class example below defines a model using some basic components. Please refer to the API reference to see how all objects translate to graph components.

> **Note**
>
> Each constructed model must start with a root node. You then build-out the model by defining how components relate.

```
package com.orchestranetworks.addon.test.dmdv.service.model;
```

```
import com.onwbp.base.text.*;
import com.orchestranetworks.addon.dmdv.model.extension.*;

/**
 */
public final class DemoCustomGraphModelFactory implements CustomGraphModelFactory
{
 public Diagram build()
 {
  Diagram diagram = new Diagram();

  // The Diagram must start with a root node.
  DatasetGroup rootNode = diagram.getRootNode();

  // Add model components relative to the root node. The following adds a Stores group with one table and two
 fields.
  DatasetGroup storeGroup = rootNode.addGroup("Stores");
  Table storeDescription = storeGroup.addTable("Description");
  storeDescription.addPrimaryKeyField("Identifier");
  storeDescription.addField("Address");
  // The following adds a Items group, a table, and updates the table label. Two of the fields are added to a
 table group. The final field is a FK to the Stores table.
  DatasetGroup itemGroup = rootNode.addGroup("Items");
  Table brandTable = itemGroup.addTable("brand");
  brandTable.setLabel(UserMessage.createInfo("Brand"));
  TableGroup brandTableGroup = brandTable.addGroup("Definition");
  brandTableGroup.addField("Name");
  brandTableGroup.addField("Comment");
  TableField storeForeignKey = brandTableGroup.addField("Store");
  storeForeignKey.setReferenceTable(storeDescription);

  return diagram;
 }

 // To get the default graph template, return a new GraphModelTemplate().
 public GraphModelTemplate getGraphModelTemplate()
 {

  return new GraphModelTemplate();
 }

}
```

When users generate a graph based on the code in the sample, they will see the following:



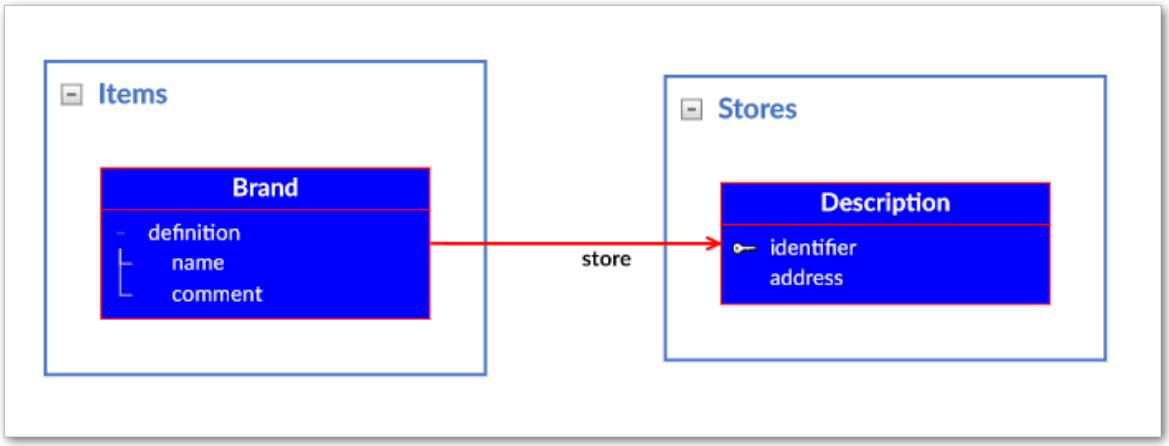
24.3 Customizing look and feel

To customize the look and feel of a data model graph, you can edit `getGraphModelTemplate()`. The following example shows a couple of basic changes to the `DemoCustomGraphModelFactory` class. Refer to the API documentation for a complete list of editable attributes.

```
package com.orchestranetworks.addon.test.dmdv.service.model;

import com.onwbp.base.text.*;
import com.orchestranetworks.addon.dmdv.model.extension.*;

/**
```

```
 */
public final class DemoCustomGraphModelFactory implements CustomGraphModelFactory
{
 public Diagram build()
 {
  Diagram diagram = new Diagram();

  // The Diagram must start with a root node.
  DatasetGroup rootNode = diagram.getRootNode();

  // Add model components relative to the root node. The following adds a Stores group with one table and two
 fields.
  DatasetGroup storeGroup = rootNode.addGroup("Stores");
  Table storeDescription = storeGroup.addTable("Description");
  storeDescription.addPrimaryKeyField("Identifier");
  storeDescription.addField("Address");
  // The following adds a Items group, a table, and updates the table label. Two of the fields are added to a
 table group. The final field is a FK to the Stores table.
  DatasetGroup itemGroup = rootNode.addGroup("Items");
  Table brandTable = itemGroup.addTable("brand");
  brandTable.setLabel(UserMessage.createInfo("Brand"));
  TableGroup brandTableGroup = brandTable.addGroup("Definition");
  brandTableGroup.addField("Name");
  brandTableGroup.addField("Comment");
  TableField storeForeignKey = brandTableGroup.addField("Store");
  storeForeignKey.setReferenceTable(storeDescription);

  return diagram;
 }

 // To get the default graph template, return a new GraphModelTemplate().
 public GraphModelTemplate getGraphModelTemplate()
 {
  DisplayOptions displayOptions = new DisplayOptions();
  displayOptions.setLabelOnLinksDisplayed(true);

  TableStyle tableTemplate = new TableStyle();
  tableTemplate.setBackground("#0706F9");
  tableTemplate.setTextColor("#FFFFFF");

  GraphModelTemplate template = new GraphModelTemplate();
  template.setDisplayOptions(displayOptions);
  template.setTableStyle(tableTemplate);

  return template;
 }

}
```

The following image shows the changes to the graph appearance by editing the template:



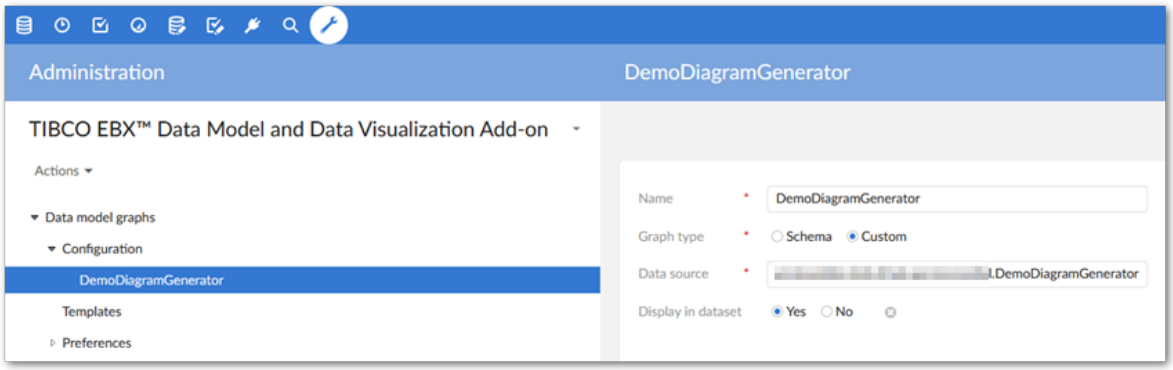
24.4 Including in a graph configuration

Once you have defined a model as described in the previous section, you can include it in a configuration in the add-on. After this users can access it via the **Generate custom graph** option in the UI.

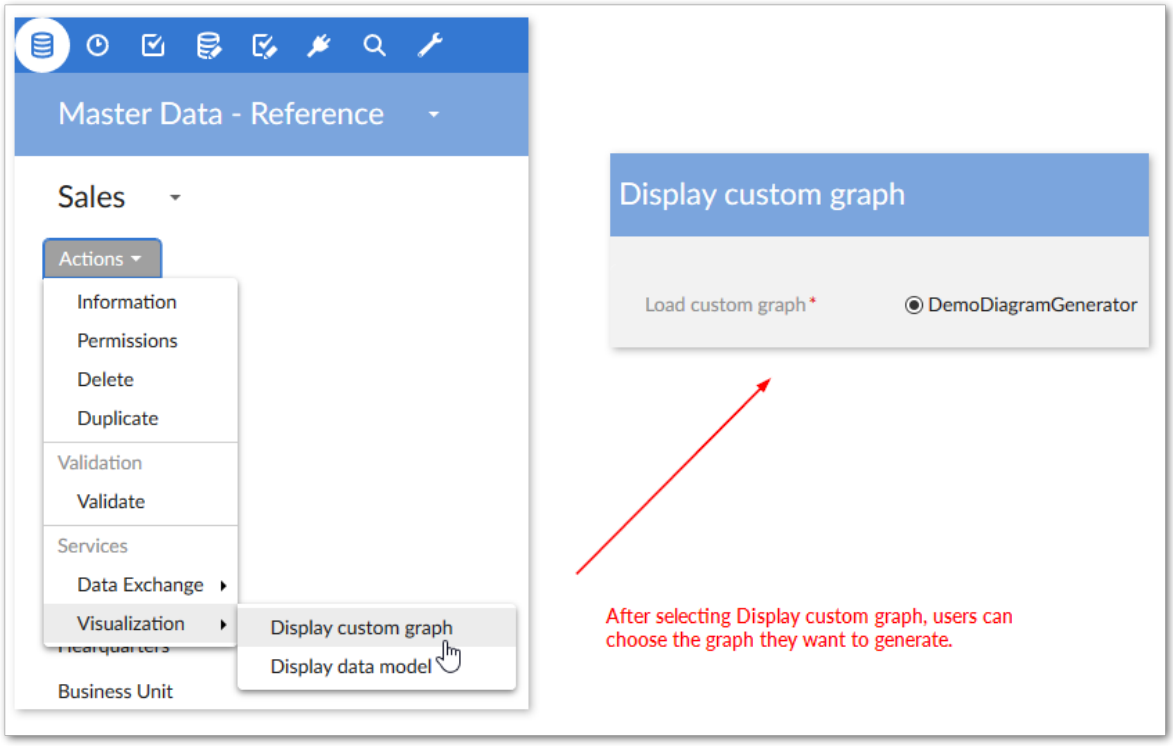
> **Note**
>
> The following steps must be performed by a user with administrative access.

To add the graph to a configuration:

1. Navigate to *Administration* > *User interface* > *TIBCO EBX® Data Model and Data Visualization Add-on* > *Data model graphs* > *Configuration* and create a new record.
2. Supply a name to identify this graph configuration. The add-on displays this name when users run the service.
3. For the **Graph type** option, select **Custom**.
4. In the **Data source** field, enter the fully qualified path to the class that defines the custom model.
5. Set the **Display in dataset** option as desired. This property determines whether users can access this graph from a dataset's **Actions** menu. If you only want this data model graph to be available from a custom service, set to **No**



The following image shows availability of the graph from a dataset's **Actions** menu:



24.5 Generating from a service

You can make a custom graph available from all locations where users can run a service. Once you have defined an external model, use `GraphModelHttpManagerComponentUtils.getComponentForGraphModelService` to get the `UIHttpManagerComponent` that holds the custom graph. For more information on creating user services, refer to the EBX® product documentation.

As shown in the following example, a `customGraphModelFactory` is created based on the model previously defined in `DemoCustomGraphModelFactory`:

```
package com.orchestranetworks.addon.test.dmdv.service.model;
import javax.servlet.http.*;
import com.orchestranetworks.addon.dmdv.model.extension.*;
import com.orchestranetworks.service.*;
import com.orchestranetworks.ui.*;

/**
 */
public final class GraphModelAPI
{
 private final ServiceContext sContext;

 public GraphModelAPI(HttpServletRequest req)
 {
  this.sContext = ServiceContext.getServiceContext(req);
 }

 public void callPage()
 {
  UIServiceComponentWriter writer = this.sContext.getUIComponentWriter();
  writer.add("<div id='GRAPH_MODEL_CONTAINER_TAB_DIV' style='height: 100%;'>");
  writer.add("<iframe id='GRAPH_MODEL_IFRAME' width='100%' height='100%'");

  CustomGraphModelFactory customGraphModelFactory = new DemoCustomGraphModelFactory();

  GraphModelSpec graphModelSpec = new GraphModelSpec(
   customGraphModelFactory,
```

```
    this.sContext.getSession());
   graphModelSpec.setDisplayGraphTitle(true);

   UIHttpManagerComponent comp = GraphModelHttpManagerComponentUtils.getComponentForGraphModelService(
    writer,
    this.sContext.getCurrentAdaptation(),
    graphModelSpec);

   String url = comp.getURIWithParameters();
   writer.add(" frameBorder='0' style='border-width: 0px; ' src='" + url + "'></iframe>");
   writer.add("</div>");
   writer.addJS_cr(
    "var GRAPH_MODEL_CONTAINER_TAB_DIV = document.getElementById('GRAPH_MODEL_CONTAINER_TAB_DIV');");

   writer.addJS_cr("function resizeGraphTabModel(size){");
   {
    writer.addJS_cr("GRAPH_MODEL_CONTAINER_TAB_DIV.style.width = size.w + 'px';");
    writer.addJS_cr("GRAPH_MODEL_CONTAINER_TAB_DIV.style.height = size.h + 'px';");
   }
   writer.addJS_cr("}");
   writer.addJS_addResizeWorkspaceListener("resizeGraphTabModel");

  }

}
```

CHAPTER 25

Data value and relationship graph options

This chapter contains the following topics:

25.1 Overview

The API allows you to:

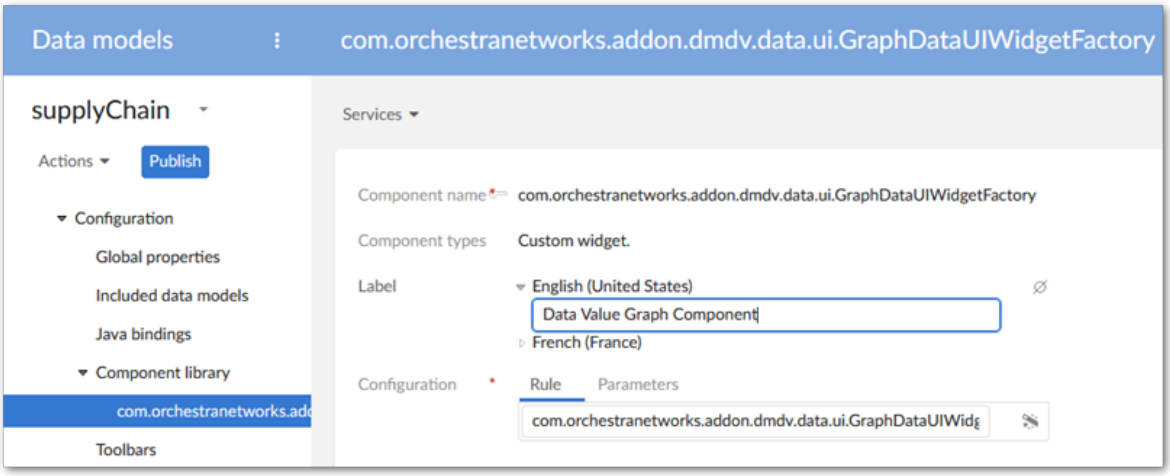
- create a service that displays an existing data value and relationship graph—one already configured using the add-on. You can add the graph to a widget and display it on a tab of an open record. See Displaying a graph in a UI tab [p 105].
- filter which nodes and links display in a data value and relationship graph. See Filtering data values and relationships [p 110] for an example.

25.2 Displaying a graph in a UI tab

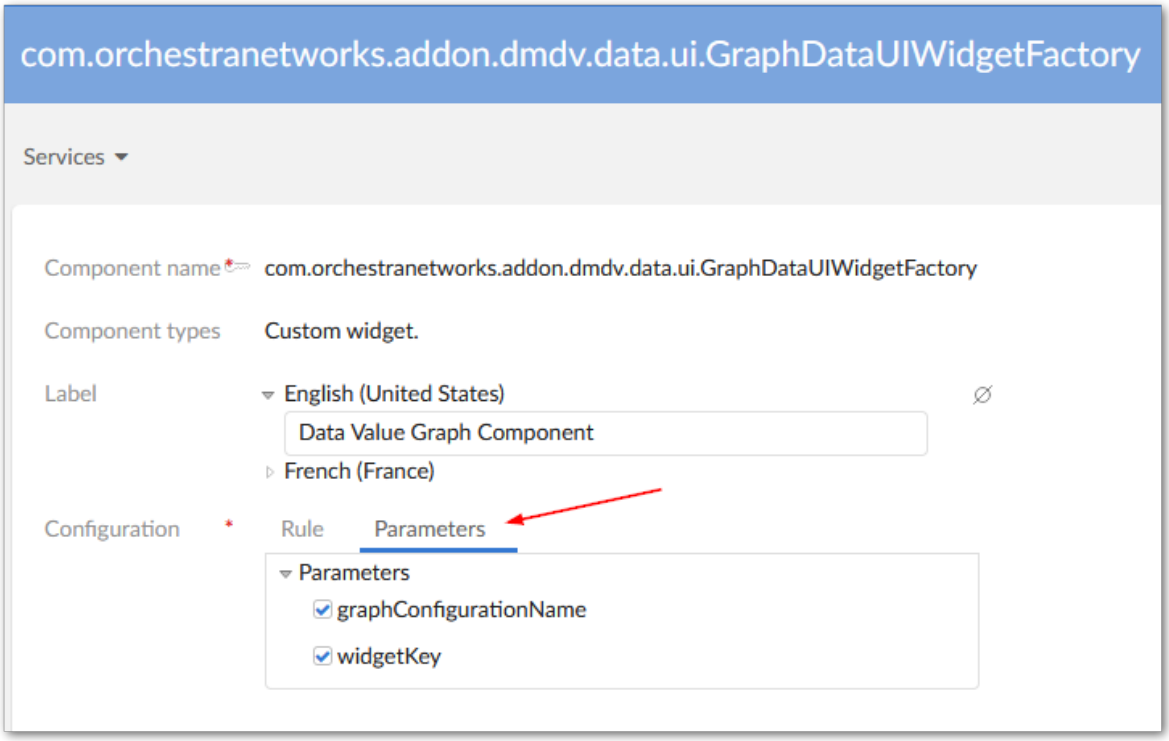
The API provides a sample widget you can use to display an existing data value and relationship graph in a UI tab. When users view a record, they can select the tab that displays the graph. To display a data value graph in a UI tab:

1. Add the EBX® Data Model and Data Visualization Add-on's `com.orchestranetworks.addon.dmdv.data.ui.GraphDataUIWidgetFactory` component to the

desired data model. As shown below, the full name of the component must be included in the **Configuration** field.



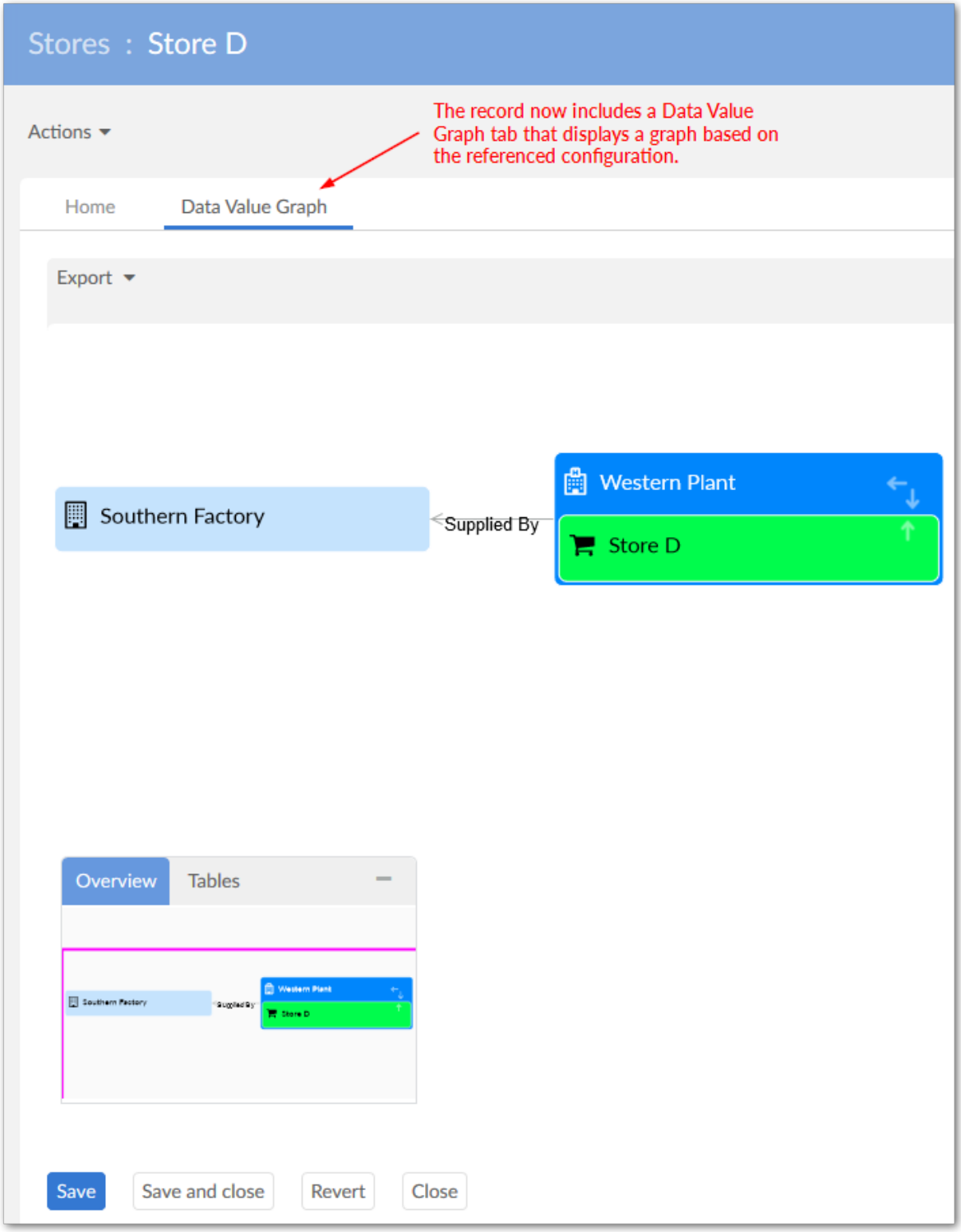
2. Select the **Configuration**'s **Parameters** tab and enable both parameters.



3. In the navigation pane select the table where the tab will be added. On the **Advanced properties** tab, under *Table > Presentation > Default rendering for groups in forms > Enabled rendering*, enable the tabs option. Save your progress.
4. Add a **Group** child element to this table.
5. Set the **Minimum number of values** property to 0. This setting doesn't change the ability to display the graph, but prevents a validation message from displaying.
6. Select the **Advanced properties** tab and under **Default view and tools**:
   - Enable the **As tab** option under **Rendering in forms**.

- In the **Widget** group's **Component** tab select the component created in previous steps.
- Select the **Parameters** tab and enter the configuration name for the graph you want to display. Enter the following in the **widgetKey** parameter: `com.orchestranetworks.addon.dmdv.data.ui.GraphDataTabUIWidget`.
- Set **Access properties** to **Read only**. Save your progress.

7. To test, navigate to the updated table, open a record, and select the new tab.



25.3 Displaying a graph from a custom service

You can write a service from which users can generate an existing data value and relationship graph. As the following sample shows you use `UIHttpManagerComponent.getComponentForGraphDataService`

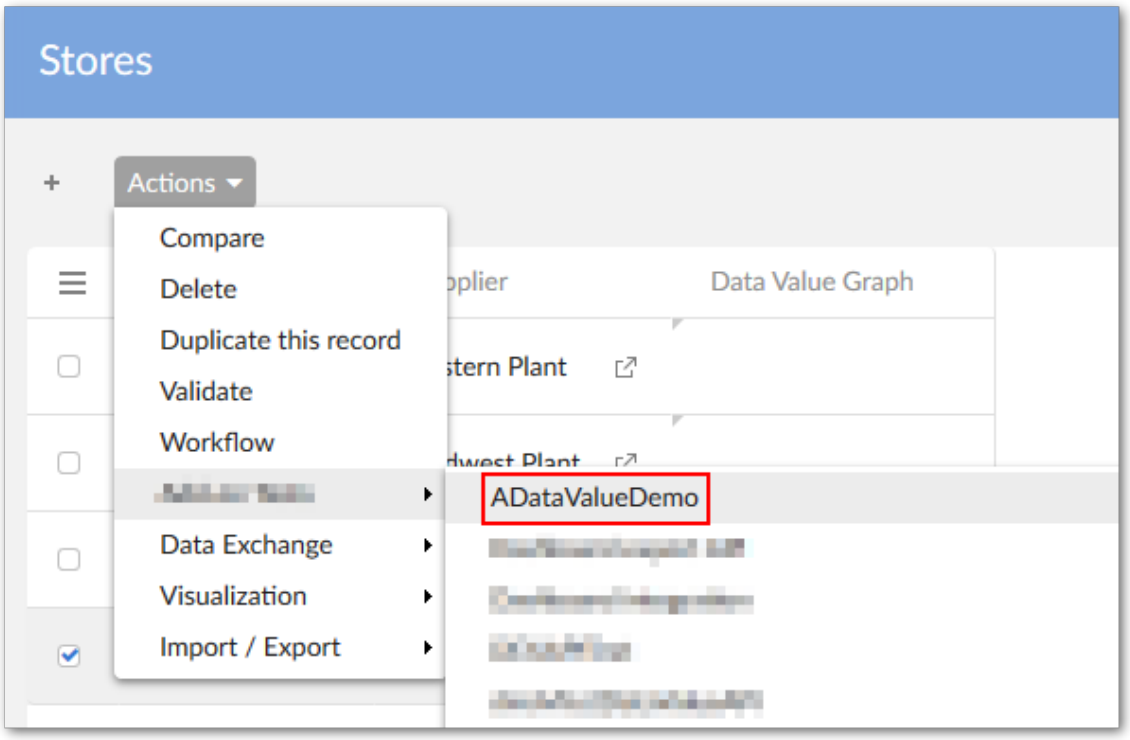
to generate a web component containing a data value graph. The `GraphDataSpec` instance must set the graph name that of an existing value and relationship graph configuration. When you register the service in `module.xml`, its scope must be set to `onRecord`. Currently, the API only supports the selection of one record. For more information on writing services, see the EBX® product documentation.

> **Note**
>
> You can use `GraphDataSpec` to enable or disable specific graph features. The `GraphDataFeatures` class enumerates these features.

```
/*
 * Copyright Orchestra Networks 2000-2008. All rights reserved.
 */
package com.orchestranetworks.addon.dmdv.userservice;

import com.onwbp.adaptation.*;
import com.orchestranetworks.addon.dmdv.data.ui.*;
import com.orchestranetworks.ui.*;
import com.orchestranetworks.ui.selection.*;
import com.orchestranetworks.userservice.*;

/**
 */
public class ADataValueDemo implements UserService<RecordEntitySelection>
{

 @Override
 public void setupObjectContext(
  UserServiceSetupObjectContext<RecordEntitySelection> context,
  UserServiceObjectContextBuilder builder)
 {
 }

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<RecordEntitySelection> context,
  UserServiceDisplayConfigurator config)
 {
  final Adaptation record = context.getEntitySelection().getRecord();

  if (record != null)
  {
   config.setContent(new UserServicePane()
   {
    @Override
    public void writePane(UserServicePaneContext context, UserServicePaneWriter writer)
    {
     String GRAPH_CONFIGURATION_NAME = "SupplyChainConfiguration";
     // Initiate an instance of Graph Data Specification and set the graph name to a working configuration
     GraphDataSpec graphSpec = new GraphDataSpec();
     graphSpec.setGraphConfigurationName(GRAPH_CONFIGURATION_NAME);

     // Prepare the necessary info: record selection and place holder for graph
     writer.add("<div id='" + GRAPH_CONFIGURATION_NAME + "' style='height:100%;'>");
     writer.add(
      "<iframe id='" + GRAPH_CONFIGURATION_NAME
       + "_frame' width='100%' height='100%'");

     // Use GraphDataHttpManagerComponentUtils to generate a web component containing the graph
     UIHttpManagerComponent comp = GraphDataHttpManagerComponentUtils
      .getComponentForGraphDataService(writer, record, graphSpec);

     // Insert graph component URL into the prepared iframe to display
     String url = comp.getURIWithParameters();
     writer.add(
      " frameBorder='0' style='border-width: 0px; ' src='" + url + "'></iframe>");
     writer.add("</div>");
    }
   });
  }
 }

 @Override
 public void validate(UserServiceValidateContext<RecordEntitySelection> context)
 {
 }

 @Override
 public UserServiceEventOutcome processEventOutcome(
```

```
  UserServiceProcessEventOutcomeContext<RecordEntitySelection> context,
  UserServiceEventOutcome eventOutcome)
 {
  return null;
 }
}
```

As shown below, the service is available from the **Actions** menu:



25.4 Filtering data values and relationships

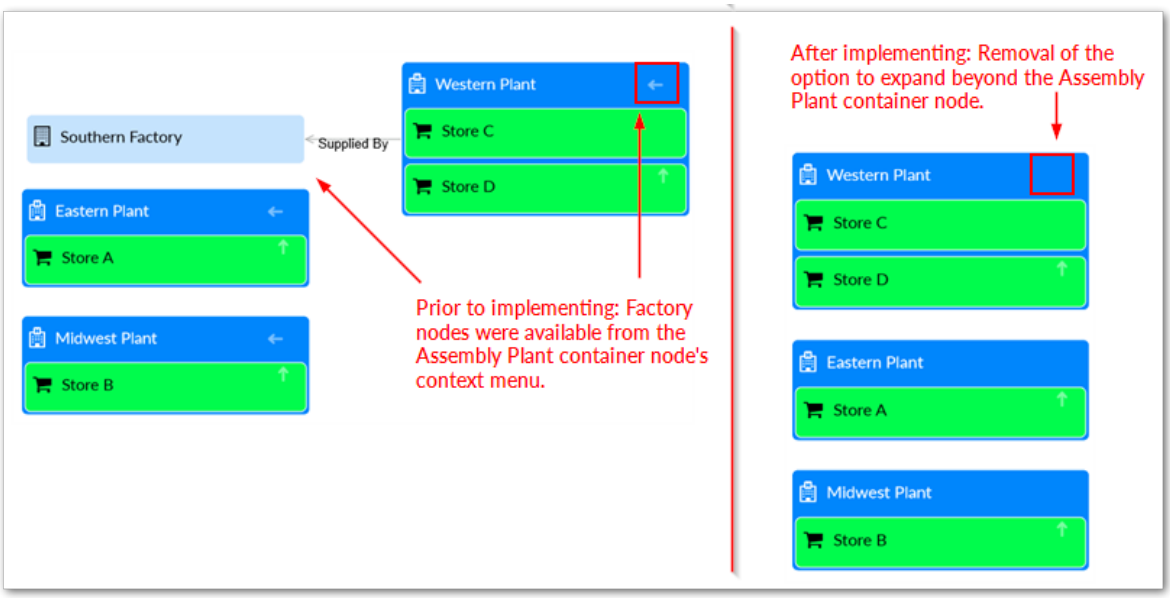
You can use a Java class to filter out values and relationships from a graph. After creating the class, it must be declared in the data value and relationship graph configuration. The following example describes a basic filter. For a more in-depth description, see the API documentation:

1. As shown below the class must implement the `DisplayFilter` interface and basic logic to determine which nodes should be filtered out. The `accept()` is called on each link in the graph during generation. If it returns `True` the link and related nodes display.

```
package com.orchestranetworks.addon.test.dmdv.model;

import com.orchestranetworks.addon.dmdv.data.filter.*;
import com.orchestranetworks.schema.*;

public class FilterNodes implements DisplayFilter
{
 @Override
 public boolean accept(FilterContext context)
 {

  NodeContext factoryNodeContext = context.getLink().getEndNodeContext();
  Node factoryNode = factoryContext.getNode();
  Record factoryRecord = factoryNode.getRecord();
  String factoryName = factoryRecord.get(Path.parse("./factory")).toString();

  return !factoryName.contains("Factory");
 }
}
```

2. Open the **Link configuration** from which you want to filter values. In the **Custom filter class** enter the qualified path to the created filter.

   The following image shows a graph before applying the filter and after:



CHAPTER 26

Displaying a default data value graph

You can write a sample service that opens a default data value and relationship graph based on a user's record selection. Note that default data value graphs require no prior configuration in the UI. The API also allows you to determine display of certain graph features. As shown in the following sample, the **Overview** map has been removed. Additionally, record details are disabled which prevents the end-user from double-clicking a node to open the record.

```
public class DefaultDataValueGraph implements UserService<RecordEntitySelection>
{

 @Override
 public void setupObjectContext(
  UserServiceSetupObjectContext<RecordEntitySelection> context,
  UserServiceObjectContextBuilder builder)
 {
 }

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<RecordEntitySelection> context,
  UserServiceDisplayConfigurator config)

 {

  final Adaptation record = context.getEntitySelection().getRecord();

  if (record != null)
  {
   UIButtonSpecNavigation close = config.newCloseButton();
   close.setDefaultButton(true);

   config.setContent(new UserServicePane()
   {
    @Override
    public void writePane(UserServicePaneContext context, UserServicePaneWriter writer)
    {

     //Disable display of the Overview box
     GraphDataSpec spec = new GraphDataSpec();
     spec.setOverviewDisplay(OverviewDisplayOptions.DISABLE);

     //Prevent users from accessing a detailed view of records
     spec.disableFeatures(GraphDataFeatures.RECORD_DETAILS);

     UIHttpManagerComponent comp = GraphDataHttpManagerComponentUtils
      .getComponentForGraphDataService(writer, record, spec);

     writer.add("<div id='incrementalDataContainerTab' style='height: 100%;'>");
     writer.add("<iframe id='incrementalDataTabIframe' width='100%' height='100%'");

     String url = comp.getURIWithParameters();
     writer.add(
      " frameBorder='0' style='border-width: 0px; ' src='" + url + "'></iframe>");
     writer.add("</div>");
     writer.addJS_cr(
      "var incrementalDataContainerTabElement = document.getElementById('incrementalDataContainerTab');");

     writer.addJS_cr("function resizeIncrementalDataTab(size){");
     {
      writer.addJS_cr(
```

```
        "incrementalDataContainerTabElement.style.width = size.w + 'px';");
       writer.addJS_cr(
        "incrementalDataContainerTabElement.style.height = size.h + 'px';");
      }
      writer.addJS_cr("}");
      writer.addJS_addResizeWorkspaceListener("resizeIncrementalDataTab");
     }
    });
   }
  }

  @Override
  public void validate(UserServiceValidateContext<RecordEntitySelection> context)
  {
  }

  @Override
  public UserServiceEventOutcome processEventOutcome(
   UserServiceProcessEventOutcomeContext<RecordEntitySelection> context,
   UserServiceEventOutcome eventOutcome)
  {
   return null;
  }
}
```

Customizing graph nodes

CHAPTER 27

Node templates

You can use the add-on's API to customize data value graph nodes. The API uses a `NodeTemplate` to define customization options. `NodePanel`s act as containers for the elements that comprise a node. These editable elements include: text, images, color, buttons, and indicators. When adding these elements to a `NodePanel`, you can determine orientation, color, size, etc.

> **Note**
>
> Once you implement a `NodeTemplate`, an administrator must add the fully qualified class name to a data value and relationship graph configuration. See Sample node template and configuration [p 119] for a sample implementation and configuration instructions.

As shown below, a node template must implement the `NodeTemplateFactory` interface. Calling this interface's `build()` method draws the custom node template.

```
public class NodeTemplateForEmployee implements NodeTemplateFactory
{
 public NodeTemplate build()
 {
  // build a node template instance
  return null;
 }
}
```

See the following sections for information on adding elements to a NodeTemplate:

- Node template elements [p 117] describes the elements that comprise a node.
- Sample node template and configuration [p 119] provides a sample implementation and configuration instructions.
- Using a node value renderer [p 123] provides information on the `NodeValueRenderer` class, which allows you to retrieve and pass custom data to nodes.

CHAPTER 28

Node template elements

This chapter contains the following topics:

28.1 Overview

This section provides information on the following `NodeTemplate` elements:

See also

Sample node template and configuration [p 119]

Using a node value renderer [p 123]

28.2 Node Panels

`NodePanels` act as containers for other elements and can nest within each other. The `NodePanel` default constructor centers elements horizontally across the panel. You can also choose to specify a vertical arrangement for a top-to-bottom placing of elements. When you add node elements to a panel, the add-on draws the elements in order of inclusion. This behavior establishes the Z-ordering of the elements.

28.3 Text Blocks

You can use `NodeTextBlock` to display text on a node. The `textAlign` property specifies where the add-on draws the characters horizontally within the text block. The `setBindingText()` method can set text using a:

- Simple String as an argument: `setBindingText("Bob Smith")`.
- Field path to pass data from a specific field: `setBindingText("${/employee_name}")`.
- Custom URL based on node data. To leverage this functionality, you can create a class that implements `NodeValueRenderer` and pass it to `setBindingText(NodeValueRenderer action)`. See Using a node value renderer [p 123] for more information.

28.4 Node Images

You can use the `NodeImage` class to display images in nodes. Its properties allow you to give exact dimensions for the image. Additionally, you can use the `imageStretch` property to automatically fit the image to its boundaries. The `setBindingSource()` method can set the image source in the following ways:

- If the image URL is a simple String, you can pass the string as an argument. For example: `image.setBindingSource("/addon/www/common/images/icons/avatar.jpg")`
- If you want to use an image stored in a media field managed by the TIBCO EBX® Digital Asset Manager Add-on, pass the field path. For example: `image.setBindingSource("${/picture/attachment}")`
- To use a custom URL based on node data, you can create a class that implements `NodeValueRenderer` and pass this information. For example: `image.setBindingSource(new SourcePictureRendererImpForEmployee2())`

28.5 Node Indicator icon and expand button

You can add a node indicator icon by using `NodeIndicator`. Each node template should have only one indicator. The add-on determines display behavior based on availability of node expansion.

Use `NodeExpanderButton` to display a button on the node that collapses and expands given elements. Pass the target using `setTargetElement(NodeElement targetElement)`. If the `targetElement` is a panel, it must not contain the current expand button.

28.6 Data tables

A node can use a table to organize data using rows and columns. You input the number of columns and add cell data on a row by row basis using `addRowData()`. Data can be added as follows:

- A constant String: `addRowData("Name")`
- A field value using its path: `addRowData("${/group/label}"`
- A combination of the above: `addRowData("Name: ${/group/label})`

CHAPTER 29

Sample node template and configuration

This chapter contains the following topics:

29.1 Overview

Once you have implemented a node template, an administrator must include the template in a value and relationship graph configuration. The following section provides a node template code sample and instructions for adding it to an existing configuration.

See also

Node templates [p 116]

Node template elements [p 117]

Using a node value renderer [p 123]

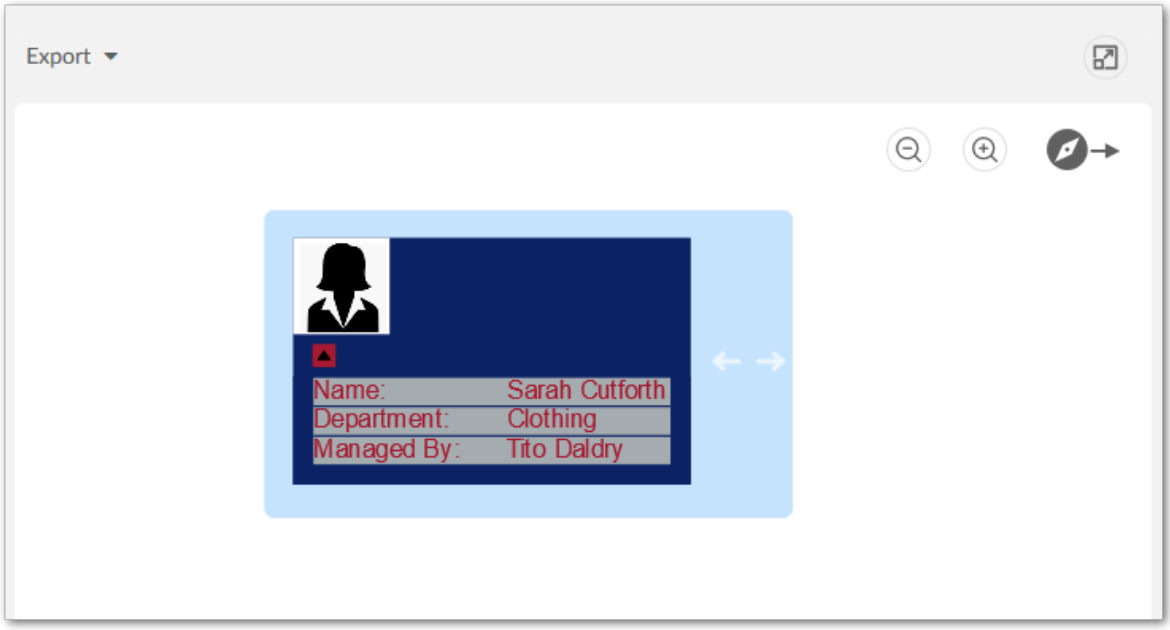
29.2 Node template code sample

The following is a sample code for a node template:

```
public class nodeTemplateExample implements NodeTemplateFactory {

 private static final int DEFAULT_WIDTH = 275;

 public NodeTemplate build() {

  NodeTemplate template = new NodeTemplate();
  template.setShapeType(ShapeType.ROUNDED_RECTANGLE);

  NodePanel rootPanel = new NodePanel();
  rootPanel.setWidth(DEFAULT_WIDTH);

  NodePanel mainPanel = new NodePanel();
  mainPanel.setPanelLayout(PanelLayout.VERTICAL);
  mainPanel.setBackgroundColor("#0B2265");
  mainPanel.setMargin(MarginType.LEFT, 12);
  mainPanel.setMargin(MarginType.TOP, 12);
  mainPanel.setMargin(MarginType.BOTTOM, 12);
  {
   NodePanel imagePanel = new NodePanel();
   imagePanel.setAlignment(AlignmentType.MIDDLE_LEFT);
```

```
   {
    NodeImage image = new NodeImage();
    image.setBindingSource("/icons/w1.png");
    image.setHeight(50);
    image.setWidth(50);

    imagePanel.addElement(image);
   }

   NodePanel table = new NodePanel();
   {
    NodeDataTable dataTable = new NodeDataTable();
    dataTable.setAlignment(AlignmentType.CENTER);
    dataTable.setNumberOfColumns(2);
    dataTable.setColumnsWidth(new int[] { 100, 85 });
    dataTable.setRowSeparatorColor("#0B2265");
    dataTable.setBorderColor("#0B2265");
    dataTable.setBackgroundColor("#A5ACAF");
    dataTable.setTextColor("#A71930");
    dataTable.addRowData("Name:", "${/name}");
    dataTable.addRowData("Department:", "${/departmentID}");
    dataTable.addRowData("Managed By:", "${/supervisorID}");
    dataTable.setMargin(MarginType.BOTTOM, 10);
    dataTable.setMargin(MarginType.LEFT, 10);
    dataTable.setMargin(MarginType.RIGHT, 10);

    table.addElement(dataTable);
   }

   NodeExpanderButton expand = new NodeExpanderButton();
   expand.setTargetElement(table);
   expand.setAlignment(AlignmentType.MIDDLE_LEFT);
   expand.setMargin(MarginType.TOP, 5);
   expand.setMargin(MarginType.LEFT, 10);
   expand.setMargin(MarginType.BOTTOM, 5);
   expand.setBackgroundColor("#A71930");
   expand.setHeight(12);
   expand.setWidth(12);

   mainPanel.addElement(imagePanel);
   mainPanel.addElement(expand);
   mainPanel.addElement(table);

  }

  NodePanel indicatorPanel = new NodePanel();
  {
   NodeIndicator indicator = new NodeIndicator();
   indicator.setOpacity(1);
   indicator.setWidth(40);
   indicator.setHeight(40);
   indicator.setMargin(MarginType.LEFT, 10);
   indicator.setAlignment(AlignmentType.CENTER);
   indicatorPanel.addElement(indicator);
  }

  rootPanel.addElement(mainPanel);
  rootPanel.addElement(indicatorPanel);
  template.setPanel(rootPanel);

  return template;
 }

}
```

The following image shows the result of implementing the above node template:



29.3 Including a template in a graph configuration

To include a node template in a graph configuration:

1. Navigate to *Administration* > *TIBCO EBX® Data Model and Data Visualization Add-on* > *Value and relationship graphs* > *Table configuration*.
2. Open the desired table configuration and select the **Node Style** tab.
3. In the **Node Template** field, enter the fully qualified name for the class that implements the template.

CHAPTER 30

Using a node value renderer

You can create a class that implements `NodeValueRenderer` if you want to customize the elements of a `NodeValueTemplate` based on node data. When you declare the `getValue()` method, you can use `NodeTemplateContext` to retrieve current node data. See the following example implementation:

```
class SourcePictureRenderer implements NodeValueRenderer
{
 @Override
 public String getValue(NodeTemplateContext context)
 {
  String countryCode = (String) context.getNodeContext()
  .getNode()
  .getRecord()
  .get(Path.parse("/country_code"));
  if(StringUtils.isEmpty(countryCode))
  {
   return "";
  }
  switch (countryCode)
  {
   case "fr":
    return "/common/icons/french_flag.png"
   case "vn":
    return "/common/icons/vietnam_flag.png"
   case "us"
    return "/common/icons/usa_flag.png"
   default:
    return "";
  }
 }
}
```

Once created, you can pass the class to certain template element methods:

- NodeImage.setBindingSource(nodeValueRendererInstance)
- NodeTextBlock.setBindingText(nodeValueRendererInstance)

The following shows how the example `NodeValueRenderer` above can be passed to a `setBindingSource()` method to retrieve an image:

```
NodeImage avatar = new NodeImage();
avatar.setBindingSource(new SourcePictureRenderer())
```

Java API Reference

Release Notes

CHAPTER 31

Version 1.4.12

Released: March 2022

This chapter contains the following topics:

31.1 New features

This release contains no new features.

31.2 Changes in Functionality

This release contains no changes in functionality.

31.3 Changes to third-party libraries

This release contains the following updates to third-party libraries:

- The jQuery library was updated to version 3.6.0.
- The Spring framework was updated to version 5.2.19.

31.4 Closed issues

This release contains no closed issues.

31.5 Known issues

This release contains the following known issues:

- Any saved data model graph based on a duplicated model will refer to the original model and not the duplicated model. Since a saved graph is linked to the XML schema, this behavior is a

result of data model duplication not duplicating the schema file name. Embedded models are not affected; only models in modules.

- If all nodes to expand in a data value graph belong to the same table, the expand service includes the table name. When a table filter is used, the service to expand nodes cannot ascertain the table name to expand. Instead, the following generalized service labels display: **Expand targets**, **Expand sources**, **Expand children**, or **Expand parents**.
- After running the **Center** service on a table, any of its expanded groups are collapsed.
- Data model graphs are not automatically refreshed to reflect updates to displayed tables and foreign keys.
- Orphan groups defined at the root of a data model are not displayed in graphs. However, fields defined at this level do display.
- After resizing a node horizontally on a saved graph, the system still displays the default height for the node.
- Data value graphs do not automatically refresh to reflect updates to displayed data, or nodes.
- When using the containment type for a self-referencing node, it does not display in the graph.
- The template configuration cannot be changed at runtime for custom data model graphs—those created using the Java API.
- In data value graphs, users can move the graph but not graph nodes.
- On Firefox 52 ESR, performance issues may occur when trying to display a data model graph that includes many tables.
- Node position is recalculated after expanding. Therefore, node position may change.
- When a table in a group is hidden, and the group is collapsed, the table's link still displays.
- Graph templates include the option to show hidden fields, but not hidden tables.
- Full screen mode is not supported on IE10 or Microsoft Edge.
- When upgrading to version 1.4.0 or newer, saved data model graphs from earlier versions will be removed.
- When generating a data model graph, the graph cannot show links made via an association.

CHAPTER 32

All Release Notes

This chapter contains the following topics:

32.1 Version 1.4.12

Released: March 2022

New features

This release contains no new features.

Changes in Functionality

This release contains no changes in functionality.

Changes to third-party libraries

This release contains the following updates to third-party libraries:

- The jQuery library was updated to version 3.6.0.
- The Spring framework was updated to version 5.2.19.

Closed issues

This release contains no closed issues.

Known issues

This release contains the following known issues:

- Any saved data model graph based on a duplicated model will refer to the original model and not the duplicated model. Since a saved graph is linked to the XML schema, this behavior is a result of data model duplication not duplicating the schema file name. Embedded models are not affected; only models in modules.
- If all nodes to expand in a data value graph belong to the same table, the expand service includes the table name. When a table filter is used, the service to expand nodes cannot ascertain the table name to expand. Instead, the following generalized service labels display: **Expand targets**, **Expand sources**, **Expand children**, or **Expand parents**.
- After running the **Center** service on a table, any of its expanded groups are collapsed.
- Data model graphs are not automatically refreshed to reflect updates to displayed tables and foreign keys.
- Orphan groups defined at the root of a data model are not displayed in graphs. However, fields defined at this level do display.
- After resizing a node horizontally on a saved graph, the system still displays the default height for the node.
- Data value graphs do not automatically refresh to reflect updates to displayed data, or nodes.
- When using the containment type for a self-referencing node, it does not display in the graph.
- The template configuration cannot be changed at runtime for custom data model graphs—those created using the Java API.
- In data value graphs, users can move the graph but not graph nodes.
- On Firefox 52 ESR, performance issues may occur when trying to display a data model graph that includes many tables.
- Node position is recalculated after expanding. Therefore, node position may change.
- When a table in a group is hidden, and the group is collapsed, the table's link still displays.
- Graph templates include the option to show hidden fields, but not hidden tables.
- Full screen mode is not supported on IE10 or Microsoft Edge.

- When upgrading to version 1.4.0 or newer, saved data model graphs from earlier versions will be removed.
- When generating a data model graph, the graph cannot show links made via an association.

32.2 Version 1.4.11

Released: January 2022

New features

A new property allows you to define the maximum length of link labels. Labels that exceed the limit are truncated. You can view the full label by hovering your mouse over it.

Changes in Functionality

This release contains the following changes in functionality:

- When selecting a graph configuration to display, the names of existing graphs now display in alphabetical order.
- The labels of highlighted links now display according to the **Highlight color** option's settings.
- The message displayed when a graph is taking too long to generate was updated to be more informative.

Changes to third-party libraries

This release contains no updates to third-party libraries.

Closed issues

This release contains no closed issues.

Known issues

This release contains the following known issues:

- Any saved data model graph based on a duplicated model will refer to the original model and not the duplicated model. Since a saved graph is linked to the XML schema, this behavior is a result of data model duplication not duplicating the schema file name. Embedded models are not affected; only models in modules.
- If all nodes to expand in a data value graph belong to the same table, the expand service includes the table name. When a table filter is used, the service to expand nodes cannot ascertain the table name to expand. Instead, the following generalized service labels display: **Expand targets**, **Expand sources**, **Expand children**, or **Expand parents**.
- After running the **Center** service on a table, any of its expanded groups are collapsed.
- Data model graphs are not automatically refreshed to reflect updates to displayed tables and foreign keys.
- Orphan groups defined at the root of a data model are not displayed in graphs. However, fields defined at this level do display.
- After resizing a node horizontally on a saved graph, the system still displays the default height for the node.

- Data value graphs do not automatically refresh to reflect updates to displayed data, or nodes.
- When using the containment type for a self-referencing node, it does not display in the graph.
- The template configuration cannot be changed at runtime for custom data model graphs—those created using the Java API.
- In data value graphs, users can move the graph but not graph nodes.
- On Firefox 52 ESR, performance issues may occur when trying to display a data model graph that includes many tables.
- Node position is recalculated after expanding. Therefore, node position may change.
- When a table in a group is hidden, and the group is collapsed, the table's link still displays.
- Graph templates include the option to show hidden fields, but not hidden tables.
- Full screen mode is not supported on IE10 or Microsoft Edge.
- When upgrading to version 1.4.0 or newer, saved data model graphs from earlier versions will be removed.
- When generating a data model graph, the graph cannot show links made via an association.

32.3 Version 1.4.10

Released: December 2021

New features

This release contains no new features.

Changes in Functionality

This release contains no functionality changes.

Changes to third-party libraries

The Spring Framework was updated to version 5.2.15.

Closed issues

This release contains no closed issues.

Known issues

This release contains the following known issues:

- Any saved data model graph based on a duplicated model will refer to the original model and not the duplicated model. Since a saved graph is linked to the XML schema, this behavior is a result of data model duplication not duplicating the schema file name. Embedded models are not affected; only models in modules.
- If all nodes to expand in a data value graph belong to the same table, the expand service includes the table name. When a table filter is used, the service to expand nodes cannot ascertain the table name to expand. Instead, the following generalized service labels display: **Expand targets**, **Expand sources**, **Expand children**, or **Expand parents**.
- After running the **Center** service on a table, any of its expanded groups are collapsed.

- Data model graphs are not automatically refreshed to reflect updates to displayed tables and foreign keys.
- Orphan groups defined at the root of a data model are not displayed in graphs. However, fields defined at this level do display.
- After resizing a node horizontally on a saved graph, the system still displays the default height for the node.
- Data value graphs do not automatically refresh to reflect updates to displayed data, or nodes.
- When using the containment type for a self-referencing node, it does not display in the graph.
- The template configuration cannot be changed at runtime for custom data model graphs—those created using the Java API.
- In data value graphs, users can move the graph but not graph nodes.
- On Firefox 52 ESR, performance issues may occur when trying to display a data model graph that includes many tables.
- Node position is recalculated after expanding. Therefore, node position may change.
- When a table in a group is hidden, and the group is collapsed, the table's link still displays.
- Graph templates include the option to show hidden fields, but not hidden tables.
- Full screen mode is not supported on IE10 or Microsoft Edge.
- When upgrading to version 1.4.0 or newer, saved data model graphs from earlier versions will be removed.
- When generating a data model graph, the graph cannot show links made via an association.

32.4 Version 1.4.9

Released: June 2021

Library updates

Spring Data was removed from `ui-framework-dependencies.jar`.

Bug fixes

[DMDV-2160] Tomcat runs out of memory shortly after startup.

32.5 Version 1.4.8

Released: May 2021

Updates

The jsPDF library was updated to version 2.3.1.

Bug fixes

[DMDV-2064] An exception occurs when generating a data graph.

32.6 Version 1.4.7

Released: March 2021

Bug fixes

An update was applied to correct an issue with the Apache Standard Taglibs library.

32.7 Version 1.4.6

Released: February 2021

Bug fixes

[DMDV-2037] The export, change orientation, and full-screen buttons do not work when viewing a graph via a workflow step.

32.8 Release Notes 1.4.5

Released: January 2021

Changes and updates

The following libraries were updated in this release:

- Apache Standard TagLib library to version 1.2.3.
- Spring framework library to version 5.2.9.
- Jackson databind library to version 2.11.2.

Bug fixes

[DMDV-1893] The add-on services do not work when TIBCO EBX® is embedded in an iframe.

32.9 Release Notes 1.4.4

Release Date: October 20, 2020

Bug fixes

[DMDV-1866] Some images do not display correctly when using a custom URL for your TIBCO EBX® server.

32.10 Release Notes 1.4.3

Release Date: September 18, 2020

Bug fixes

[DMDV-1794] A deadlock in the add-on affects the server where EBX® is running.

32.11 Release Notes 1.4.2

Release Date: June 23, 2020

Updates and changes

The jQuery and Jackson Databind libraries were updated and the SLF4J library was removed.

32.12 Release Notes 1.4.1

Release Date: April 20, 2020

Bug fixes

[DMDV-1698] A deadlock in the add-on affects the server where EBX® is running.

32.13 Release Notes 1.4.0

Release Date: February 20, 2020

New features

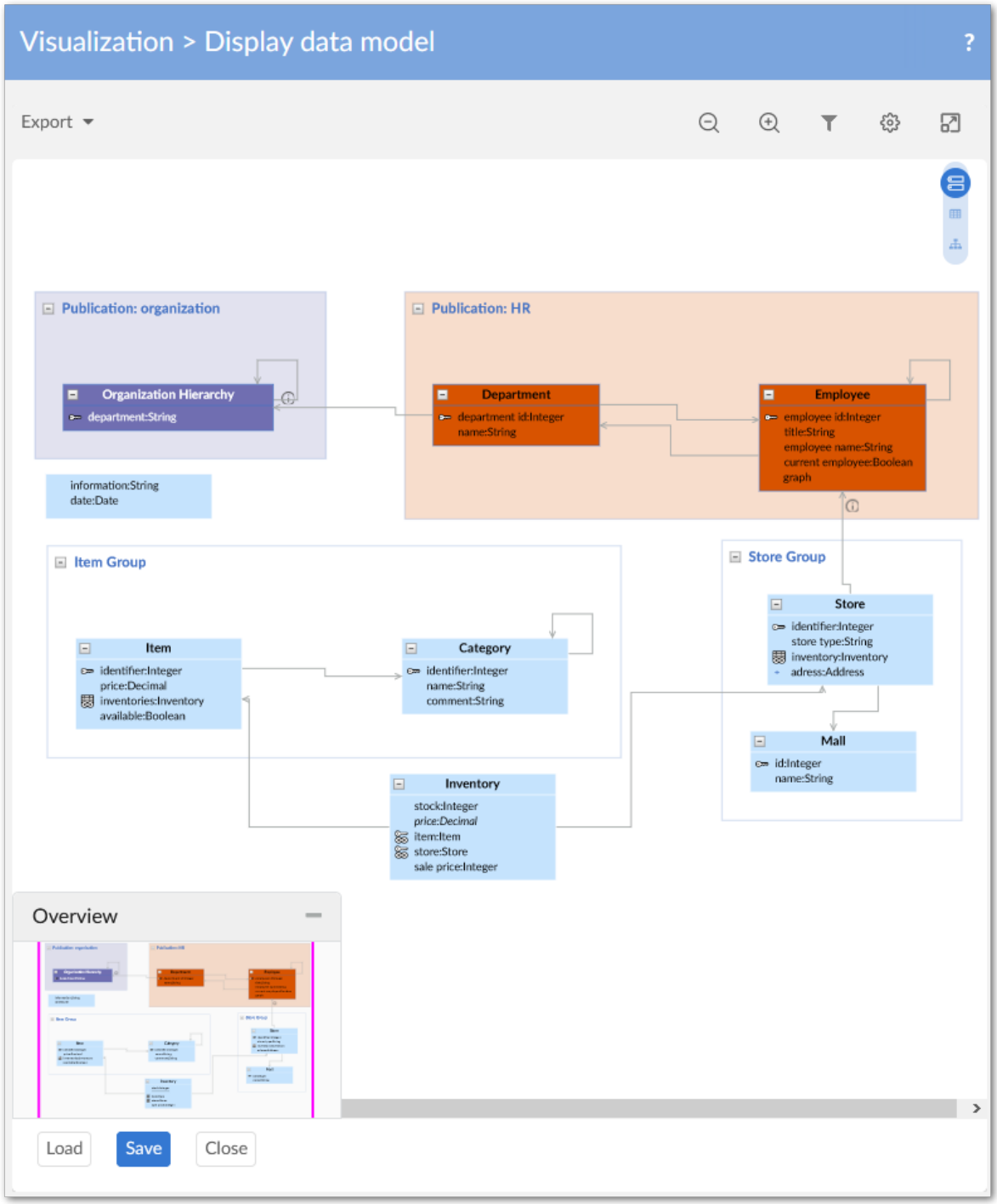
The sections below highlight new in this release:

- New data model graph features [p 137]
- New data value graph features [p 139]

New data model graph features

Data model graphs can now display multiple data models. When a graph includes a table from an outside data model, users can right-click to select the **Display external model** service. Each outside data model displays using alternate colors to differentiate between them. Normal graph interaction is available when displaying multiple models. For example, users can apply templates to change appearance, re-arrange, filter, save and share the graphs. An additional feature for graphs containing multiple models is the level selector. This allows users to change the level of the graphs detail to: **Field**, **Table**, or **Model**. Choosing a level expands graph components to the selected level of detail and automatically resizes the graph.

The following image shows a data model graph with external models shaded and the base data model highlighted like a group:



Filter pane and table selection

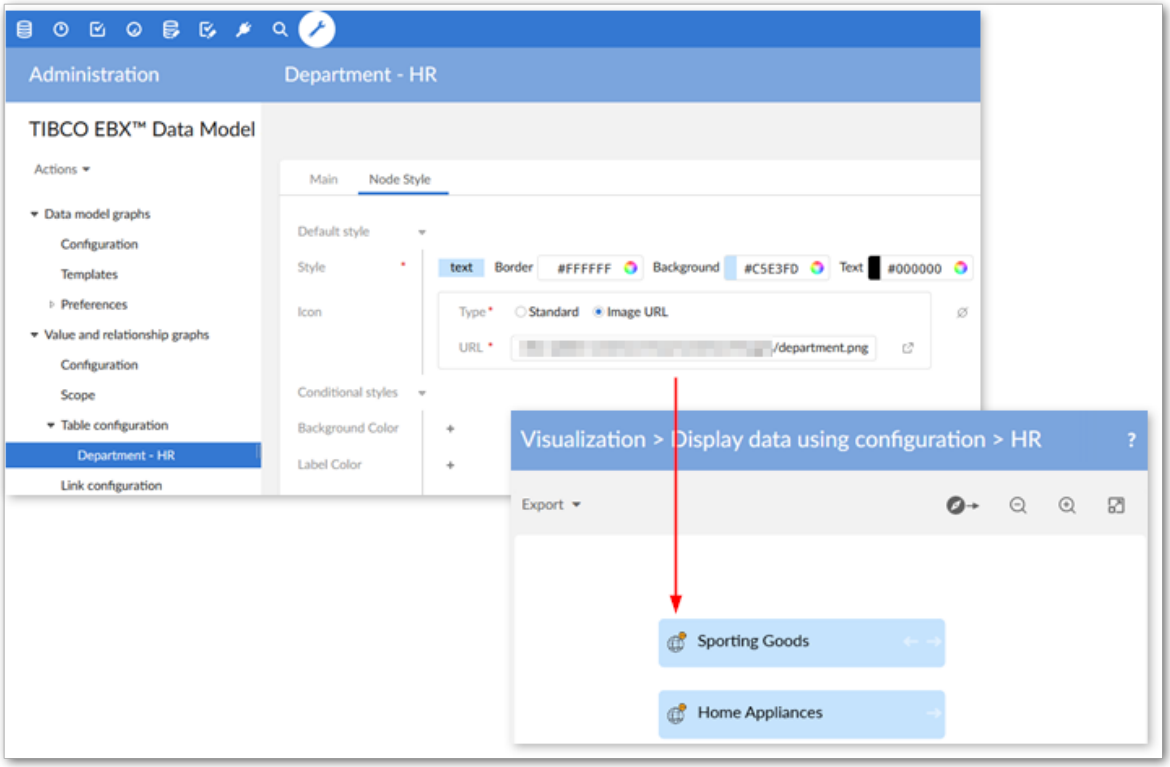
The data model filter pane and table selection screens have been updated with:

- Options to display and hide external data models.
- An  icon to indicate that a graph is displaying a table from an outside model, but the model is hidden.
- Tabs that contain all, displayed, and hidden tables and data models.

New data value graph features

In addition to the option of adding one of the icons provided with the add-on, you can now update a table node style to include a custom icon. This functionality is provided through the new **Image URL** option when viewing a configured table's **Node Style** tab. The URL must point to an image located in the same domain as TIBCO EBX®.

The following image shows an example of adding a custom image to a node with a URL:



Attention

Saved data model graphs from previous add-on versions cannot be migrated and will be deleted.

32.14 Release Notes 1.3.0

Release Date: November 8, 2019

New features

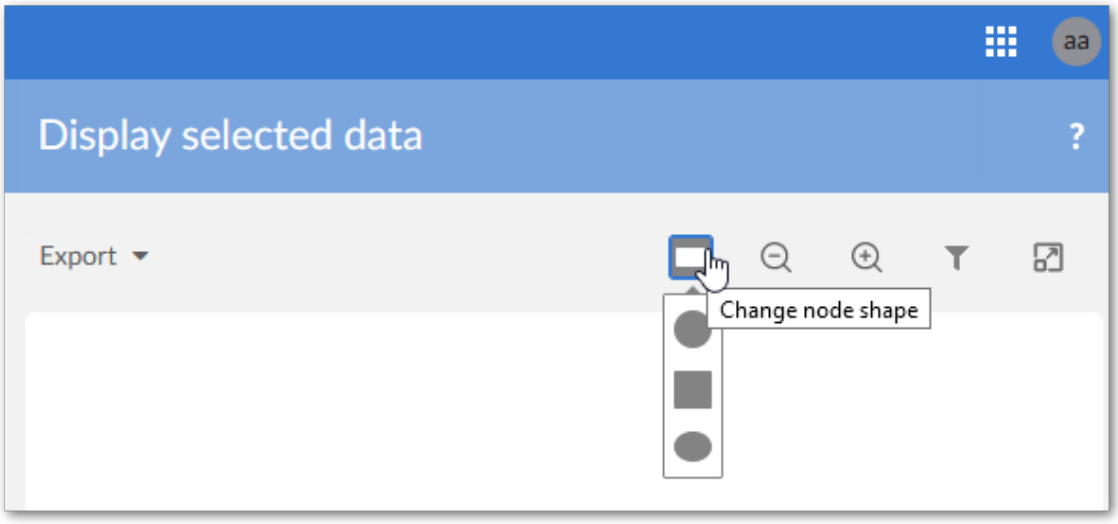
The sections below highlight new in this release:

- New graph features [p 140]

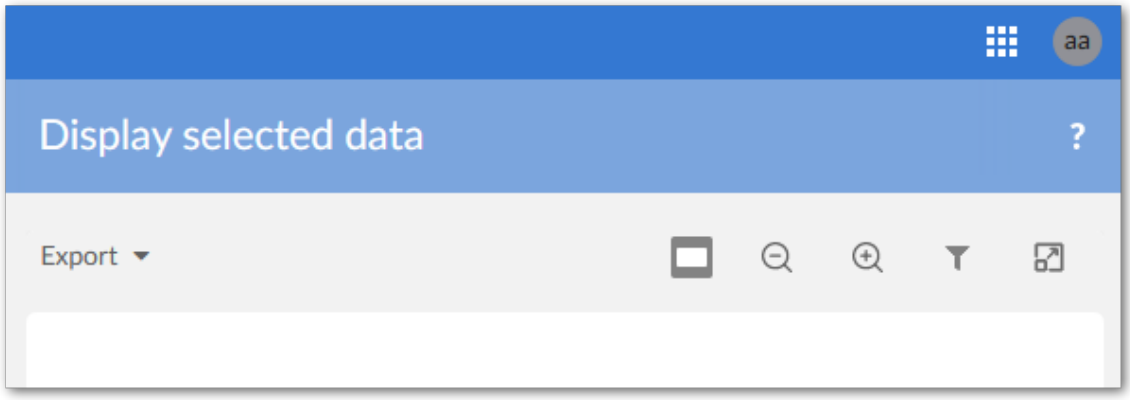
New graph features

The following updates have been made to graph functionality:

- Administrators can set a new configuration option to automatically expand child and source nodes to display their respective parents and targets. When enabled, the add-on automatically expands nodes on graph open or when a user expands an existing node to display these child/source nodes.
- As shown below, when viewing data value and relationship graphs, users can now change node shapes:



- To improve the feel of the interface, the tooltip design has been updated and as shown below, graph button display locations have been changed:



API features

This release contains the following new and updated API features:

- A custom service can now be written using the add-on's API to display a default data value graph based on record selection. See Displaying a default data value graph [p 113] for a sample implementation.

- Data value graph features including, but not limited to, **Overview** box display, access to detailed record views, and availability of contextual menu options can be enabled/disabled. The new `GraphDataFeatures` enumerates available features.

Bug Fixes

This release contains the following bug fixes:

- **[DMDV-1323]** An exception is thrown when accessing a table.
- **[DMDV-1355]** Datasets based on external models were not working as expected due to a bug in the add-on.

32.15 Release Notes 1.2.1

Release Date: August 6, 2019

Bug Fixes

This release contains the following bug fixes:

- **[DMDV-1229]** Users can run **Display selected data** service from the **Actions** menu of the **History** table.
- **[DMDV-1230]** Users can run **Display data model** service from the **Actions** menu of the **Permission** table.

32.16 Release Notes 1.2.0

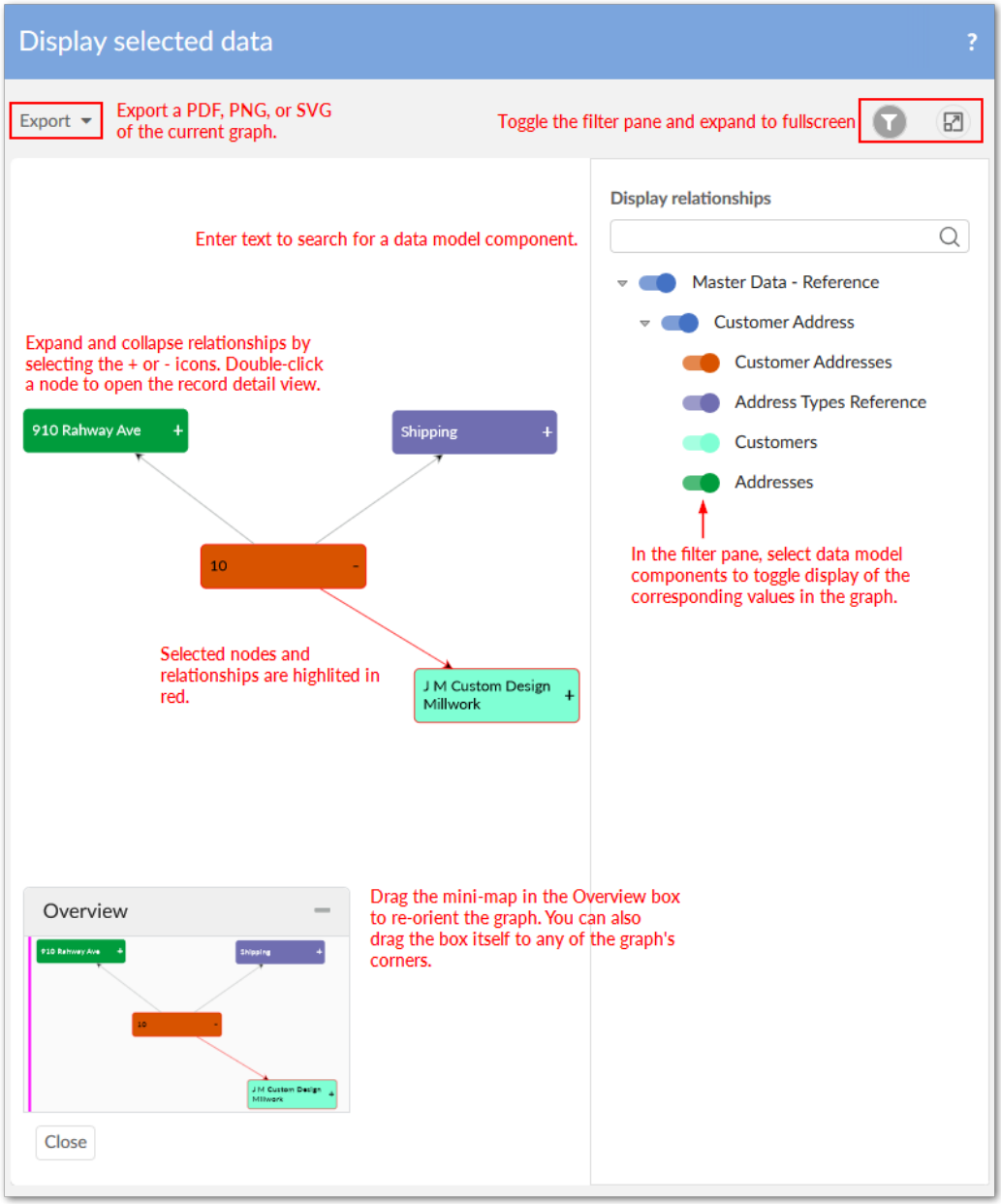
Release Date: June 20, 2019

New features and enhancements

This release contains the following new features:

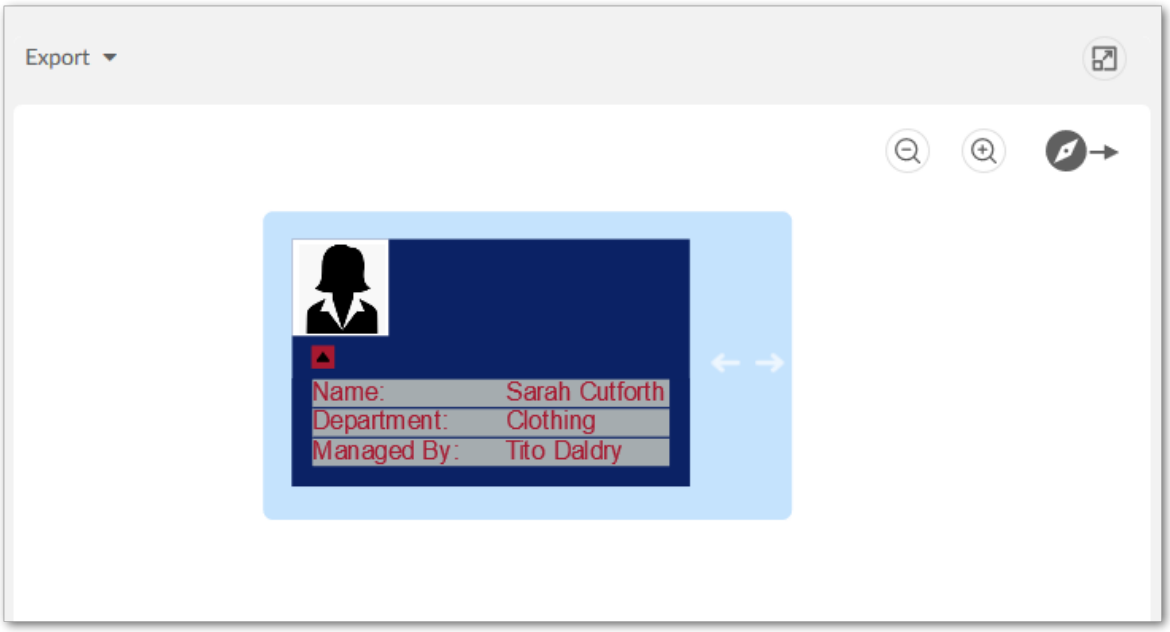
- The **Display data model graph** service is now available to run when viewing a data model in the Data Modeler Assistant area.
- By default, the ability to view a data value and relationship graph no longer requires configuration. The new **Display selected data** service is available from table **Actions** menus. Selected records and relationships display using default settings. In addition to standard navigation and export features, users can search for and toggle display of nodes and relationships. See Default graphs [p 46] for more information.

The following image highlights some of the default features:



- Data value graph services can now be called from a perspective or workflow. See Graphs in perspectives and workflows [p 75] for more information.
- The API now allows you to create custom node templates that determine node appearance. Customizable node components include, but are not limited to, images and data tables. See Node templates [p 116] for more information.

The following image provides an example of a custom node:



32.17 Release Notes 1.1.0

Release Date: February 22, 2019

New features and enhancements

This release contains the following new features:

- Data value graph orientation has been updated. When administrators configure or users view a graph, they can specify that nodes display: Left-to-right, Right-to-left, Top-down, or Bottom-up.
- Dynamic labels can now be defined for data value graph nodes and links. The following types of labels can be defined:
  - Default: Label text can be directly entered, or data model elements selected. Labels based on data model elements automatically update based on changes to the associated value.
  - Localized: In addition to sharing the same features as default labels, these labels can leverage the EBX® localization feature.
  - Programmatic: Label definitions can be implemented in a Java class.
- Node and link styles can now be based on the evaluation of an XPath expression. For example, you could specify that all graph nodes that refer to an employee with a status of inactive are colored red. You can also use a Java class to define programmatic styles for links.
- Links and their corresponding values can now be filtered out of data value graphs. A conditional filter is used for each defined foreign key in the graph. If the filter evaluates to true, the relationship is removed.
- Data model graphs can now be shared with other users.

Known limitations

This release contains the following known limitations:

- The indicator and node label position are incorrect when expanded with many containment levels.
- When using a curved line to display a link, the arrow doesn't exactly align with the line.

Bug Fixes

This release contains the following bug fixes:

- **[DMDV-258]** The **Load a graph** menu displays even when users do not have permission to view any of a dataset's tables.
- **[DMDV-753]** When updating a record label in a parent dataset, the external dataset's node label is not updated.
- **[DMDV-828]** When generating a model graph, fields are still displayed even when set to **Not a data node** in the data model.

32.18 **Release Notes 1.0.2**

Release Date: January 11, 2019

Bug fixes

This release contains the following bug fixes:

- **[DMDV-867]** The **Display data model** service cannot generate a graph when the data model links to an external table, which in turn links to another external table.

32.19 **Release Notes 1.0.1**

Release Date: December 3, 2018

Bug fixes

This release contains the following bug fixes:

- **[36371]** Provide an API to display or hide the Close button.
- **[36394]** Due to low resolution, images of exported graphs are blurry. This prevents users from being able to zoom in and get detailed information.
- **[36840]** The **Display data using configuration** service does not work on a table row toolbar.

32.20 **Release Notes 1.0.0**

Release Date: October 26, 2018

Overview of features

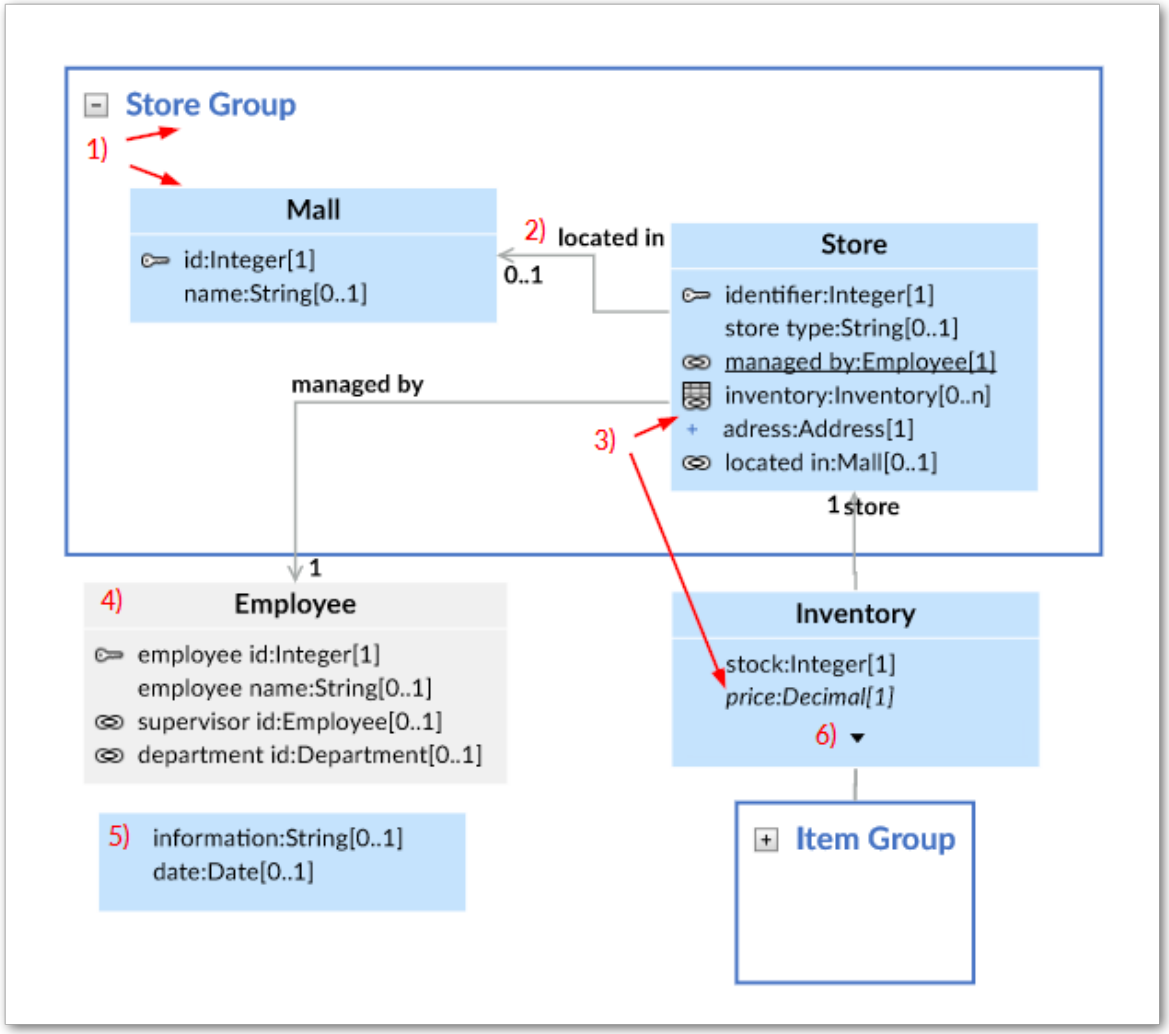
By generating interactive graphs of your data, relationships, and data structure, the EBX® Data Model and Data Visualization Add-on allows you to:

- View *data structure* by rendering data model graphs. Data model graphs are not limited to EBX® data sources. Using the API, graphs can be generated from external sources.
- View *data values and relationships* by rendering graphs that show selected values and their relationships.

The following sections highlight additional features and functionality of the EBX® Data Model and Data Visualization Add-on.

Data model graphs

By default, no configuration is required to display data model graphs. You can display a graph by running the **Display data model** service from a table or dataset **Actions** menu. What displays in a graph depends on the model content from which it was generated and the template settings. The following image and table describe data model graph components.

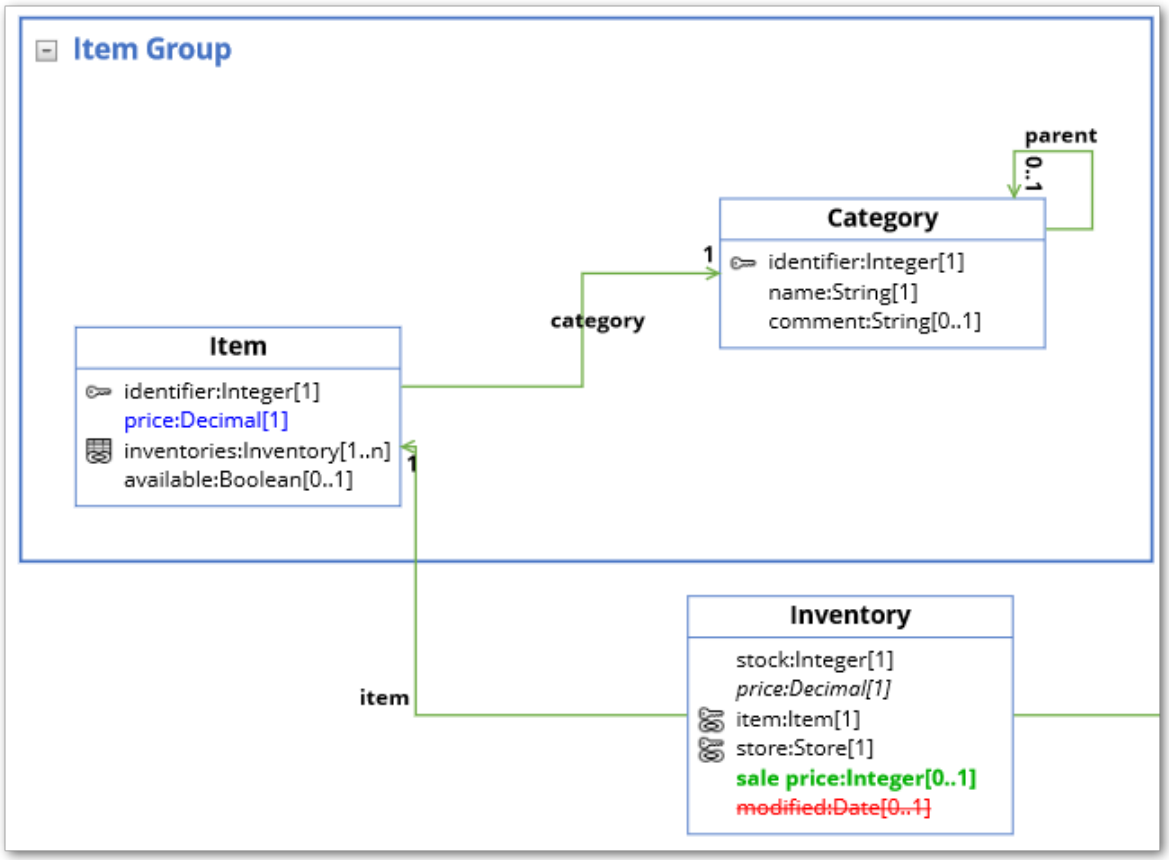


1) Groups and Tables	A box is automatically drawn around components included in a group. You can collapse, expand, re-size and rearrange groups. The bottom-right corner of the images shows a collapsed group. Tables display their labels and fields. Standard DMA icons are used to indicate keys and relationships.
2) Relationships	Arrows indicate foreign key relationships and their direction. The labels correspond to each foreign key field.
3) Underlined field, plus icon, and italic fields.	An underlined field belongs to a table located outside of the current dataset. The plus icon next to a field indicates you can expand to display more information, such as fields included in a complex type. Fields in italics represent inherited fields.

4) Table located outside of the current dataset	When a table is located outside of the current dataset, it will be shaded in the graph. The color of the shade is editable in the template.
5) Orphan fields	Fields that are not part of a table display without a title.
6) Display additional fields	When a table holds more fields than can display, use the arrow icon to expand/collapse.

For more information on interacting with data model graphs, see Interacting with data model graphs [p 33].

You can save and load data model graphs. When loading a saved graph the add-on can help you visualize changes to model structure. As shown below, updated graphs highlight new items in green, changed items in blue, and deleted items in red (colors may differ depending on configuration):

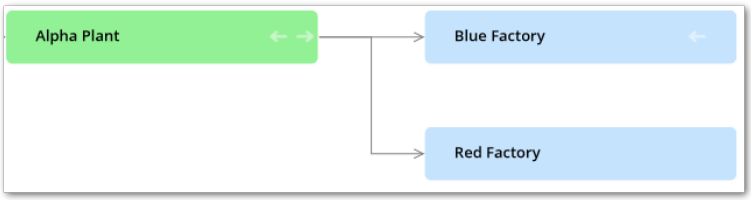


Model graph API

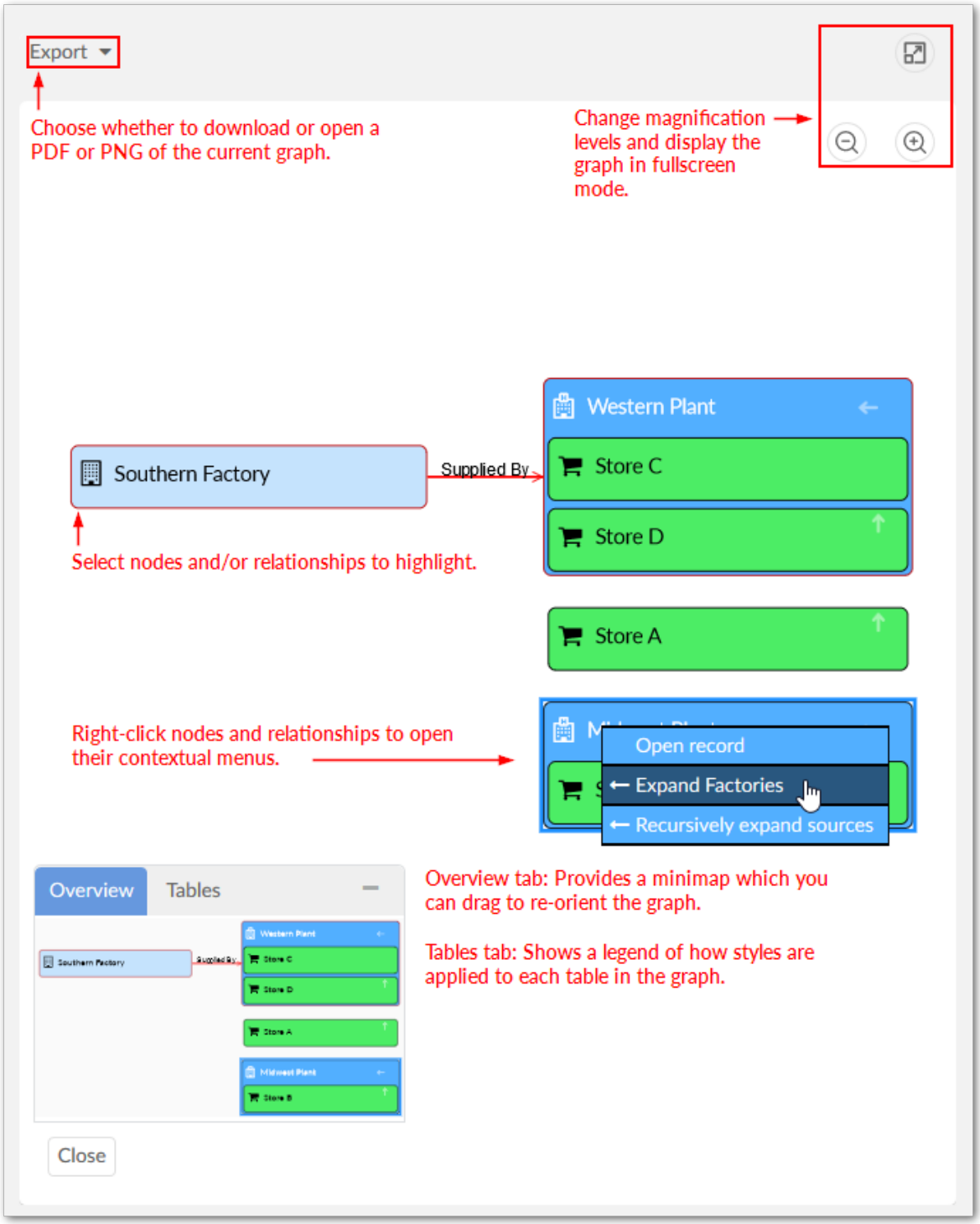
The EBX® Data Model and Data Visualization Add-on's API allows you to generate custom data model graphs from external sources and customize templates to alter look and feel. See Generating a model graph from an external source [p 99] for more details. Additionally, you can create a custom service from which users can generate a custom data model graph. See Generating from a service [p 103] for more details.

Data value and relationship graphs

A data value graph is a visual representation of how data values relate to each other. A graph can display relationships between multiple data models. Administrators configure the way relationships render when users run the **Display values and relationships** service by assigning a display type to each relationship. The table below describes the available types:

Type	Example and description
As Lines	As shown in the following image when you set the display to lines, values display in rectangular nodes and lines between nodes represent relationships. 
As Containers	As shown in the following image containers show a parent/child relationship as nodes within nodes. Outer nodes are the parents of the smaller, inner nodes. 

The following image highlights some of the data value and relationship graph features; see Configuring graphs [p 49] for more details:



Value graph API

The EBX® Data Model and Data Visualization Add-on's API allows you to:

- Generate an existing data value and relationship graph for display in a UI tab. See Data value and relationship graph options [p 105] for more details.
- Write a custom service from which users can generate an existing data value and relationship graph. See Displaying a graph from a custom service [p 108] for more information.

- Create a custom filter for data value and relationship graphs. See Filtering data values and relationships [p 110] for more details.

Known limitations

The EBX® Data Model and Data Visualization Add-on has the following known limitations:

- Any saved data model graph based on a duplicated model will refer to the original model and not the duplicated model. Since a saved graph is linked to the XML schema, this behavior is a result of data model duplication not duplicating the schema file name. Embedded models are not affected; only models in modules.
- If all nodes to expand in a data value graph belong to the same table, the expand service includes the table name. When a table filter is used, the service to expand nodes cannot ascertain the table name to expand. Instead, the following generalized service labels display: **Expand targets**, **Expand sources**, **Expand children**, or **Expand parents**.
- After running the **Center** service on a table, any of its expanded groups are collapsed.
- The **Display data model** service is not available in the EBX® Data Modeler Assistant (DMA).
- Data model graphs are not automatically refreshed to reflect updates to displayed tables and foreign keys.
- Orphan groups defined at the root of a data model are not displayed in graphs. However, fields defined at this level do display.
- After resizing a node on a saved graph, the system still displays the default size for the node.
- Data value graphs do not automatically refresh to reflect updates to displayed data, or nodes.
- When using the containment type for a self-referencing node, it does not display in the graph.
- The template configuration cannot be changed at runtime for custom data model graphs—those created using the Java API.
- In data value graphs, users can move the graph but not graph nodes.
- On Firefox 52 ESR, performance issues may occur when trying to display a data model graph that includes many tables.
- Node position is recalculated after expanding. Therefore, node position may change.
- When a table in a group is hidden, and the group is collapsed, the table's link still displays.
- Graph templates include the option to show hidden fields, but not hidden tables.
- Full screen mode is not supported on IE10 or Microsoft Edge.
- In workflows, no user task is available to run visualization services.
- The **Display data values and relationships** service is not available for perspectives.
- The add-on cannot generate a data value and relationship graph that includes more than 2000 nodes.