



TIBCO® BPM Enterprise

Case Data User Guide

Version 4.3.2

May 2022



Contents

Case Data Overview	5
Case Data Terminology	6
Case Data Model Deployment Lifecycle	8
Creating and Deploying a Case Data Model	10
BOM Type to Database Type Mappings	10
Modeling Case Data in a Global BOM	11
Case Classes, Global Classes and Local Classes	12
Case Identifiers	12
Adding a Case Identifier to a Case Class	13
Changing a Case Identifier's Type	13
Case States and Case Actions	14
Adding a Case State to a Case Class	15
Converting an Existing Case Class Attribute to a Case State	15
Creating a Case Action	16
Generating a Case Action From a Case Class	16
Case Summaries	17
Adding Attributes to a Case Summary	18
Deleting Attributes From a Case Summary	19
Defining the Order in Which Case Summary Attributes are Displayed at Runtime	19
Searchable Attributes	20
BOM Relationships and Global BOMs	20
Converting a Local BOM to a Global BOM	23
Upgrading a Case Data Model	24
Case Data Model Versioning	25
Process Dependencies and Case Data Model Versions	27
Upgrading a Case Data Model to a New Minor Version	31
Upgrade Error: Incompatible changes made during upgrade of version number of model	32
Upgrading a Case Data Model to a New Major Version	33
Deleting a Case Data Model	35
Using a Case Data Model in a Business Process	36
Creating a New Case Object	36
Displaying a Case Object in a User Task - Using a Form	37
Displaying a Case Object in a User Task - Using a Pageflow	37
Updating a Case Object	38
Notifying a Process That a Case Object It Uses Has Been Modified	39
CaseSignalAttributes Class and Methods	43

Deleting Case Objects	43
Reasons to Avoid Deleting Case Objects	43
Deleting Case Objects by Case Reference	44
Deleting Case Objects by Case Identifier	45
Deleting a Case Object by Composite Case Identifier	47
Searching For and Reading Case Objects	48
Case Access Classes	48
Finding a Case Object by its Case Identifier	49
Finding Case Objects by Example	50
Finding All Case Objects of a Particular Type	51
Finding Case Objects by Criteria	52
Case Data Query Language (DQL)	54
Attribute Paths	56
Operators	58
Functions	59
Values	59
Creating a Criteria Object	62
Criteria Object Methods and Attributes	63
Case Reference Methods and Attributes	64
Navigating Association Links to find Related Case Objects	65
Association Links and Association Relationships	68
Creating an Association Relationship Between Two Case Classes	69
Creating an Association Link	70
Deleting an Association Link	72
Reading Case Objects	73
Handling Case Objects That Use Inheritance in Scripts	73
Processing a Paginated List of Case References	75
Generating a Business Service to Create/Update/Delete Case Data	77
Using Case Data to Display Work Item Lists From Business Services Using TIBCO Openspace	78
Using Case Data to Open Work Lists From Business Services Using Custom Clients	79
Business Data Services Properties (bds.properties)	82
Configuring Whether Case Data Tables Are Managed Automatically or Manually	86
Case Folders	87
Terminology for Case Folders	87
Content Management System	88
TIBCO DocumentStore	89
Case Folder Configuration	89
Document Operations Service Type	91
Delete Document	92

Find Documents Linked to Case Object 93

Link Document to Case Object 94

Link System Document 94

Move Document Between Case Objects 95

Unlink Document From Case Object 95

Viewing Case Folders in Openspace 95

Case Folder System Actions 96

TIBCO Documentation and Support Services 97

Legal and Third-Party Notices 99

Case Data Overview

Case data is business data that is centrally managed by TIBCO® BPM Enterprise (formerly TIBCO ActiveMatrix® BPM) and can therefore be accessed and updated by multiple BPM process applications.

Business data is structured data that contains information about real-world entities or business concepts that an organization needs to manipulate, for example: Customer, Order, Orderline, Claim or Policy.

ActiveMatrix BPM supports two sorts of business data - normal (or local) business data, and (from version 3.0) case data. In contrast to normal business data, case data:

- can have a state (that is, it can have data and/or documents associated with it).
- can be referenced by a set of processes and/or work items that are responsible for making state changes to it.
- has its own audit history, independent of any processes or work items that operate on it.

Case data can be modeled at design-time as [case classes](#) in a [case data model](#), then represented at runtime as [case objects](#), which can be referenced by corresponding [case references](#).

Case objects and case references are both stored in a set of [case data tables](#) (that represent the case data model) in the [case data store](#).

Any BPM application can interact with a case data model to create and use case data. A BPM application, in this context, is either:

- a BPM process, defined in TIBCO Business Studio and deployed to the BPM runtime.
- a custom client application that uses the ActiveMatrix BPM public API to invoke BPM services provided by the BPM runtime.

Case Data Features in ActiveMatrix BPM

ActiveMatrix BPM provides a broad range of features and capabilities that you can use to manage and manipulate case data in the following ways:

- create the database tables to contain the case data store.
- create, deploy, upgrade and delete case data models.
- create, update and delete case objects in a process.
- display and update case objects in forms or pageflows.
- use case data to find in-progress work items and/or process instances that are associated with a particular case.
- locate a particular case and perform [case state-specific actions](#) on it.
- update case objects on an ad-hoc basis, independent of any enterprise process update - for example, a customer reporting a change of address.
- generate standalone pageflows and forms which you can use to create, update or delete case objects in a particular case model.

Case Data Terminology

Case Actions

A case action is a special type of business service, associated with a case class, that defines an action a user can perform that is related to a case. The availability of case actions can be restricted based on the current case state and/or the user's permissions.

Case Classes

A case class is a template for a case object. A case class must have at least one case identifier, and can have as many as are required by the nature of the data represented.

You can create a case object from a case class from within any BPM process application that references that case class.

Unlike local objects, which exist only over the lifespan of a process, case objects are persisted - each case object is stored in the case data tables that represent the case model from which the case object was instantiated.

Case Data Models (or Case Models)

A case data model is the ActiveMatrix BPM representation of a particular set of case data.

At design-time, a case data model is represented as one or more global BOMs in a Business Data project. Cases are modeled as case classes. The project must be deployed to the BPM runtime to make the case data model available to BPM applications.

At runtime, a case data model has two components:

- a business data application. (This contains the BDS Plug-ins generated from the Business Data project BOMs.)
- an associated set of case data tables in the case data store. These tables will be used to store case objects created and used by BPM process applications that reference this case data model.

Case Data Store

The case data store is the database in which case data models are stored, and from which processes can access items of case data (case objects).

The case data store is created as part of the ActiveMatrix BPM installation process, and can be either part of the BPM database, or a separate database

Case Data Tables

When a case data model is deployed from Business Studio, case data tables representing that case data model must be created in the case data store. (ActiveMatrix BPM can be configured to either do this automatically or to generate SQL scripts that must be manually run by a DBA.) After this has been done, instances of process applications that use that case model can be started and run.

TIBCO recommend that:

- In a development environment, you use the BPM database and automatic table management.
- In a production environment, you use a separate database and manual management. This gives the DBA full control over the database.

Case Identifiers

A case identifier (CID) is a special type of attribute that can be used to uniquely identify an instance of a case class. It can be used in processes, scripts or API calls to create or to find a particular case.

Case Objects

A case object is an instance of a case class. Data for a case object is stored in the case data tables associated with the case data model to which the case class belongs. You can create case objects from within a process by using a Global Data service task.

Case References

A case reference is a unique reference (or pointer) to a case object, created by ActiveMatrix BPM when that case object is created.

Creating a case object creates and returns a case reference. That reference can be used by any process to find and manipulate that case object.

You can use a case reference within a process to find, display, update or delete an existing case object.

Case States

A case state is a special type of attribute, available only on case classes, that can be used to uniquely identify a set of business-specific states that a case can be in. Case states can be used to control the availability of case actions to users.

Case Summaries

A case summary is a specified subset of the attributes defined for a case class. The case summary defines the initial set of information that is returned about a case object when that object is returned as part of a search result. When a user performs a search on a case class using the Openspace Case Management gadget, the case summary information is displayed for each matching case object.

Global BOMs

A global BOM must contain a case class or a global class. It can also contain, as required, further case classes or global classes, and local classes. A global BOM can only be created in or added to a Business Data project.



A Business Data project can also contain local BOMs that contain only local classes.

Global Classes

A global class is a component of one or more case classes. A global class cannot have a case identifier and cannot be created or accessed independently of a case class.

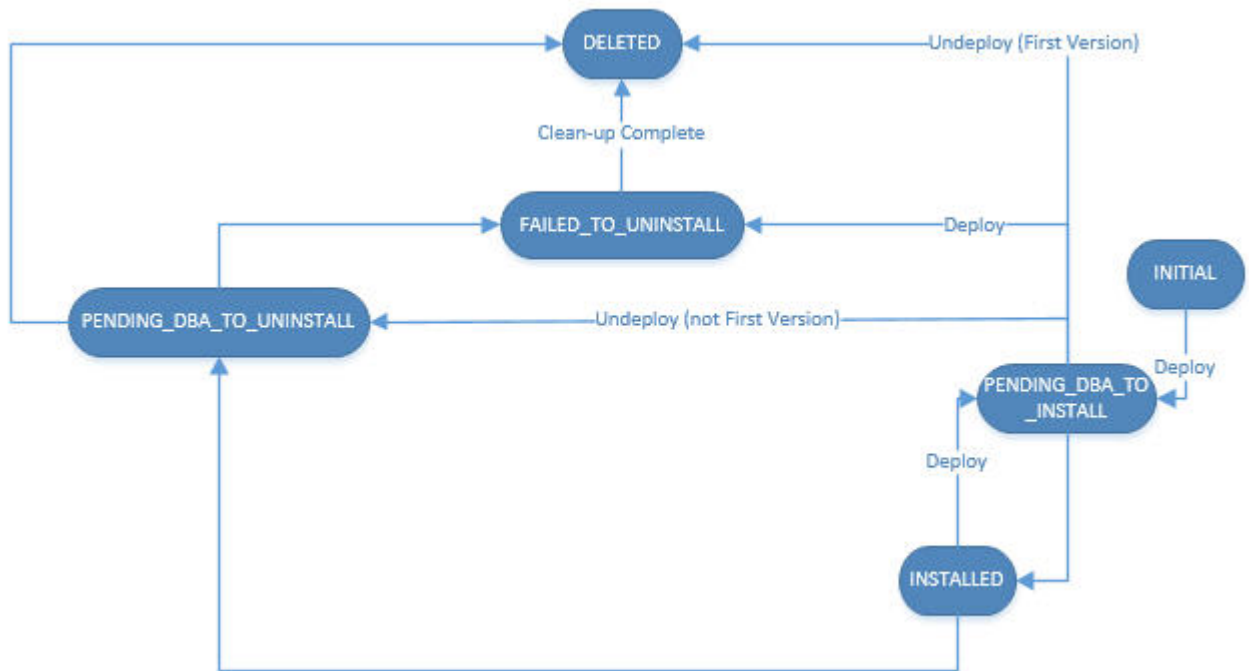
Case Data Model Deployment Lifecycle

Case data models have a deployment lifecycle and states.

The *deployment lifecycle* means that you can deploy new versions and obtain UPDATE database scripts. Similarly, undeployment of the last remaining version of a deployed application creates the DROP scripts that you can use to remove the case data from the BPM database. See *TIBCO ActiveMatrix BPM Deployment Guide* for more information about application deployment lifecycles.

The case model *state* depends on where the case model is in the deployment lifecycle.

The following diagram shows the deployment lifecycle.



The following table describes the states a case model can have.

State	Description
INITIAL	This describes the initial state of the case model. The case model is either still in the process of deploying or is undeployed.
PENDING_DBA_TO_INSTALL	The CREATE/UPDATE scripts are ready for retrieval. You must retrieve the scripts and manually execute them on your database to generate the case model database schema. If the first version of a case model is undeployed before the UPDATE scripts for a new version of the case model have been executed, then the case model's state changes to PENDING_DBA_TO_UNINSTALL so the DROP scripts can be retrieved.
INSTALLED	The CREATE/UPDATE scripts have been executed and TIBCO ActiveMatrix BPM has been notified.

State	Description
PENDING_DBA_TO_UNINSTALL	The DROP script is ready for retrieval. You must retrieve the scripts and manually execute them on your database to generate the case model database schema.
FAILED_TO_UNINSTALL	The scripts have not executed correctly on the database. You must clean-up your database and then notify TIBCO ActiveMatrix BPM when the clean-up is complete.
DELETED	The DROP script has been executed and the case model database tables deleted.

Creating and Deploying a Case Data Model

Before you can use case data in a business process, you must create a model of that data and deploy it as a separate application.

Procedure

1. In TIBCO Business Studio, click **File > New Business Data project**.
2. In the New Business Data Project wizard:
 - a) Enter a suitable **Project name**.
 - b) Set the project's **Version number**.
 - c) In **Destination Environments**, click **BPM**.
 - d) Click **Finish**.

A new project is created, containing an empty BOM.
3. [Model your case data](#) as required.
4. Use your preferred application deployment method to deploy the Business Data project.
5. In ActiveMatrix Administrator, verify that the new business data application is **Running** and **In Sync**.
6. Using the Openspace Data Admin gadget, verify that the new case model is in the expected state:
 - **INSTALLED**, indicating that the required case data tables have been automatically created by ActiveMatrix BPM.
 - **PENDING_DBA_ACTION_TO_INSTALL**, indicating that a DBA will need to obtain and manually run the script to create the case data tables, then notify the BPM runtime that this has been done. See "Retrieving CREATE Case Data Database Scripts" in the *TIBCO Openspace User's Guide* for more information about how to do this.

Result

If the model is in the **INSTALLED** state, you can now use it in business processes, by referencing the defined case classes to create, read, update, delete and search for case objects. See [Using a Case Data Model in a Business Process](#).

BOM Type to Database Type Mappings

When a case model is deployed to ActiveMatrix BPM, the different BOM types are converted into the appropriate database types for use in the CREATE, UPDATE or DROP database scripts.

BOM Type	Database Type		
	SQL Server	Oracle	DB2
Boolean	tinyint	number(1,0)	smallint
Date	datetime	timestamp	timestamp
Date Time	datetime	timestamp	timestamp
Date Time and Time Zone	datetime	timestamp	timestamp

BOM Type	Database Type		
	SQL Server	Oracle	DB2
Decimal (Floating Point)	double	binary_double	double
Decimal (Fixed Point)	numeric(p,s)	number(p,s)	numeric(p,s)/decimal
Integer (Signed)	int	number(10,0)	integer
Integer (Fixed Length)	numeric(p,0)	number(p,0)	numeric(p,0)/decimal
Duration	nvarchar(255)	nvarchar2(255)	varchar(255)
ID	nvarchar(255)	nvarchar2(255)	varchar(255)
Text	Either: <ul style="list-style-type: none"> nvarchar(n), if up to 4000 characters, or nvarchar(max) 	Either: <ul style="list-style-type: none"> nvarchar2(n), if up to 4000 characters, or nclob 	Either: <ul style="list-style-type: none"> varchar(n), if up to 4000 characters, or clob
Time	datetime	timestamp	timestamp
URI	nvarchar(255)	nvarchar2(255)	varchar(255)

Modeling Case Data in a Global BOM

Use the BOM Editor to model your case data.

Procedure

1. Add at least one [case class](#) to the BOM.








A BOM that contains a case class or a global class is a global BOM. A BOM that contains only local classes is a local BOM.

2. Add suitable [case identifiers](#) to each case class.
3. Define any [case states and case actions](#) that you want for each case class.
4. Modify the [case summary](#) as required for each case class.
5. Define the attributes that you want to be [searchable attributes](#) in case classes and global classes:
 - a) In the BOM Editor, select the attribute.
 - b) On the **General** tab of the **Properties** view, select or clear **Searchable**.
6. Define required [BOM relationships](#).
7. Save the BOM.

Case Classes, Global Classes and Local Classes

A global BOM must contain a case class or a global class. It can also contain, as required, further case classes or global classes, and local classes.

Type	Icon	Description
Case		<p>A case class is a template for a case object. A case class:</p> <ul style="list-style-type: none"> • must have one or more case identifiers. • can have searchable attributes. <p>You can create case objects from case classes using either a case data service task in a process or a public API. Creating a case object returns a case reference. That reference can be used by any process to find and manipulate that case object.</p> <p> Unlike local objects, which exist only over the lifespan of a process instance, case objects are persisted - each case object is stored in the case data tables that represent the case model from which the case object was instantiated.</p>
Global		<p>A global class is a component of one or more case classes. A global class:</p> <ul style="list-style-type: none"> • cannot be created or accessed independently of a case class. <p>For example, a Person global class holds basic details such as name, gender, DOB, which can be used by other global or case classes. However, the Person class doesn't have a unique ID and cannot be directly accessed from a process.</p> <ul style="list-style-type: none"> • cannot have a case identifier. • can have searchable attributes.
Local		<p>A local class is a template for a local object. A local class:</p> <ul style="list-style-type: none"> • cannot have a case identifier. • cannot have searchable attributes. <p> A business object created from a local class only has local scope, even though the class exists in a global BOM. The local object is not persisted to the case data store and can only be accessed by the process that creates it.</p>






You can convert a class from its existing type to either of the other types. Right-click the class and select the appropriate **Convert to Type** option. For example, if you right-click a case class, **Convert to Global** and **Convert to Local** menu options will be available.

Case Identifiers

A case identifier (CID) is a special type of attribute that can be used to uniquely identify a case class. It can be used in processes, scripts or API calls to delete or to find a particular case.

A case class must have at least one case identifier, and can have as many as are required by the nature of the data represented.

There are three types of case identifier:

Type	Icon	Description
Automatic		A unique integer that is automatically generated by ActiveMatrix BPM when an instance of the case class is created. This is the default option.
Custom		A single attribute of the class that has unique values within the business domain. For example, a policy ID could be used as a custom case identifier for a policy class.
Composite		A combination of attributes of the class that together provide unique values within the business domain. For example, customerName and customerDateOfBirth attributes could be used together as a composite case identifier for a customerNotes case class.

Custom or composite case identifiers:

- must be defined as decimal, integer or text primitive types, or user-defined primitive types based on those types.
- must be populated by the process, script or API call that creates an instance of the case class.

Adding a Case Identifier to a Case Class

A case class must have at least one case identifier.

Procedure

1. In the BOM Editor, drag a **Case Identifier** from the Global Data section of the palette to the case class. An automatic case identifier is created by default.
2. On the **General** tab of the **Properties** view, change the **Label** to a suitable name to identify the case identifier.

What to do next

You can:

- add additional case identifiers to the case class by using the same procedure.
- change this case identifier to be a custom or composite type.



You must add at least two **Case Identifier** elements to a case class to be able to create a composite case identifier.



You can also refactor an existing attribute as a case identifier. Right-click the attribute and select **Convert to Case Identifier**. Similarly, you can refactor an existing case identifier as an attribute. Right-click the case identifier and select **Convert to Attribute**.

Changing a Case Identifier's Type

The default case identifier type is automatic (a unique integer that is automatically generated by ActiveMatrix BPM when an instance of the case class is created). You can instead use custom or composite identifier types, which use attribute values as the identifier.

Procedure

1. In the BOM Editor, select the **Case Identifier**.

2. Select the **General** tab of the **Properties** view.
3. Set the **Case Identifier** value to **Auto**, **Custom** or **Composite**.
4. Set the **Multiplicity** value as follows:

Multiplicity Value	Case Identifier Type
0..1	Automatic
1	Custom or Composite

5. For a custom or composite case identifier, in the **Type** field, click **Browse** and select an appropriate primitive type.

Case States and Case Actions

Case states and case actions provide a powerful case management tool, allowing a user to choose the most appropriate actions to perform on a case at a particular point, according to the business context of the case.

Cases (for example, claims or orders) flow through states that are particular and specific to the type of case itself. For example, an order may be in one the states "submitted", "fulfilled", "dispatched" and "delivered". Based on the state of the case, there may be one or more applicable actions that a user might need or want to perform. What actions are required, in what order they are performed or indeed whether an action is performed more than once is at the discretion of the user working on the case. Implementing case states and case actions for a case class gives users the ability to make these decisions and perform the required actions.

At design time, you must:

1. define a case state for the case class.
2. define the set of case actions that are required to support the case.
3. for each case action, define the case states in which it will be available, and the permissions a user needs to be able to use it.

At runtime, when a user has located a particular case, the case actions are displayed that are available to them and appropriate to the current case state. They can then decide which, if any, of these actions they want to perform based on the particular case circumstances.

Case States

A case state is a special type of attribute, available only on case classes, that can be used to uniquely identify a set of business-specific states that a case can be in.

A case class can have only a single case state attribute (optional), which must:

- be a text enumeration type.
- have multiplicity of either 0..1 (optional) or 1 (required).

You can either [create a new case state](#) and add it to a case class or [convert an existing attribute](#) of the case class to be the case state.

Case Actions

A case action is a special type of business service, associated with a case class, that defines an action a user can perform on a case. You can either [create a new case action](#) or [generate one directly from a case class](#). Template case action processes are provided that allow you to view or update the contents of a case object, but you can modify these templates to provide whatever functionality you need for a particular case class.

You can define as many case actions as you need for a particular case class. The set of case actions define all the actions that a user could perform on a case. You can then restrict the availability of these actions by case state and user privileges:

- **Case state:** You can specify that a case action is only available when a case object is in a specific case state (or states). For example, if you have defined a case action to update delivery details for an order, you may want to restrict this to be available only for orders that are "submitted".
- **Privileges:** You can specify that a case action is only available to users who have a specific set of privileges. For example, in a call center, you may want to make certain actions available only to supervisors.

At runtime, a case action can only be invoked for a particular case object, either from the Openspace Case Management gadget or, in a custom client, by using the BusinessService service.

Adding a Case State to a Case Class


Prerequisites

The case class must not already contain a [case state](#).



You can [convert an existing case state back to an attribute](#). Right-click the case state, then select **Convert to Attribute**.

Procedure

1. In the BOM Editor, create a text enumeration that defines the possible valid [case states](#) for this case class.
2. Drag a **Case State**  from the **Global Data** section of the palette to the case class.
3. On the **General** tab of the **Properties** view, change the **Label** to a suitable name to identify the case state.
4. Set the attribute's **Multiplicity** to be either 0..1 (optional) or 1 (required).
5. Select the appropriate **Enumeration Type**.

Converting an Existing Case Class Attribute to a Case State

Prerequisites

- The attribute to be converted must be a text enumeration type containing the desired enumeration literals. (You can convert an attribute that does not meet this requirement, but you must then convert it afterwards. If you are converting an attribute in a BOM that has already been deployed, you must ensure that the conversion is a [non-destructive change](#). If you attempt to deploy an updated case data model that contains destructive or unsupported changes as a new minor version, deployment will fail.)
- The case class must not already contain a [case state](#).

Procedure

1. Right-click the attribute that you want to convert, then select **Convert to Case State**.



If you want to convert a case state back to an attribute, right-click it, then select **Convert to Attribute**.

2. On the **General** tab of the **Properties** view, set the attribute's **Multiplicity** to be either 0..1 (optional) or 1 (required).

Creating a Case Action

Prerequisites

- The Business Data project containing the case data model that you want to use must be open in the same Workspace.
- The case class that you want to associate with this [case action](#) should contain a case state attribute that defines the required [case states](#). (if it does not, you can still create the case class but a problem marker will be displayed.)
- If you want to assign privileges to control the availability of the case action, the Organization project in which those privileges are defined must be open in the same Workspace.

Procedure

1. Open the project in which you want to create the case action.
2. Right-click the process package in which you want to create the case action, or the **Processes** folder within that package, then select **New > Case Action**.
3. In the Case Action wizard:
 - a) Enter a suitable descriptive **Label** for the case action.
 - b) Select the **Case Class** with which you want to associate this case action.
 - c) Select one of the **Case Action Process Templates** to use as the basis for the process - either **Update Case Action Process** (to generate a process that displays a form showing the contents of the case object, then updates the case object with any changes when the form is submitted), or **View Case Action Process** (to generate a process that displays a read-only form showing the contents of the case object).
 - d) Click **Finish**.
The process is displayed in the Process Editor.
4. On the **General** tab of the **Properties** view for the process:
 - a) (Optional) Select the specific case state (or states) which the case must be in for this case action to be available to a user at runtime.
By default, the case action is always available, whatever case state the case is in.
 - b) (Optional) Select the privilege (or privileges) that a user must hold to be able to use this case action.
5. Modify the template process as required so that it performs the required action (or actions) when it is used.
6. Save the process.

Generating a Case Action From a Case Class

Prerequisites

- The BPM Developer project in which you want to save the generated [case action](#) must be open in the same Workspace.
- The case class that you want to associate with this case action must contain a [case state](#) attribute that defines the required case states.
- If you want to assign privileges to control the availability of the case action, the Organization project in which those privileges are defined must be open in the same Workspace.

Procedure

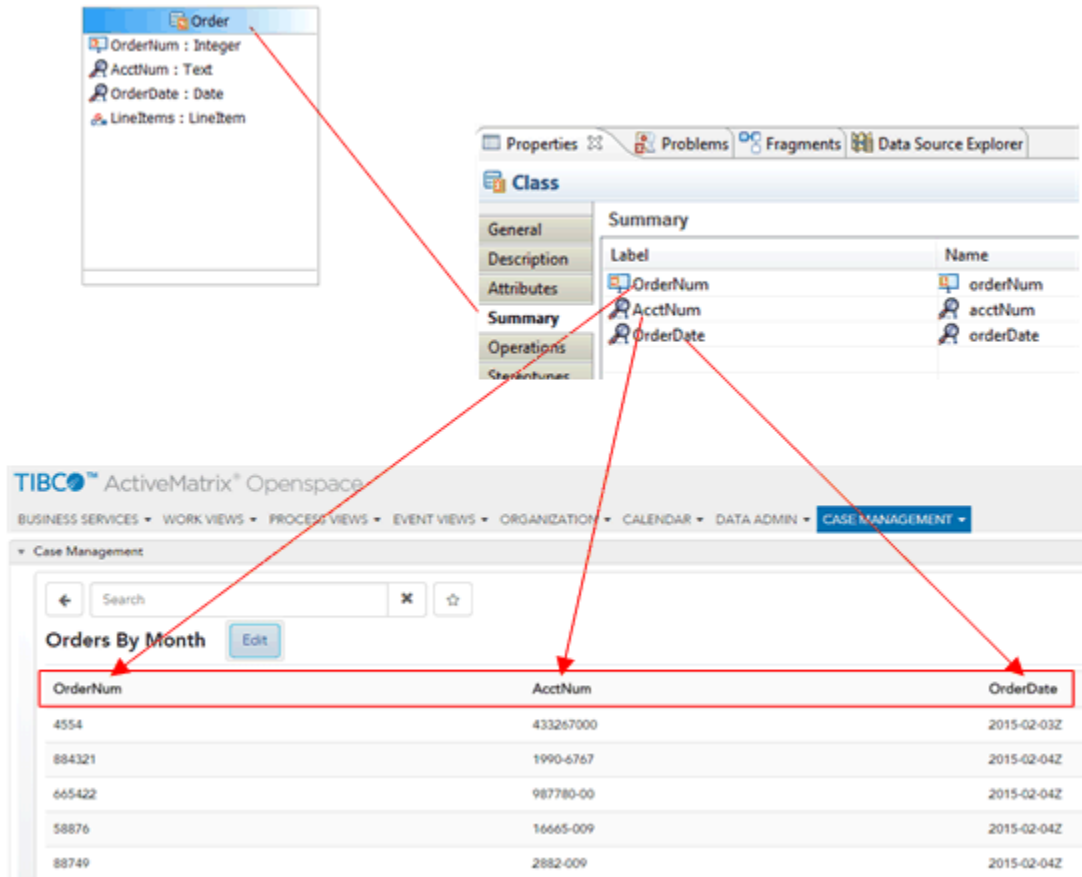
1. In the BOM Editor, right-click the case class from which you want to generate a case action, then select either:
 - **Generate Case Action > To Update Case Data**, to generate a process that displays a form showing the contents of the case object, then updates the case object with any changes when the form is submitted.
 - **Generate Case Action > To View Case Data**, to generate a process that displays a read-only form showing the contents of the case object.
2. In the Generate Case Action dialog, select the appropriate existing process *package*.xpd1 file (or select a process packages folder and enter the name of a new .xpd1 file to create) in which you want to save the generated case action process, then click **Finish**.
The generated process is displayed in the Process Editor.
3. On the **General** tab of the **Properties** view for the process:
 - a) (Optional) Select the specific case state (or states) which the case must be in for this case action to be available to a user at runtime.
By default, the case action is always available, whatever case state the case is in.
 - b) (Optional) Select the privilege (or privileges) that a user must hold to be able to use this case action.
4. Modify the template process as required so that it performs the required action (or actions) when it is used.
5. Save the process.

Case Summaries

A case summary is a specified subset of the attributes defined for a case class. The case summary defines the initial set of information that is returned about a case object when that object is returned as part of a search result.

At design time, you define the attributes that make up a case summary in the BOM Editor.

At runtime, when a user searches the case class for matching case objects (using the Openspace Case Management gadget), the case summary information is displayed for each matching case object. Each attribute is displayed as a column. The attribute's **Label** is used as the column heading. For example:



A case summary includes, by default, any attributes that are defined as case identifiers or case states. You cannot remove these attributes from the case summary. You can add any of the other attributes defined for the case class to the case summary provided that the attribute:

- has multiplicity of either 0..1 or 1.
- is either a primitive type or an enumeration.

You can also define the order in which case summary attributes will be displayed at runtime.

Case summary attributes can also be obtained using the `getCaseSummary` operation from the `BusinessDataServices` service in the BPM Web Service API. See the *TIBCO ActiveMatrix BPM Developer's Guide* for more information.

Adding Attributes to a Case Summary


You can add attributes of a case class to its case summary so that the attribute information is returned in search results for matching case objects.

Prerequisites

The attribute that you want to add must already exist, either directly in the case class or inherited via a generalization relationship. An attribute can only be used in a case summary if:

- It has multiplicity of either 0..1 or 1.
- It is either a primitive type or an enumeration.

Procedure

1. In the BOM Editor, select the required case class.
2. On the General tab of the Properties view, click **Summary**.
3. Click .
4. In the Summary dialog, select the attribute that you want to add, then click **OK**.

The dialog displays all attributes that belong directly to the class, or that are inherited by the class via a generalization relationship. Attributes are displayed whether they are valid for use in a case summary or not. Attributes that are already included in the case summary are disabled.

Result

The attribute is added to the bottom of the list of attributes displayed on the **Summary**. If you have selected an invalid attribute, an error marker is displayed against that attribute in its class on the palette. (If the attribute is inherited, the error marker is displayed against the attribute in its own class, not the class for which you are defining the case summary.)

Deleting Attributes From a Case Summary


You can delete attributes from a case summary so that the attribute information will not be returned in search results for matching case objects.

You cannot delete attributes that are defined as case identifiers or case states.



If you delete an attribute from a case class, the attribute is automatically deleted from any case summary in which it is currently included. You do not have to manually remove it from the case summary as well.

Procedure

1. In the BOM Editor, select the required case class.
2. On the General tab of the Properties view, click **Summary**.
3. Select the attribute that you want to delete.
4. Click .

Result

The attribute is removed from the list of attributes shown on the **Summary**.



Defining the Order in Which Case Summary Attributes are Displayed at Runtime

You can change the order in which attributes are listed in the case summary so that the attribute information is listed in that order in search results for matching case objects.

At runtime, when case summary information is displayed in the Openspace Case Management gadget, each attribute listed in the **Summary** is displayed as a column. Attributes are displayed in the search results left to right, in the order that they are listed on the **Summary**, top to bottom.

Procedure

1. In the BOM Editor, select the required case class.
2. On the General tab of the Properties view, click **Summary**.
3. Select the attribute that you want to move.

4. Click  or  to move the attribute to the desired location in the attribute list.


Searchable Attributes

A searchable attribute is one that is intended to be used in case data search criteria. Searchable attributes can be used in process scripts or BPM public API calls to find cases that match specified attribute values.

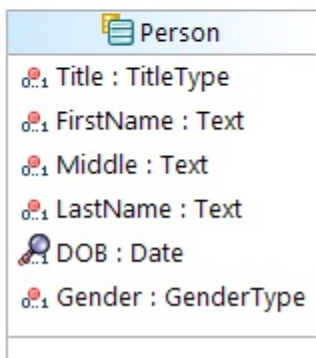
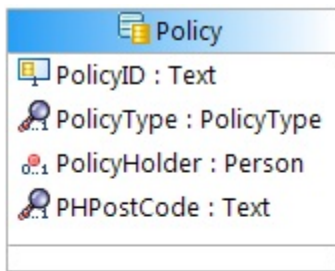
For example, find all policies whose IDs begin with the string PN-456.

Searching is possible using any attribute of an appropriate type, but is much more efficient when using searchable attributes. (In the case data tables that represent the case model, indexes are created for all searchable attributes, to facilitate rapid searching of the database.)

Any appropriately typed attribute of a case class or global class can be defined as searchable. Attributes of local classes cannot be made searchable.

Case identifiers are always searchable. (This setting cannot be changed.) Other attributes can be set to being searchable or not searchable (their default value). A searchable attribute is indicated by the  icon.

For example, in the following case data model fragment, the Policy case class has been optimized to search for policies by a policy ID or type, or by a policy holder's post code or date of birth.



Searching for a policy based on the other PolicyHolder attributes - such as name or gender - is still possible, but may be much less efficient.

BOM Relationships and Global BOMs

Only certain types of UML relationships are permitted when using case data. Whether or not a relationship is supported depends upon its type, direction, the source and target classes, and whether those classes are both in the same global BOM and the same Business Data project.



Bi-directional composition relationships, and uni-directional association or aggregation relationships, are not supported.

Validation errors are displayed if you try to define an invalid relationship between two classes.

Uni-directional Composition

Source Type	Target Type		
	Case	Global	Class (local)
Case	No	Yes	No
Global	No	Yes	No
Class (local)	Yes	Yes	Yes

Bi-directional Association

Source Type	Target Type		
	Case	Global	Class (local)
Case	<p>Yes, provided that both of the following are true:</p> <ul style="list-style-type: none"> The relationship is optional (multiplicity = 0..n). Mandatory associations (multiplicity = 1) are not supported. Both classes are in the same global BOM. Cross-BOM or cross-project associations are not supported. 	No	No
Global	No	No	No
Class (local)	No	No	No

Bi-directional Aggregation

Source Type	Target Type		
	Case	Global	Class (local)
Case	<p>Yes, provided that both of the following are true:</p> <ul style="list-style-type: none"> The relationship is optional (multiplicity = 0..n). Mandatory associations (multiplicity = 1) are not supported. Both classes are in the same global BOM. Cross-BOM or cross-project aggregations are not supported. 	No	No
Global	No	No	No
Class (local)	No	No	No

Generalization

Source Type	Target Type		
	Case	Global	Class (local)
Case	Yes, provided that both classes are in the same project and only the base class contains any case identifiers.	No	No
Global	No	Yes, provided that both classes are in the same project.	No
Class (local)	No	No	Yes

Converting a Local BOM to a Global BOM

You can use a local BOM as the basis for a global BOM.

Procedure

1. Add a local BOM to a Business Data project, using one of these methods:
 - Convert an existing local BOM project into a Business Data project: Right-click the Local Business Object Model project, select **Refactor > Convert to Business Data Project**, then click **OK**.
 - Copy or move a local BOM into the Business Objects folder of an existing Business Data project.
 - Import an external model: Right-click the Business Objects folder in the Business Data project, then select **Database, UML Model, WSDL** or **XML Schema**.
2. To convert a local class to a global class or case class, right-click the local class and select either **Convert to Case** or **Convert to Global**.

You can do this either from the palette, or from Project Explorer.
3. Delete any local classes that you no longer need.
4. Make any other changes you want, then save the BOM.

Upgrading a Case Data Model

When you upgrade a case data model that has already been deployed, each change is validated by the BPM runtime as part of the deployment process (based on either database rules or the ability to support existing data from an earlier version). Whether changes are valid or invalid determines how you can deploy the updated case model.

If your update contains only non-destructive and supported changes you can:

1. increment the *minor/micro/qualifier* components of the project's version number, unless you are allowing Business Studio to automatically manage the version number.
2. deploy the case data model as a new minor version, which upgrades the existing business data application and case data tables.

If you make any destructive changes, you must:

1. increment the *major* component of the project's version number.
2. deploy the case data model as a new major version, which creates a new business data application and a new set of case data tables. The application and case data tables for the existing case data model version are unaffected. (If you do not want to do this, you can delete the existing version, clear the ActiveMatrix Administrator software cache, then deploy the new version with the same version number. However, this will remove any existing data in the case data tables for this case data model.)



Validation is performed as part of the deployment process (by the BDS component of the BPM runtime). If you attempt to deploy an updated case data model that contains destructive or unsupported changes as a new minor version, deployment will fail.



You cannot add (as part of upgrade) a generalization to an existing class such that an existing class becomes the child of another existing or newly added class.

Non-Destructive Changes

The following changes are non-destructive, allowing upgrade to a higher minor version:

- Add a new optional attribute to a class.
- Add a new class.
- Add an optional composition.
- Add an association.
- Change multiplicity upper bounds from 2 or higher to a higher value.
- Change multiplicity lower bounds to a lower value.
- Add a new enumeration literal to an existing enumeration type.
- On an attribute:
 - Increase the length of a text field up to 400 (if it is currently less than 400).
 - Increase the "Integer - Fixed Length" number of digits up to 31.
 - Increase the "Decimal - Fixed Length" number of digits up to 31.
 - Increase the upper bounds of a numeric type.
 - Decrease the lower bounds of a numeric type.
 - Remove restrictions such as upper bound, length, lower bound or pattern.
 - Make it searchable or not searchable.

Destructive Changes

Any change other than those specifically listed as non-destructive is destructive, requiring an upgrade to a higher major version. The following are specific examples of destructive changes:

- Add a generalization to an existing class.
- Change the type of an existing attribute.
- Add mandatory compositions.
- Change multiplicity from 1..1 to 1..*.
- Delete mandatory attributes or compositions.
- Add a mandatory attribute to a class.
- Change multiplicity 0..* to 0..1 or 1..1.
- Delete optional attributes, associations, compositions.
- Delete a class.
- On an attribute:
 - Change between different numeric types.
 - Change a pattern that is applied to the attribute.
 - Decrease the length of a text field.
 - Increase the lower bounds of a numeric type.
 - Decrease the upper bounds of a numeric type.

Case Data Model Versioning

Case data model versions are set on the project level. Correct version management is essential when upgrading case data models or process applications that reference them.

Case data model version numbers are set on the **Properties > Lifecycle** dialog of the Business Data project.

The default format for a case data model version number is:

major.minor.micro.[qualifier]

where:

- *major* defines the major version number of the case data model.
- *minor.micro.[qualifier]* defines the minor version number of the case data model.
- *qualifier* is an optional parameter that, if used, will be replaced by the timestamp value in the **Properties > Lifecycle > Build Information > Build Version** field.
- **Build Version** is a timestamp that is updated whenever the project is updated.



The use of *qualifier* is different in a process project. On a process project version number, *qualifier* is replaced by a timestamp when the DAA is created.

The current case data model version number is used when the Business Data project is deployed, and is visible:

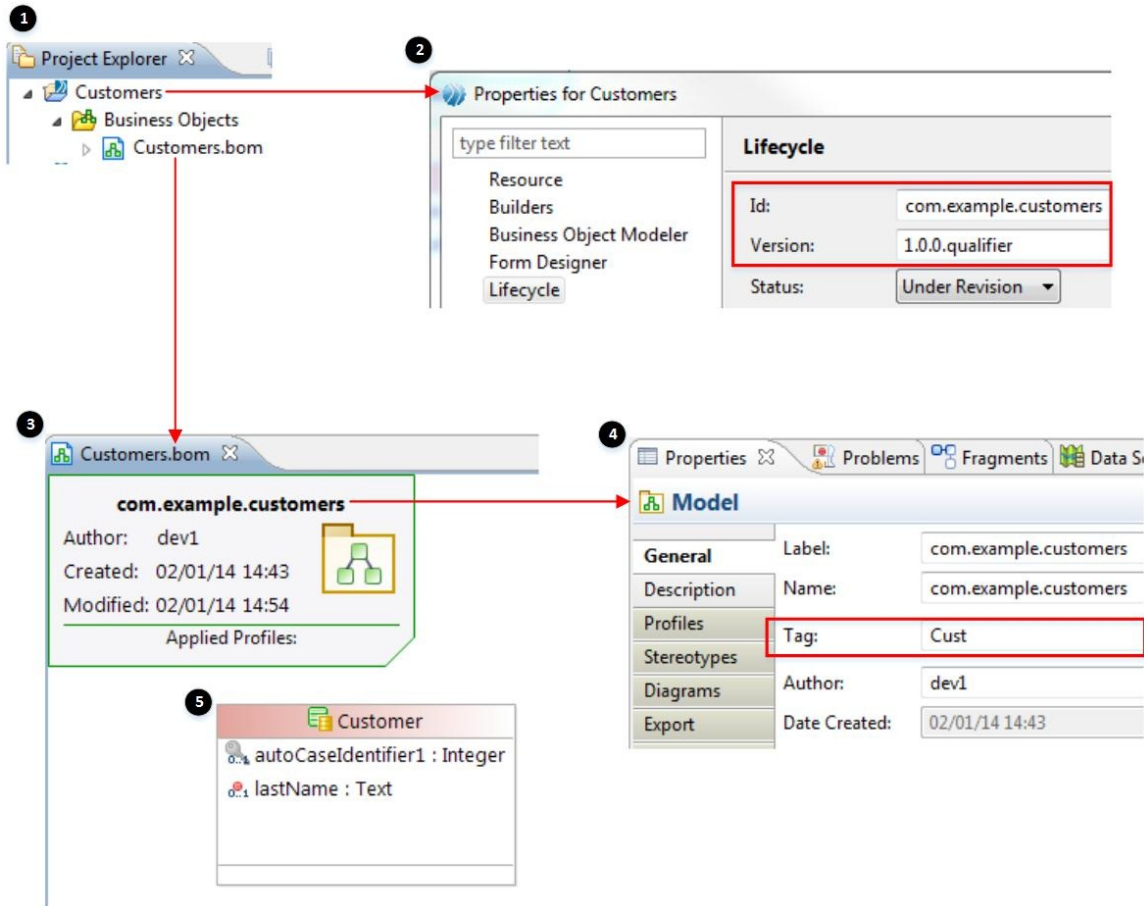
- in TIBCO Administrator, as the business data application's **Application Template Version**.
- in the Openspace Data Admin gadget, as the case model's version number.

When you upgrade a case data model, you must update the Business Data project's major or minor version number, depending on whether you have made destructive or non-destructive changes to the model. See [Upgrading a Case Data Model](#).

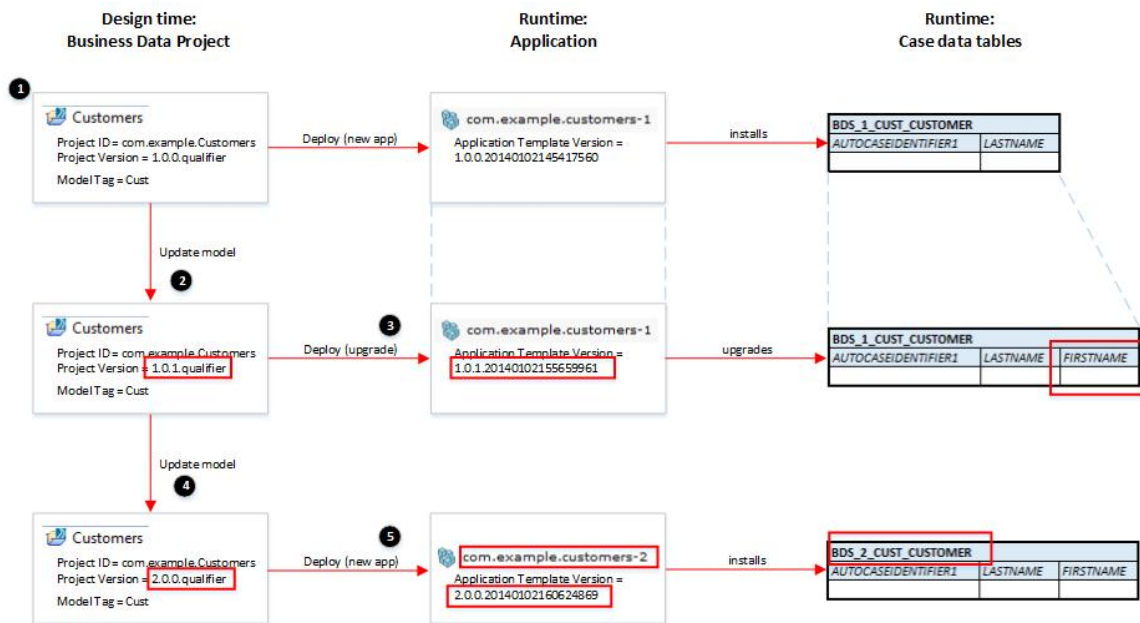
If a process project references the Business Data project, the current case data model version number is used in the reference when the DAA for the process project is generated. This creates an exact-match application dependency from the process application to that version of the business data application.

Consequently, TIBCO recommend that when you upgrade a case data model, you should also upgrade any existing process application that references that case data model - even if that process application makes no use of the updated parts of the case data model. Keeping case data model applications and dependent process applications in step in this way facilitates subsequent deployments or undeployments of either application. See [Process Dependencies and Case Data Models](#).

The following example illustrates how case data model version numbers are constructed, and how they should be managed.



1. Customers is a Business Data project that contains the case data model.
2. The project uses the default **Project Lifecycle** values for **Id** and **Version**.
3. The project includes a single BOM, also called **Customers**.
4. The **Tag** property on the Model defines the prefix that will be applied to all case data tables created for the **Customers** BOM. If **Tag** is empty (the default option), the system decides the value to be used.
5. The **Customers** BOM contains a single case class - **Customer**, which in turn contains an automatic case identifier and a single, optional attribute for the customer's surname.



1. When the **Customers** project is deployed to the BPM runtime:

- By default, Business Studio uses the project **Id** as the application name, suffixed with the major component of the project **Version** number (1).



TIBCO recommend that you do not change the default application name.

- The BPM runtime uses the major component of the project **Version** number (1) and the BOM **Tag** property value (**Cust**) to build the name of the case data table that represents the **Customer** case class.
- The solution designer changes the model, adding an optional first name attribute to the **Customer** case class. Adding an optional attribute to a class is a non-destructive change to an existing model, so he updates the project's version number to 1.0.1.qualifier. (If he does not do this, the version number is updated automatically anyway.)
 - He now deploys the project. Because the major version and project Id are unchanged, this is a minor upgrade to the existing application and case data table.
 - The solution designer now makes a further change to the model, changing **lastName** to be a compulsory attribute. This constitutes a destructive change to the model, so the major component of the project **Version** number must be updated (to 2).
 - Because the major component of the version number has been updated, deploying the project results in the creation of a new application and a new case data table.

Process Dependencies and Case Data Model Versions

A process that references an element in a case data model has an application dependency on the exact version of the referenced case data model. It is important to understand the implications of this exact-match relationship, and to manage those implications when upgrading either a case data model or a dependent process. Not doing so could result in deployment errors or runtime issues.

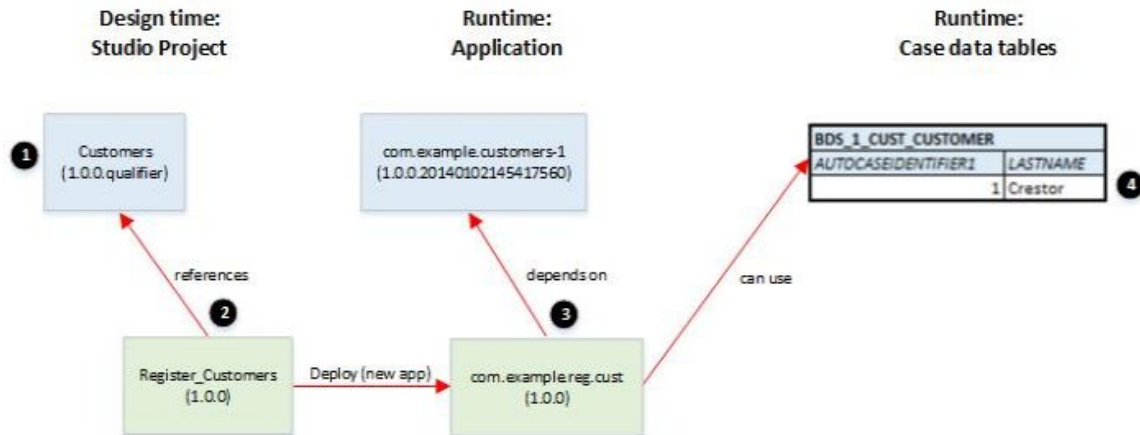
When you deploy (or generate the DAA for) a process project that references a data element in a Business Data project:

- The process application has an application dependency on the referenced business data application. This dependency is recorded against the exact version number of the Business Data project that exists when the process application is deployed (or the DAA generated).

- That version of the case data model must be deployed before you can deploy any process project that references it. Attempting to deploy the process project first will result in a deployment error.
- Process instances can only access the case data tables and case data specifically defined by the version of the case data model on which they are dependent.

Consequently, TIBCO recommend that you keep version numbers of case data models and dependent process projects in step with each other. Whenever you upgrade a case data model, you should review the impact on all process projects that reference that case data model, and upgrade process projects as required.

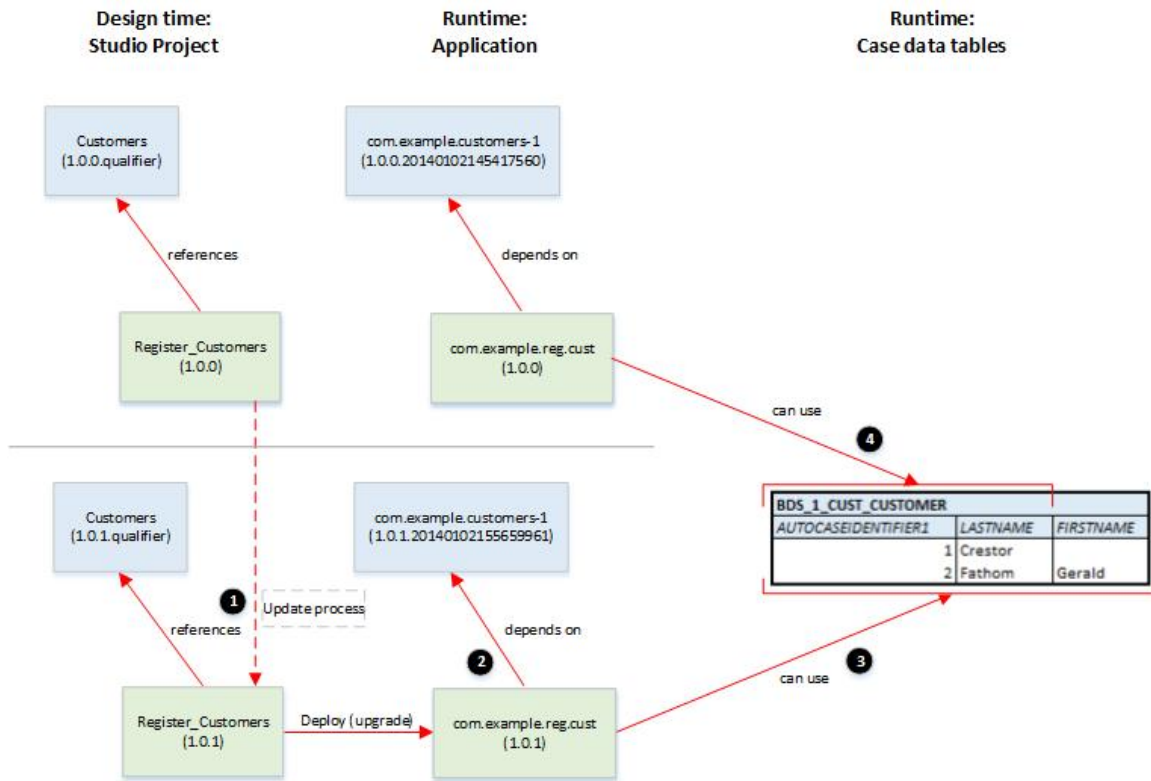
The following example illustrates how successive updates to the very simple Customers case data model (described in [Case Data Model Versioning](#)) could impact an equally simple process project that uses that case data model, and how a solution designer could manage that impact.



1. The solution designer creates and deploys the Customers Business Data project.
2. He creates a process project, Register_Customers, that uses the Customer case class provided in the Customers Business Data project.
3. He deploys the project, creating the process application com.example.reg.cust. Because it uses the Customer case class, this application has a dependency on the 1.0.0.qualifier version of the com.example.customers-1 business data application.
4. He runs an instance of the Register_Customers process, which creates a customer object in the BDS_1_CUST_CUSTOMER case data table.

Upgrading the Process Application Following a Minor Upgrade of the Case Data Model

The solution designer subsequently updates the Customers project, adding an optional firstName attribute to the Customer class.



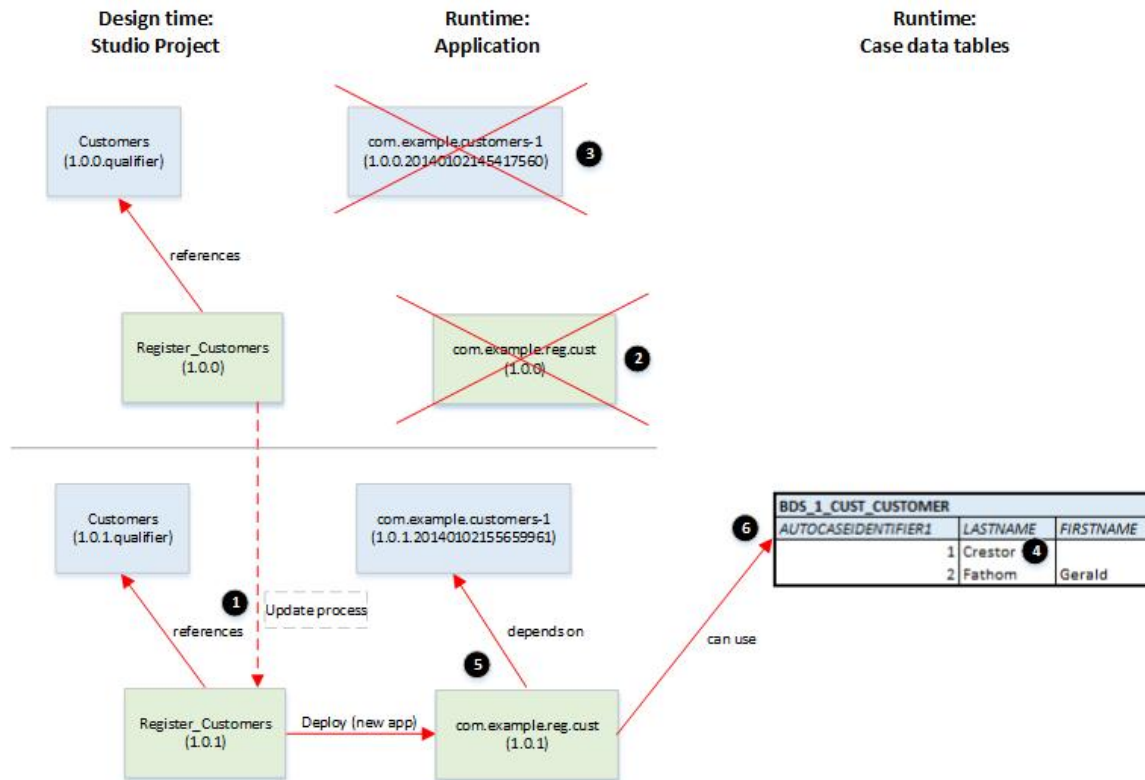
1. The solution designer deploys the Customers project.
2. He then updates the Register_Customers process to make use of the new firstName attribute on the Customer case class.
3. He deploys the process as an upgrade to the com.example.reg.cust application. This version of the application has a dependency on the upgraded version of the com.example.customers-1 business data application.
4. He runs an instance of the upgraded Register_Customers process, which creates a customer object in the BDS_1_CUST_CUSTOMER case data table (for the customer Gerald Fathom).
5. Note that existing process instances of the original version of the com.example.reg.cust application continue to run and use the same case data table, but cannot access the FIRSTNAME data values. For example, if a later task in the Register_Customer process returns details of a customer based on search criteria applied to the LASTNAME, the following table shows the different search results that would be returned by each version of the process, even though both versions are using the same search string and accessing the same case data table.

Search string	Register_Customers process version	Data returned
Crestor	1.0.0	Crestor
	1.0.1	Crestor
Fathom	1.0.0	Fathom
	1.0.1	Gerald Fathom



This scenario is only possible if the Register_Customer process uses the firstName attribute in a way that does not affect the process' service interface. If, for example, the process wanted to use the customer's first name as an input parameter, this would change the service interface and application upgrade would be impossible.

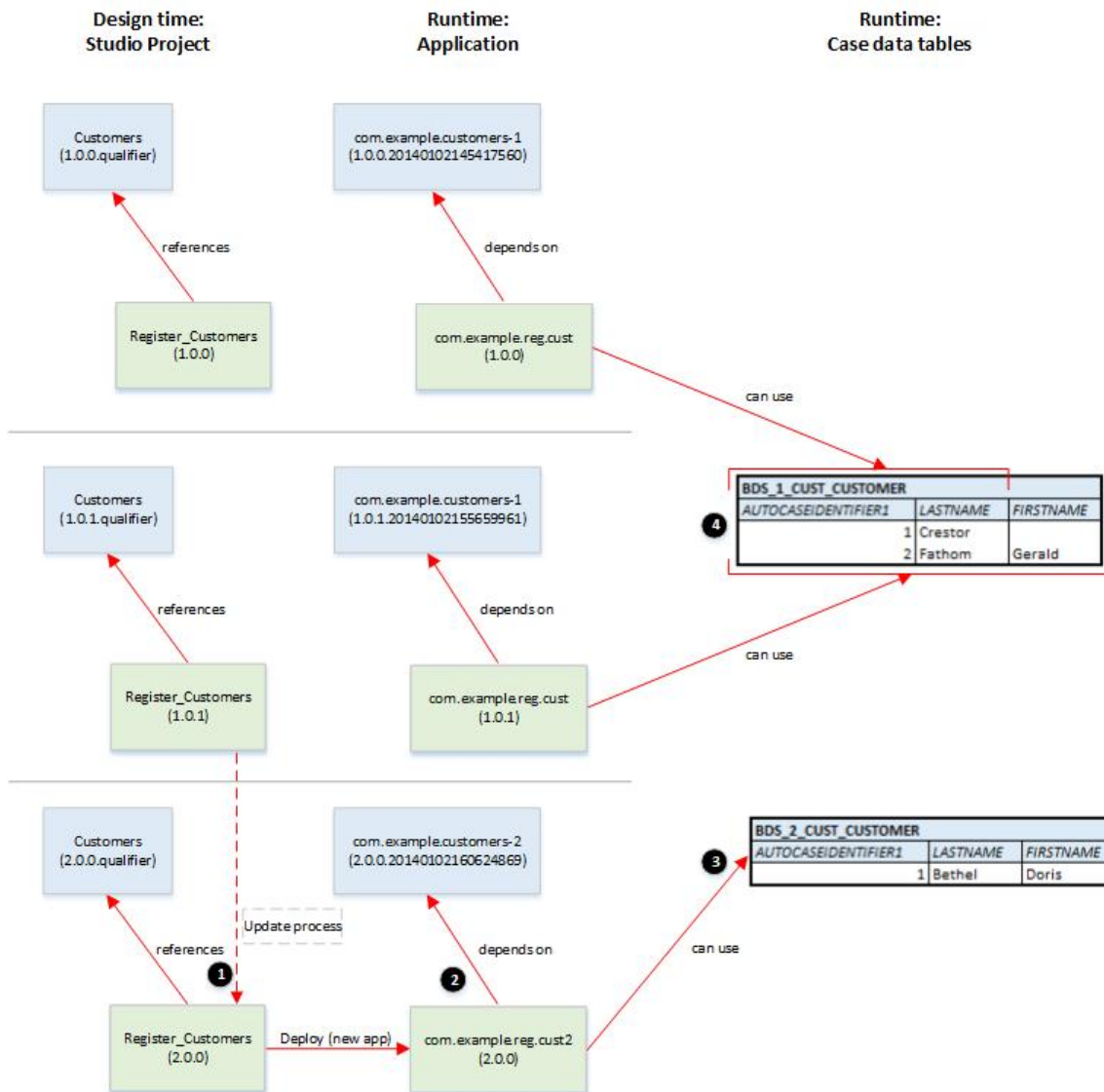
Upgrading the Process Application Following a Minor Upgrade of the Case Data Model that Breaks the Process' Service Interface



1. The solution designer updates the Register_Customers process to make use of the firstName attribute, using it as an input parameter to the process. However, this changes the process' service interface, which means that the process application cannot be upgraded.
2. He cancels all in-progress instances of the version 1.0.0 application, then undeploys and deletes the application.
3. Deleting the process application removes the dependency on the original version of the com.example.customers-1 business data application. As a later version of this application has already been deployed, the BPM runtime now deletes the 1.0.0 version.
4. Although the 1.0.0 versions of both the process and business data applications have been deleted, the data written by those applications to the BDS_1_CUST_CUSTOMER case data table (the Crestor record) is not deleted.
5. The solution designer can now deploy the Register_Customers process as a new application.
6. He then runs an instance of the Register_Customers process, which creates a customer object in the BDS_1_CUST_CUSTOMER case data table (for the customer Gerald Fathom).

Upgrading the Process Application Following a Major Upgrade of the Case Data Model

The solution designer again updates the Customers project, changing the firstName attribute to be a mandatory attribute of the Customer class. This constitutes a destructive change to the model, so the major component of the project Version number must be updated (to 2). Deploying the project again results in the creation of a new application and a new case data table.



1. The solution designer updates the Register_Customers process to handle the use of the mandatory Customer.firstName attribute.
2. He then deploys the process as a new application, which has a dependency on the com.example.customers-2 application.
3. Running an instance of the Register_Customers process creates a customer object in the BDS_2_CUST_CUSTOMER case data table (for the customer Doris Bethel).



The com.example.reg.cust2 application has no access to the data in the BDS_1_CUST_CUSTOMER case data table, and the com.example.reg.cust application similarly has no access to the data in the BDS_2_CUST_CUSTOMER case data table.

Upgrading a Case Data Model to a New Minor Version

When you upgrade an already deployed case data model, you can deploy it as a new minor version if you have made only non-destructive changes to the model.

If any BOM contains destructive changes, you must deploy the case data model as a new major version instead. See [Upgrading a Case Data Model to a New Major Version](#)



If you attempt to deploy a case data model containing destructive changes as a new minor version, deployment will fail.

Prerequisites

- Make sure that the Business Data project has the desired version number (with the same major component of the version number as the currently deployed version). See [Case Data Model Versioning](#).
- Using the Openspace Data Admin gadget, make sure that the existing version of the case model is not frozen. You cannot upgrade a frozen case model.
- TIBCO recommend that you only attempt to upgrade a case data model to a new minor version when the BPM system - in particular, any processes that use the case data tables that are being upgraded - is not under heavy load. (If these case data tables are involved in significant numbers of transactions, the upgrade process may fail because it cannot access the required tables to upgrade them within the required timeframe.)

Procedure

1. Use your preferred application deployment method to deploy the Business Data project.
2. In ActiveMatrix Administrator, verify that the upgraded version of the business data application is **Running and In Sync**.
3. Using the Openspace Data Admin gadget, verify that the upgraded version of the case model is **INSTALLED**.

This indicates that the case data tables have been successfully updated.



If ActiveMatrix BPM is configured to just generate SQL scripts when a case data model is deployed, a DBA will need to obtain and manually run the script to update the case data tables, then notify the BPM runtime that this has been done. See "Obtaining UPDATE Case Data Database Scripts" in the *TIBCO Openspace User's Guide*.

Result

You can now deploy and run any process applications that use this version of the case data model.

What to do next

You should upgrade any existing process application that references an earlier minor version of this case data model to use this version instead - even if that process application makes no use of the updated parts of the case data model. Keeping case data model applications and dependent process applications in step in this way facilitates subsequent deployments or undeployments of either application. See [Process Dependencies and Case Data Models](#).

Upgrade Error: Incompatible changes made during upgrade of version *number* of model

This error occurs if you try to upgrade a case data model that contains destructive changes as a new minor version. To recover, you must revert to the old version of the business data application, then correct and redeploy the new version of the case data model.

If you try to upgrade a case data model that contains [destructive changes](#), deployment of the business data application fails with an error like this:

```
java.lang.Exception: Incompatible changes made during upgrade of
version 1.1.0.20140110110801325 of model
com.example.secondcasemodel-1 Details: The following unsupported
changes were made: 1. [Addition] Mandatory Attribute
"requiredattribute" added
```

The `Details` section in the message provides more information about the type of incompatible changes that have been detected.

In TIBCO Administrator, the business data application's status changes to **Configure failed**.

Procedure

1. In TIBCO Administrator, select the business data application.
2. Click **Undeploy > Force undeploy**.
The application's status changes to **Running**, but **Out of Sync**.
3. In the **Application Template Version** field, select the previous version of the application, then click **Save** in the **Effects of updating to version *number*** dialog.
4. Click **Deploy**.
The business data application's status changes to **Running** and **In Sync**. The previous version of the case data model is now restored.
5. Click **Infrastructure > Software Management > Application Templates**, and **Delete** the failed version of the business data application.

What to do next

In TIBCO Business Studio, either:

- Remove the destructive changes from the case data model and redeploy it as a new minor version. (You can re-use the version number that failed to upgrade if you wish.)
- If you need the destructive change, change the major version number and [redeploy the project as a new major version](#).

Upgrading a Case Data Model to a New Major Version

When you upgrade an already deployed case data model, you must deploy it as a new major version if you have made any destructive changes to the model.

You should only redeploy an existing case data model as a new major version if you have made destructive changes to the model. See [Upgrading a Case Data Model](#). If your changes are purely non-destructive, you may upgrade to a new minor version instead. See [Upgrading a Case Data Model to a New Minor Version](#).

Prerequisites

Make sure that the Business Data project that contains the case data model has the desired version number (with a higher major component of the version number than the currently deployed version). See [Case Data Model Versioning](#).

Procedure

1. Use your preferred application deployment method to deploy the Business Data project to the BPM runtime. See "Deploying BPM Applications" in the *TIBCO ActiveMatrix BPM – BPM Deployment* guide.



TIBCO recommend that you do not change the default application name used for the case data model. This name is *projectId-projectMajorVersionNumber*. For example:

```
com.example.simpleCaseModel-2
```

2. In ActiveMatrix Administrator, verify that the new business data application is **Running** and **In Sync**.
3. Using the Openspace Data Admin gadget, verify that the case model is either **INSTALLED** (if table management is automatic) or **PENDING_DBA_ACTION_TO_INSTALL** (if table management is manual).

This indicates that the required case data tables have been successfully created.



If ActiveMatrix BPM is configured to just generate SQL scripts when a case data model is deployed, a DBA will need to obtain and manually run the database script to create the case data tables, then notify the BPM runtime that this has been done. See "Retrieving CREATE Case Data Database Scripts" in the *TIBCO Openspace User's Guide*.

Result

You can now deploy and run any process applications that use this version of the case data model.

Earlier major versions of the case data model are not affected. Process applications that reference an earlier major version of this case data model will continue to run against that version, and cannot create or access case objects in this new version. This is because each major version of the case data model is a separate business data application with its own set of case data tables. See [Process Dependencies and Case Data Models](#).

Deleting a Case Data Model


You can delete a case data model that you no longer need by first undeploying or force undeploying, then deleting, the appropriate business data application. (You use undeploy if the associated case data tables are empty. If data is present, you must use force undeploy.)

If Business Data Services is managing the database automatically, deleting a case data model deletes the associated case data tables and, therefore, all the case data in them. If you want to delete a case data model (that contains data) without deleting the associated case data tables, contact TIBCO Support for assistance.

Prerequisites

- Complete or cancel all outstanding work items and process instances that belong to any process application that references this case data model.
- Undeploy all process applications that have an application dependency on this business data application.
- Using the Openspace Data Admin gadget, make sure that the live version of the case model is not frozen. You cannot delete a frozen case model.

Procedure

1. In TIBCO Administrator, select the business data application.
 2. Click:
 - **Undeploy** if the case data tables associated with this case data model contain no data.
 - **Undeploy > Force undeploy** if the case data tables associated with this case data model contain data.
- 
- If you just click **Undeploy** by mistake the application's state changes to **Preparing for undeploy** with an **Action History of In Progress (Undeploy)**. This indicates that the BPM runtime cannot complete undeploying the application until all data in the associated case data tables has been deleted. While the application is in **Preparing for undeploy** state, the BPM runtime will periodically check to see if the data has been deleted. Alternatively, you can click **Undeploy > Force undeploy** to force the deletion to take place.
3. Using the Openspace Data Admin gadget, check that the case model is in the expected state:
 - If ActiveMatrix BPM is configured to automatically manage case data tables, the case model should no longer be listed. This indicates that the case data tables have been successfully deleted.
 - If ActiveMatrix BPM is configured to just generate SQL scripts when a case data model is undeployed, the case model should be listed with a state of `PENDING_DBA_ACTION_TO_UNINSTALL`. A DBA will need to obtain and manually run the database script to delete the case data tables, then notify the BPM runtime that this has been done. See "Obtaining DROP Case Data Database Scripts" in the *TIBCO Openspace User's Guide*.
 4. In TIBCO Administrator, select the business data application, then click **Delete**.
(You will be prompted to also delete any dependent process applications.)

Using a Case Data Model in a Business Process

Within a process, you can create, read, update and delete case objects created from the case classes defined in a referenced case data model.

A case object is an instance of a case class. Data for a case object is stored in the case data tables associated with the case data model to which the case class belongs. You can create case objects from within a process by using a Global Data service task.

A case reference is a unique reference (or pointer) to a case object, created by ActiveMatrix BPM when the case object is created. You can use a case reference within a process to find, display, update or delete an existing case object.

Creating a New Case Object

Within a process, you can create a case object by first creating a local BOM object from the case class, then using a global data service task to create the case object from the local data (by adding the case object to the case data store).

Prerequisites

The Business Data project that contains the case class that you want to use must exist in the same workspace as the process from which you want to create the case object.

Procedure

1. Create a new data field, of type **BOM Type**, which references the appropriate case class.
2. In a script, use the appropriate factory method to create an instance of the case class as a local business object and assign it to the data field. (See [Creating a Case Data Model](#).) For example:


```
cust = com_example_scriptingguide_Factory.createCustomer();
```
3. Populate the local business object with the appropriate data. You must assign a value for each mandatory attribute in the case class.



If the case class uses an automatic case identifier you cannot (and do not need to) assign a value for it. You must assign values for custom or composite case identifiers.

4. Add a service task to the process.
5. On the **General** tab of the **Properties** view, set **Service Type** to **Global Data Operations**.
6. Select **Create Case Object(s) or Delete Case Object(s) by Case Identifier**.
7. In the **Case Class** field, select the same case class that you used to create the local business object.
8. In the **Local Data Value(s)** field, enter the name of the data field that holds the local business object.
9. Create a new data field, of type **Case Class Reference**, which references the same case class.
10. In the **Return Case Reference(s)** field, enter the name of this field.

When the service task is executed, the case reference for the newly created case object will be returned to this field.

Result

ActiveMatrix BPM now has two copies available of the same data - the local BOM object and the case object. These objects are separate entities and will not be maintained in step:

- The local BOM object is only visible to the process and can only be manipulated in and by that process.
- The case object is visible to and can be manipulated by any process that has access to its case data model.

Any updates to the case object are *not* automatically reflected in the local BOM object, and vice versa.

What to do next

(Optional) Add appropriate error handling to the service task boundary to deal with the following specific error that may be returned by the **Create Case** operation.

Error (Error Code)	Description	Possible solutions
NonUniqueCaseIdentifierError (NonUniqueCIDError)	The (custom or composite) case identifier specified in the Local Data Value(s) field cannot be used because it is not unique. An existing case object already uses this case identifier.	Amend the case identifier value(s) in the local data object and retry the Create Case operation.

Displaying a Case Object in a User Task - Using a Form

You can display a case object in the form that is displayed when a user performs a user task. You can also, optionally, automatically update the case object with any changes made by the user when the user closes or submits the form.

Prerequisites

An already populated data field must exist, containing a case reference to the case object that you want to display.

Procedure

1. Select the user task from which the form will be displayed.
2. On the **Interface** tab of the **Properties** view, make sure that the appropriate **Case Reference** data field is included in the **Parameters** list, either explicitly or as part of the **[All Process Data]** setting.
3. If you want to automatically update the associated case object when the form is closed or submitted, set the data field's mode to **IN/OUT** and click **Mandatory**.

Result

At run time, when the form is displayed, the referenced case object is retrieved from the case data store and displayed in the form.



Association and aggregation relationships are ignored when the form is generated.

When the form is closed or submitted, any changes made to fields that are part of the case object are written back to the case data store.

Displaying a Case Object in a User Task - Using a Pageflow

You can display a case object as part of a pageflow that is displayed when a user performs a user task. If the user updates data for the case object, you will need to manually update this data in a subsequent process step.

Prerequisites

An already populated data field must exist in the business process, containing a case reference to the case object that you want to display.

Procedure

1. Select the user task from which the form will be displayed.
2. On the **Interface** tab of the **Properties** view, make sure that the appropriate **Case Reference** data field is included in the **Parameters** list, either explicitly or as part of the **[All Process Data]** setting.
3. On the **General** tab of the **Properties** view, click **Pageflow**, select the pageflow process that you want to use, then click **Open PageFlow**.

Make sure that the pageflow process contains the **Case Reference** data field as a formal **Parameter**.

4. Create a new data field, of type **BOM Type**, which references the appropriate case class.
5. Add a script task that uses the read method available on the case reference to read the associated case object data and assign it to the local business object.

For example, the following script reads the data from the case object referenced by `custRef` and assigns it to the `cust` local business object:

```
cust = custRef.readCustomer();
```

This creates a local business object that contains a copy of the case object and is scoped to the pageflow process.

6. Add a user task that displays the local business object in a form, allowing the user to view or update the local business object as required.
Association and aggregation relationships in the case object are ignored when the form is generated.
7. If necessary, add a service task (and, optionally, appropriate error handling) to update the case object with the changes to the local business object - see [Updating a Case Object from a ServiceTask](#).

Updating a Case Object

From within a process, you can use scripting to update a local BOM object of the appropriate case class, then use a global data service task to update the case object from the local data.



You do not need to follow this procedure if you use a case object as an **IN/OUT** parameter on a form in a business process. In that case, changes made by the user are automatically written back to the case data store when the user closes or submits the form. See [Displaying a Case Object in a User Task - Using a Form](#).

However, you do need to follow this procedure if you display a case object as part of a pageflow process. See [Displaying a Case Object in a User Task - Using a Pageflow](#).

Prerequisites

The process must already contain:

- A data field containing a case reference to the case object that you want to update.
- A local business object created from the appropriate case class, containing the data that you want to write to the case object.

Procedure

1. Add a service task at an appropriate point in the process.
2. On the **General** tab of the **Properties** view, set **Service Type** to **Global Data Operations**.
3. Select **Change or Delete Case Object(s) Using Case Reference Field**.
4. In the **Case Reference** field, select the data field that contains the case reference to the case object that you want to update.

Content assistance is available on the **Case Reference** and **Local Data Value** fields. Click the lightbulb icon or press CTRL+space to display a list of valid field names that are available to the task and that begin with the characters to the left of the cursor.

5. In the **Operation** field, select **Update Case Object(s) From Local Data**.
6. In the **Local Data Value(s)** field, enter the name of the data field that holds the local business object. If the selected **Case Reference** field is an array then the selected **Local Data Value** field must also be an array, with an element corresponding to each case reference array element.

What to do next

1. (Optional) Add appropriate error handling to the service task boundary to deal with the following specific errors that may be returned by the **Update Case Object(s) From Local Data** operation.

Error (Error Code)	Description	Possible solutions
CaseOutOfSyncError (CaseOutOfSyncError)	The case object referenced from the Case Reference field has been modified in some way since the field was last populated. That is, the case reference is stale. This could be either a change to the data held in the case object, and/or a change in the object's internal version number.	<ol style="list-style-type: none"> 1. Re-read the case object. 2. If necessary, examine the case data to check that no significant changes have occurred. 3. Retry the Update Case Object(s) From Local Data operation.
UserApplicationError (CaseNotFoundError)	The case object referenced from the Case Reference field does not exist.	
UserApplicationError (UpdateToNullError)	The case object referenced from the Case Reference field has not been initialized (has a null value).	

2. (Optional) Modify any process that may need to do something if this case object is updated:
 - a. Add a [case data signal event](#) to the process, so that the process will be notified when the case object is updated.
 - b. Add suitable business logic to perform any actions that the process needs to take as a result of the update to the case object.

Notifying a Process That a Case Object It Uses Has Been Modified

You can add a case data signal event handler to a process so that it can subscribe to a particular case object that the process uses. The process will then be notified if that case object is modified, and can take appropriate action to respond to the change.

At runtime, the case data signal is:

- initialized when the case reference field is first assigned.
- re-initialized if the case reference field is changed to reference a different case object.
- uninitialized if the case reference field is set to null.


While the case data signal is initialized, the event is triggered whenever the referenced case object is either updated, linked to or unlinked from other case objects, or (in some circumstances) deleted.

Prerequisites

The process must already contain a data field of type **Case Class Reference** that identifies the case object to which the case data signal event handler will subscribe.

Procedure

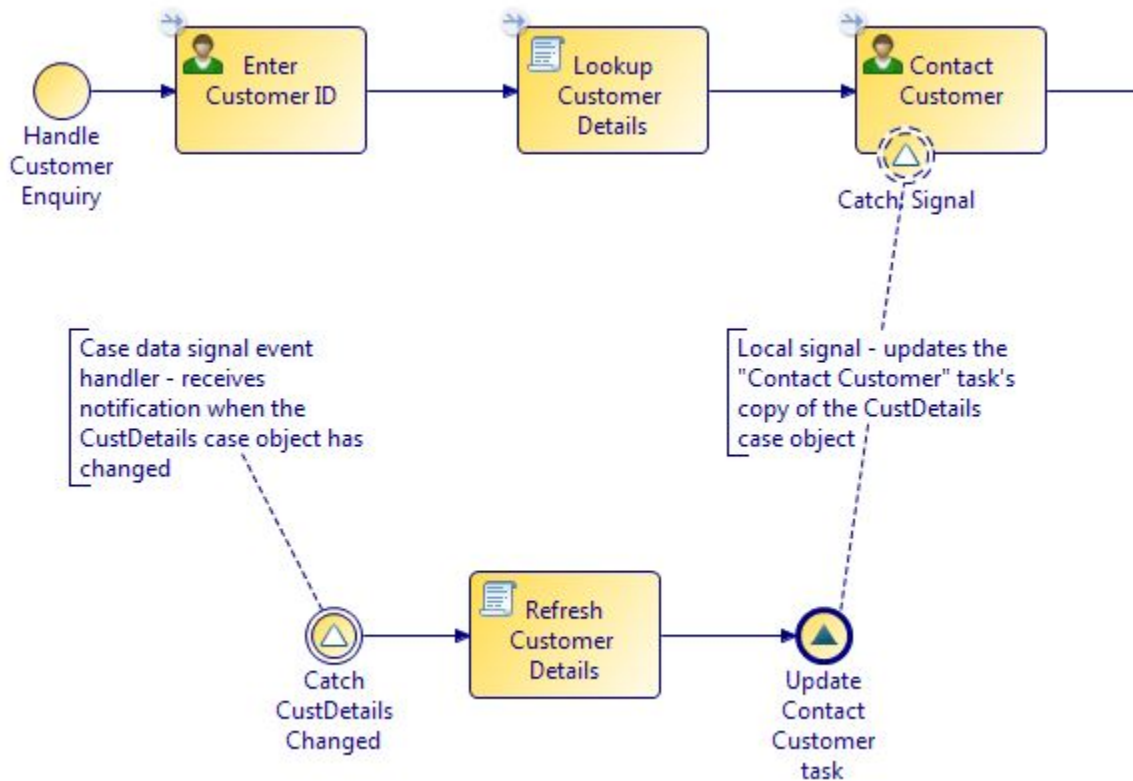
1. Either:
 - Add a catch signal intermediate event to the process (if you want to use a signal event handler).
 - Add an event sub-process to the process, then add a signal start event to the sub-process (if you want to use a signal event sub-process).
2. On the **General** tab of the signal event's **Properties** view, click **Case Data** as the **Signal Type**, then enter the name of the process data field that will contain the case reference for the required case object.



Selecting **Case Data** also sets the **Serialize concurrent flows for this event handler** option. Concurrent execution of the event handler flow is disabled. (This behavior would be undesirable if local copies of the referenced case object are being updated as a result of the event, which is likely.)
3. Add a script task that uses the [CaseSignalAttributes](#) class and methods to determine how the referenced case object has been modified - whether it has been updated, linked to or unlinked from other case objects, or (in some circumstances) deleted.
4. Add appropriate business logic to the process to deal with the change to the case object.

Example

The following process fragment is part of a business process that handles a customer enquiry.



The main **Handle Customer Enquiry** process flow works like this:

1. A customer support representative (CSR) dealing with the enquiry enters the customer's ID, which is then used in the **Lookup Customer Details** script task to retrieve a case object containing the customer's details:

```
// Get the case reference to the CustDetails class that matches the
// identifier provided in the Customer ID (custID) field.
custRef = cac_com_example_caseclass_CustomerDetails.findByCustID(CustID);

// Read the case reference to get the corresponding CustomerDetails case data
// object.
CustDetails = custRef.readCustomerDetails();
```

2. The customer's details are displayed to the CSR in the **Contact Customer** task, which is offered to the CSR pool of workers.

To cater for the possibility that a customer's details could change in between the time that a **Contact Customer** work item is scheduled and a CSR opening and completing that work item, a case data signal event handler is implemented:

1. The **Catch CustDetails Changed** event subscribes to the `custRef` case reference. It is triggered whenever the referenced `CustDetails` case object changes.
2. The **Refresh Customer Details** script task interrogates the received event to find out if it was caused by an update. If it was, it re-reads the customer's details from the referenced case object and updates the local `CustDetails` data field.

```
if
    // Check if the change that triggered the event was an update to the case
    // object
```

```

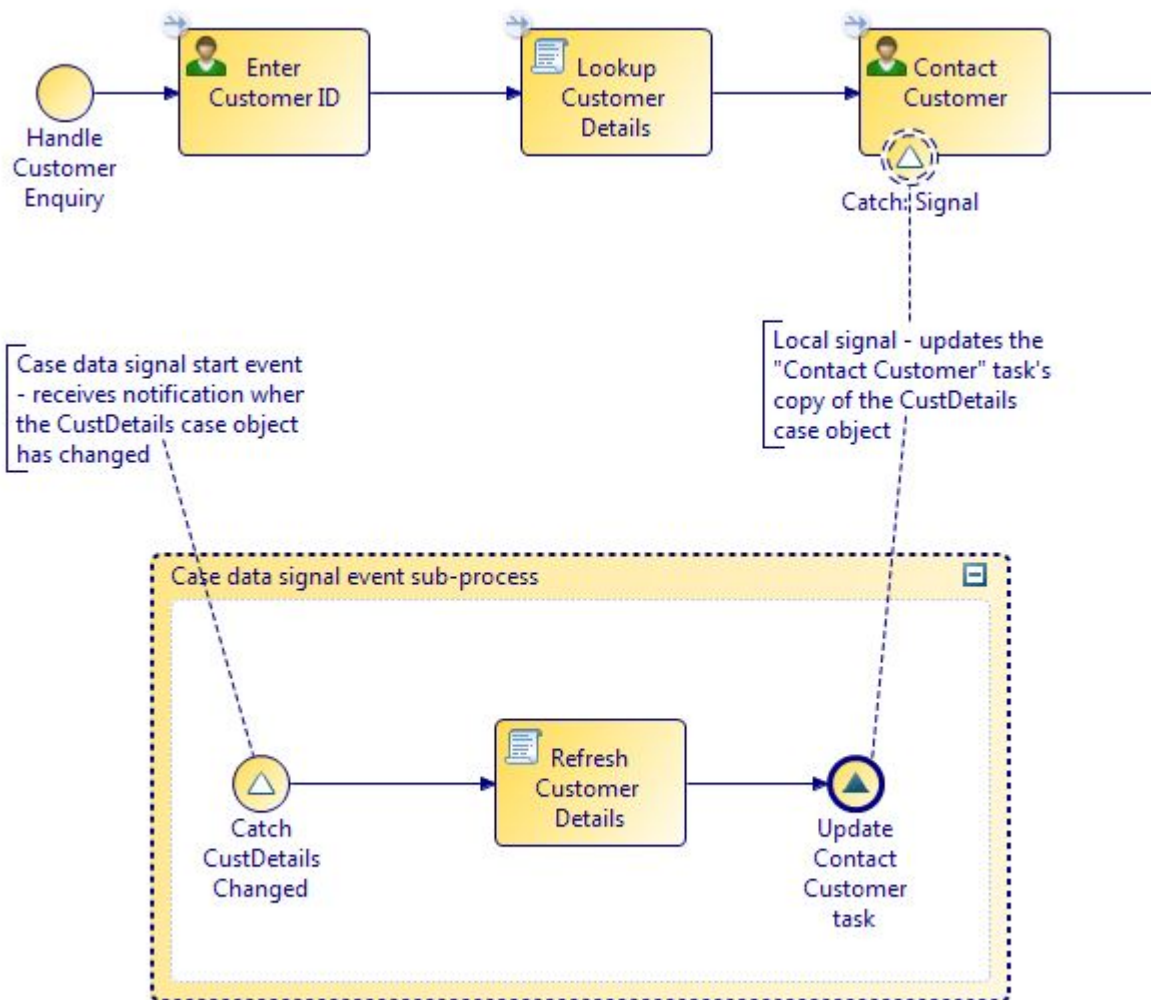
    (CaseSignalAttributes.isCaseUpdate())
    {
        CustDetails = custRef.readCustomerDetails();
    }
else
    {
        // Optionally perform other activities to handle the event if the
        case object has been
        // linked to or unlinked from other case objects, or deleted
    }
}

```

3. The **Update Contact Customer** task throw signal event sends the updated CustDetails data field to its corresponding **Catch Signal** event, which is configured to update that data in the **Contact Customer** work item.



You can use a signal event sub-process instead of a signal event handler, as shown below.



CaseSignalAttributes Class and Methods

As part of a case data signal event handler flow, you can use the methods available on the `CaseSignalAttributes` class to determine whether the referenced case object has been updated, linked to or unlinked from other case objects, or (in some circumstances) deleted.

The `CaseSignalAttributes` class is available in any script in a case data signal event handler flow (or in a case data signal event sub-process).

Method	Description
<code>isCaseUpdate()</code>	Returns <code>true</code> if one or more of the referenced case object's properties has been updated.
<code>isCaseAssociation()</code>	Returns <code>true</code> if one or more association links to or from the referenced case object has been created or deleted. Use the <code>getChangedAssociationName()</code> method to obtain the name of the created or deleted association link(s).
<code>isCaseDelete()</code>	Returns <code>true</code> if the referenced case object has been deleted. See Reasons to Avoid Deleting Case Objects .
<code>isUnknownChange()</code>	Returns <code>true</code> if the BPM runtime cannot determine whether the referenced case object has been updated, deleted, or had an association link created or deleted. This can happen if the change occurs after the case reference used by the case signal is set, but before the event handler is initialized.
<code>getChangedAssociationName()</code>	Returns the name of the association link (to or from the referenced case object) that was created or deleted. Use this method with <code>isCaseAssociation()</code> .

Deleting Case Objects

Within a process, you can use a global data service task to delete one or more case objects. You can identify the case object(s) to be deleted using either case reference(s) or case identifier(s).

Reasons to Avoid Deleting Case Objects

The best practice is to *not* delete case objects as part of a normal operation. If you *do* delete case objects, the best practice is to only delete using a single case reference from within a service task in a business process.

The primary reason to not delete case objects is that if there are other processes (other than the process performing the deletion) that have a reference to the deleted case object, those processes become halted and cannot proceed. In addition, case history (audit trail) is constructed using case objects; if those objects are deleted, the history is no longer available.

Therefore, case objects should not be deleted until it is known for certain that no other processes are referencing the object, and that the case history is no longer needed.

If you delete a case object using a single case reference from within a service task in a business process, built-in checking is provided for other processes that are referencing the case object. (You can catch the `UnsafeToDeleteCaseError` error code.)

It is also possible to use the following methods to delete case objects, but these methods do not provide any error checking for other processes that are referencing the case object(s) that will be deleted. Using any of these methods could result in processes being halted because they are referencing a case object that no longer exists.



- Using the "deleteCase" operations in the BusinessDataServices API.
- Using a service task in a business process to delete:
 - multiple case objects using an array of case references.
 - a single case object using a case identifier.
 - a single case object using a composite case identifier.
- Using a service task in any type of process other than a business process - for example, a pageflow process or service process - to delete case objects (by any method).

Deletion of case objects is controlled by a system action (Delete Global Data) that defaults to deny, therefore you must be explicitly granted the permission to delete case objects.

Deleting Case Objects by Case Reference

Within a process, you can delete one or more case objects for which you have the identifying case reference(s).

Prerequisites

The process must already contain a data field containing either:

- a single case reference to the case object that you want to delete.
- an array of case references to a set of case objects that you want to delete.

If a case object that you want to delete has any [association links](#) to other case objects, you must [delete those association links](#) before attempting to delete that case object. If any case object has any active links to other case objects, the entire delete operation will fail - no case objects will be deleted.



Best practice is to avoid deleting case objects at all, but if you must do so, to delete only a single case object using a single case reference. For more information, see [Reasons to Avoid Deleting Case Objects](#).

Procedure

1. Add a service task at an appropriate point in the process.
2. On the **General** tab of the **Properties** view, set **Service Type** to **Global Data Operations**.
3. Select **Change or Delete Case Object(s) Using Case Reference Field**.
4. In the **Case Reference** field, select the data field that contains the case reference(s) that you want to delete.
5. In the **Operation** field, select **Delete Case Object(s)**.

What to do next

1. (Optional) Add appropriate error handling to the service task boundary to deal with the following specific errors that may be returned by the **Delete Case Object(s)** operation.

Error (Error Code)	Description	Possible solutions
CaseOutOfSyncError (CaseOutOfSyncError)	<p>A case object referenced from the Case Reference Field has been modified in some way since the field was last populated. That is, the case reference is stale.</p> <p>This could be either a change to the data held in the case object, and/or a change in the object's internal version number.</p>	<ol style="list-style-type: none"> 1. Re-read the case object. 2. If necessary, examine the case data to check that no significant changes have occurred. 3. Retry the Delete Case Object(s) operation.
UserApplicationError (CaseNotFoundError)	A case object referenced from the Case Reference Field does not exist.	
UserApplicationError (DeleteWhileLinkedError)	A case object referenced from the Case Reference Field has one or more association links to other case objects.	<ol style="list-style-type: none"> 1. Find and delete any association links. 2. Retry the Delete Case Object(s) operation.
UnsafeToDeleteCaseError (UnsafeToDeleteCaseError)	<p>The case object is referenced by processes other than the one that is attempting to delete the case object, therefore it is unsafe to delete the object.</p> <p>(This error is catchable only if you have specified a single case reference, not an array of case references.)</p>	Do not delete the case object.

2. (Optional) Modify any process that may need to do something if this case object is deleted:
 - a. Add a [case data signal event](#) to the process, so that the process will be notified if the case object is deleted.
 - b. Add suitable business logic to perform any actions that the process needs to take as a result of the case object being deleted.

Deleting Case Objects by Case Identifier

Within a process, you can delete one or more case objects whose case identifiers match the values that you supply.

Prerequisites

The process must already contain a data field containing either:

- the case identifier value of the case object that you want to delete.
- an array of case identifier values for a set of case objects that you want to delete.

The data field must be of the same type as the case identifier attribute in the case class.

Each case identifier value(s) must be a complete and exact match for the case object that is to be deleted. You cannot use partial matches or wildcard characters - these will result in a `CaseNotFoundError` runtime error.

If a case object that you want to delete has any [association links](#) to other case objects, you must [delete those association links](#) before attempting to delete that case object. If any case object has any active links to other case objects, the entire delete operation will fail - no case objects will be deleted.



The best practice is to *not* delete case objects as part of a normal operation. If you *do* delete case objects, the best practice is to only delete using a single case reference from within a service task in a process. For more information, see [Reasons to Avoid Deleting Case Objects](#).

Procedure

1. Add a service task at an appropriate point in the process.
2. On the **General** tab of the **Properties** view, set **Service Type** to **Global Data Operations**.
3. Select **Create Case Object(s) or Delete Case Object(s) by Case Identifier**.
4. In the **Case Class** field, select the appropriate case class for the case object(s) that you want to delete.
5. In the **Operation** field, select **Delete Case Object(s) by Case Identifier**.
6. In the **Where case identifier** field, enter the name of the case identifier that you want to use to identify the case object(s).
7. In the **Equals value of** field, enter the name of the local data field that contains the case identifier value(s) of the case object(s) to be deleted.

What to do next

1. (Optional) Add appropriate error handling to the service task boundary to deal with the following specific errors that may be returned by the **Delete Case Object(s) by Case Identifier** operation.

Error (Error Code)	Description	Possible solutions
UserApplicationError (CaseNotFoundError)	A case object specified in the Equals value of does not exist.	
UserApplicationError (DeleteWhileLinkedError)	A case object specified in the Equals value of has one or more association links to other case objects.	<ol style="list-style-type: none"> 1. Delete the association links. 2. Retry the Delete Case Object(s) by Case Identifier operation.

2. (Optional) Modify any process that may need to do something if this case object is deleted:
 - a. Add a [case data signal event](#) to the process, so that the process will be notified if the case object is deleted.
 - b. Add suitable business logic to perform any actions that the process needs to take as a result of the case object being deleted.

Deleting a Case Object by Composite Case Identifier

Within a process, you can delete a single case object whose composite case identifier matches the values that you supply.

Prerequisites

The process must already contain data fields containing the composite case identifier values of the case object that you want to delete.

Each data field must be of the same type as the appropriate composite case identifier attribute in the case class.

Each case identifier value(s) must be a complete and exact match for the case object that is to be deleted. You cannot use partial matches or wildcard characters - these will result in a `CaseNotFoundError` runtime error.

If the case object that you want to delete has any [association links](#) to other case objects, you must [delete those association links](#) before attempting to delete that case object. If the case object has any active links to other case objects, the delete operation will fail.



The best practice is to *not* delete case objects as part of a normal operation. If you *do* delete case objects, the best practice is to only delete using a single case reference from within a service task in a process. For more information, see [Reasons to Avoid Deleting Case Objects](#).

Procedure

1. Add a service task at an appropriate point in the process.
2. On the **General** tab of the **Properties** view, set **Service Type** to **Global Data Operations**.
3. Select **Create Case Object(s) or Delete Case Object(s) by Case Identifier**.
4. In the **Case Class** field, select the appropriate case class for the case object that you want to delete.
5. In the **Operation** field, select **Delete Case Object by Composite Case Identifier**.
A **Where *CIDAttributeName* equals value of** field is displayed for each attribute of the case class that forms its composite case identifier.
6. In each **Where *CIDAttributeName* equals value of** field, enter the name of the local data field that contains the value of the case object to be deleted for this *CIDAttributeName*

What to do next

1. (Optional) Add appropriate error handling to the service task boundary to deal with the following specific errors that may be returned by the **Delete Case Object(s) by Case Identifier** operation.

Error (Error Code)	Description	Possible solutions
UserApplicationError (CaseNotFoundError)	The case object specified in the Where <i>CIDAttributeName</i> equals value of fields does not exist.	
UserApplicationError (DeleteWhileLinkedError)	The case object specified in the Where <i>CIDAttributeName</i> equals value of fields has one or more association links to other case objects.	<ol style="list-style-type: none"> 1. Delete the association links. 2. Retry the Delete Case Object by Composite Case Identifier operation.

2. (Optional) Modify any process that may need to do something if this case object is deleted:
 - a. Add a [case data signal event](#) to the process, so that the process will be notified if the case object is deleted.
 - b. Add suitable business logic to perform any actions that the process needs to take as a result of the case object being deleted.

Searching For and Reading Case Objects

Within a process, you can use BDS scripting capabilities to search for and read case objects.

See the *TIBCO ActiveMatrix BPM Business Data Services* guide for general information about BDS scripting.

Case Access Classes

Case access classes are dynamically created classes, available in scripts (only), that allow you to access and retrieve case objects from the case data store.

If a business process contains a data field of type **BOM Type** or **Case Class Reference** that references a case class in a global BOM, a case access class is available in any scripts for each case class that exists in that global BOM.



If the referenced Business Data project contains other global BOMs (.bom files), case access classes will not be available for case classes in those BOMs unless at least one of those case classes is used by the process in a data field.

Within a script, you can use content assistance to check which case access classes are available.

The format of a case access class name is:

```
cac_packageName_caseClassName
```

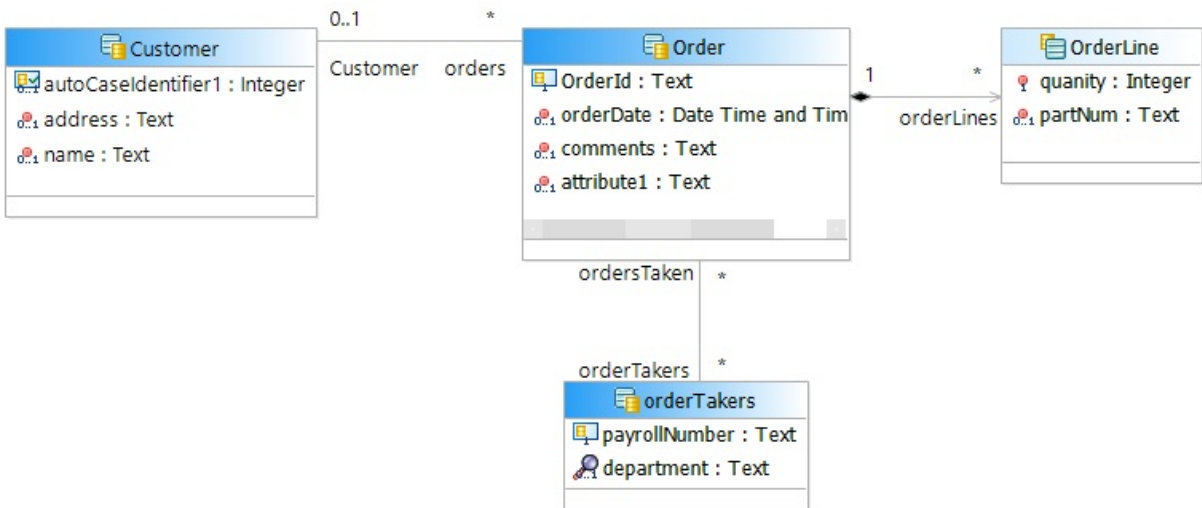
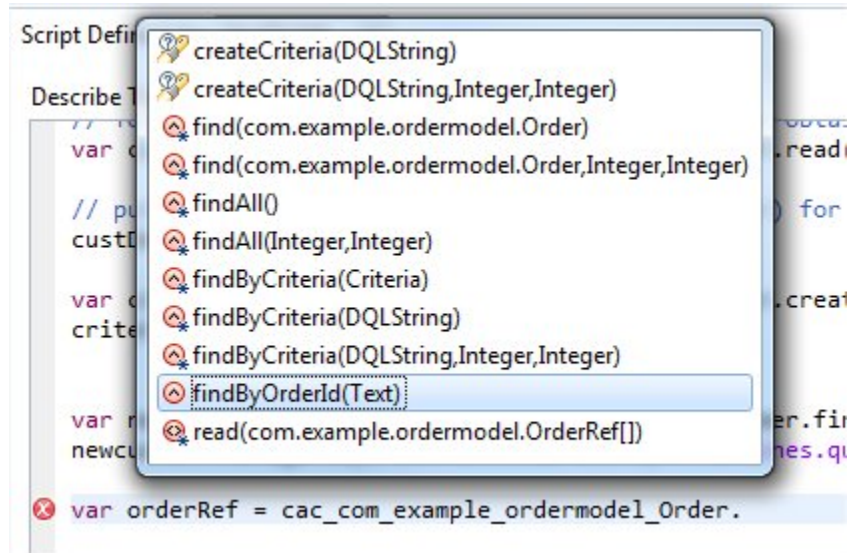
For example, if a `com.example.ordermodel` case data model contains a `Customer` class, the corresponding case access class for the `Customer` class is called:

```
cac_com_example_ordermodel_Customer
```

Each case access class exposes a set of methods that allow you to:

- search for case objects. You can search:
 - by [using a specific case identifier](#).
 - by [using an example set of searchable attributes](#).
 - for [all case objects of a particular case class](#).
 - by [using criteria](#) (defined as a DQL query string).
- [create criteria objects](#) that can be re-used in different searches.
- [read case objects](#).
- cast a case reference from a general class to a specialized class. This method only appears for a class that is a specialization of a general class. See [Handling Case Objects That Use Inheritance in Scripts](#).

For example, this is the content assistance that shows the methods and attributes available on the `cac_com_example_ordermodel_Order` case access class shown below.



Finding a Case Object by its Case Identifier

From within a script, use the `findByCaseIdentifier` method on the appropriate case access class to return the case reference to the case object that matches the specified case identifier value(s).

Method syntax	Description
<code>findByCaseIdentifierName(caseId)</code>	Returns the case reference to the case object that matches the specified <code>caseId</code> case identifier value. One <code>find</code> method is available for each <code>CaseIdentifierName</code> attribute of type Auto or Custom defined on the case class.

Method syntax	Description
<code>findByCompositeIdentifier(caseId,caseId,..)</code>	<p>Returns the case reference to the case object that matches the specified composite case identifier values.</p> <p>A single <code>findByCompositeIdentifier</code> method is available if the case class has a composite case identifier. The method contains one <i>caseId</i> parameter, of the appropriate type, for each attribute that is part of the composite case identifier.</p>

The method returns null if there is no case object that matches the specified *caseId* value(s).

Example

This example finds a customer's ID using the value in a `custID` field, which is assumed to have been populated earlier in the process (for example, by a customer service representative filling in a form when the customer telephones).

```
// Get the case reference to the Customer class that matches the identifier
// provided in the custID field.
customerReference =
cac_com_example_ordermodel_Customer.findByAutoCaseIdentifier1(custID);

// You can now read the reference to get the corresponding Customer case data
// object.
customer = customerReference.readCustomer();
```

Finding Case Objects by Example

From within a script, use the `find` method on the appropriate case access class to return a list of case references whose searchable attributes match the searchable attributes set in a local data object instance of the case class.

Method syntax	Description
<code>find(localObject)</code>	Returns a single list containing all matching case references.
<code>find(localObject,index,pageSize)</code>	Returns a paginated list of matching case references.

where:

- localObject* is a local data object instance of the case class, containing one or more searchable attributes set to the values to be used to find matching case objects. If multiple searchable attributes are specified, the case objects must match all specified values.



You must use only searchable attributes. If you use any non-searchable attributes they will be ignored. If you use only non-searchable attributes the script will fail at runtime, with an `BDSMissingCIDORSearchableAttributeException` exception in the BPM log file.

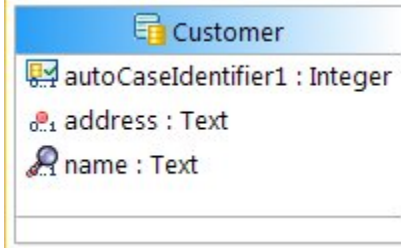
- index* is the (zero-based) number of the first record to return in a paginated list.
- pageSize* is the number of records to return in each page of a paginated list. A value of `-1` means return all records.



These methods are not sophisticated enough to find case objects based on searchable attributes specified in global objects contained in a case data object. To do this, you must use the `findByCriteria` method. See [Finding Case Objects by Criteria](#).

Example

This example uses the following case class:



The script finds all the customers with a name of "Harris", and returns a list of references to them:

```

// Create a local data object instance of the Customer case class.
var customer = com_example_ordermodel_Factory.createCustomer();

// Set the name field to the value you want to search on. 'name' must be a
searchable attribute
// on the Customer class. 'address' is not searchable and so cannot be used.
customer.name = 'Harris';

// Get a list of all Customer case references that contain the name 'Harris'.
var customerReferenceList = cac_com_example_ordermodel_Customer.find(customer);

// You can now read the list of case references to obtain a list of corresponding
Customer case data objects.
var customerList = cac_com_example_ordermodel_Customer.read(customerReferenceList);

// Then process matching customers.
for (var ix = 0; ix < customerList.size(); ix++)
{
    var customer = customerList.get(ix);

    // process customer objects...
}
  
```

Finding All Case Objects of a Particular Type

From within a script, use the `findAll` method on the appropriate case access class to return a list of all case references for that case class.

Method syntax	Description
<code>findAll()</code>	Returns a single list containing all case references for the case class.
<code>findAll(index, pageSize)</code>	Returns a paginated list of all case references for the case class.

where:

- *index* is the (zero-based) number of the first record to return in a paginated list.
- *pageSize* is the number of records to return in each page of a paginated list. A value of `-1` means return all records.

Examples

This example shows how to get a list of all Customer case references.

```
// Get a list of all Customer case references
customerReferenceList = cac_com_example_ordermodel_Customer.findAll();

// You can now read the list of case references to obtain a list of corresponding
Customer case data objects.
var customerList = cac_com_example_ordermodel_Customer.read(customerReferenceList);

// Then process those customers.
for (var ix = 0; ix < customerList.size(); ix++)
{
    var customer = customerList.get(ix);

    // process customer objects...
    // ...
}
```

This example shows how to use a paginated list to iterate through the returned list of customers and find the one with the largest number of orders.

```
var maxOrderCount = 0;
var name = "";
var index = 0;
var pageSize = 100;

while (true) {
    var custRefs = cac_com_example_ordermodel_Customer.findAll(index, pageSize);
    for (var ix = 0; ix < custRefs.size(); ix++) {
        var customerRef = custRefs.get(ix);
        var count = customerRef.getOrdersRefs().size();
        var customer = customerRef.readCustomer();
        if (count > maxOrderCount) {
            maxOrderCount = count;
            name = customer.name;
        }
    }
    if (custRefs.hasMoreResults) {
        index += pageSize;
    } else {
        break;
    }
}
```

Finding Case Objects by Criteria

From within a script, use the `findByCriteria` method on the appropriate case access class to return a list of case references that match the criteria defined in a query string.

The query must be expressed in Data Query Language (DQL), and can be supplied either as a string or in a previously created [criteria object](#).

Method syntax	Description
<code>findByCriteria(criteriaObject)</code>	Returns a paginated list of case references that match the criteria specified in the <i>criteriaObject</i> .
<code>findByCriteria(DQLString)</code>	Returns a list of case references that match the criteria specified in the <i>DQLString</i> .
<code>findByCriteria(DQLString,index,pageSize)</code>	Returns a paginated list of case references that match the criteria specified in the <i>DQLString</i> .

where:

- *DQLString* defines the query to be executed. See [Case Data Query Language \(DQL\)](#).
- *criteriaObject* is a previously created [criteria object](#).
- *index* is the (zero-based) number of the first record to return in a paginated list.
- *pageSize* is the number of records to return in each page of a paginated list. A value of -1 means return all records.

Examples

This example shows three different ways of using a query string.

```
// Find customers who have no orders (returning references to the first 100
results).
var custRefs = cac_com_example_ordermodel_Customer.findByCriteria("size(orders) =
0", 0, 100);

// Find orders from customers whose name begins with "EasyAs" (returning
references to the first 20).
var ordRefs =
cac_com_example_ordermodel_Order.findByCriteria("customer.name='EasyAs*'",0,20);

// Find customers who have orders that include Part Number PN:10001
// ordering the results by customer name (returning up to 100 customer references).
var custRefs =
cac_com_example_ordermodel_Customer.findByCriteria("orders.orderLines.orderItem.partNum
= 'PN:10001' order by name", 0, 100);
```

This example shows how to use a query object, and how to use parameters to re-initialize the query on different executions.

```
// Create a new Criteria object defining a query to run against the Order class.
var criteria = cac_com_example_ordermodel_Order.createCriteria("orderTakers.name
= :orderClerk ORDER BY orderTaker.name ASC", 0, 10);

// Set the initial value of the orderClerk parameter.
criteria.setQueryParameter("orderClerk", "Fred%");

// Execute the query.
var orderList1 = cac_com_example_ordermodel_Order.findByCriteria(criteria);
var ref1 = orderList1.get(0);
var order1= ref1.readOrder();

// Run the query again using a different orderClerk parameter value.
criteria.setQueryParameter("orderClerk", "Ginger%");
var orderList2 = cac_com_example_ordermodel_Order.findByCriteria(criteria);
var ref2 = orderList2.get(0);
var order2= ref2.readOrder();
```

This example shows the use of a parameter with multiple values to find customers whose name begins with Fred, Ginger or Eric.

```
// Create a list of user patterns that we want to find.
var userList = Datautil.createList();
userList.add("Fred%");
userList.add("Ginger%");
userList.add("Eric%");

// Create a new Criteria object to run against the Customer class.
var criteria=cac_com_example_ordermodel_Customer.createCriteria("name
IN :userList",0,100);
criteria.setQueryParameter("userList", userList);

// Execute the query.
var customerList = cac_com_example_ordermodel_Customer.findByCriteria(criteria);
var ref1 = customerList.get(0);
var customer = ref1.readCustomer();
```

The final example demonstrates two ways of finding customers whose orders were dispatched in August 2013:

- using a query string,

```
var customerList =
cac_com_example_ordermodel_Customer.findByCriteria("orders.dateOfDispatch between
2013-08-01 and 2013-08-31");
```

- or using a query object with parameters.

```
var startDate = DateTimeUtil.createDate("2013-08-01");
var endDate = DateTimeUtil.createDate("2013-08-31");

var dateRange = DataUtil.createList();
dateRange.add(startDate);
dateRange.add(endDate);

var criteria =
cac_com_example_ordermodel_Customer.createCriteria("orders.dispatchNote.dateOfDispatc
h between :dateRng");
criteria.setQueryParameter("dateRng", dateRange);

var customerList = cac_com_example_ordermodel_Customer.findByCriteria(criteria);
```

Case Data Query Language (DQL)

DQL is used to define query expressions that can be used as parameters in the `findByCriteria` methods to search for case references that match the criteria defined by the query expression.

DQL expressions use the syntax:

attributeName operator value

where:

- *attributeName* can be specified either as a simple name, or as an [attribute path](#), which uses dot notation to navigate around the data model
- *operator* is one of the supported [operators](#).
- *value* is the value to match against, which can use [wildcards, constants and parameters](#).

For example:

```
name = "Tony Pulis"
```

Compound Expressions

Simple expressions can be combined with AND or OR to make more complicated expressions. For example, to match records that contain a '?' in the address attribute where the name starts with 'Fred', or, the name starts with Bill:

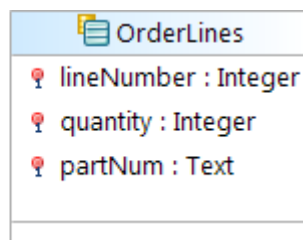
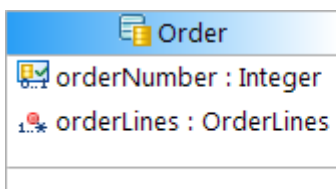
```
address = "London" AND name = "Fred" OR name = "Bill"
```

Normal operator precedence applies (AND is highest and OR is lowest) and can be overridden with the use of brackets. The example just given would be evaluated as:

```
((address = "London") AND (name = "Fred")) OR (name = "Bill")
```

AND and OR are case-insensitive.

When a compound expression refers to attributes that have multiplicity in their path, then the conditions are required to match the same object. For example, consider the following classes:



The following query requires a single orderline to match both conditions - it would not match if one orderline satisfies the first condition and another satisfies the second:

```
orderLines.quantity > 100 and orderLines.partNum = '1350CNW'
```

This query would not, therefore, match an order with the following orderLines:

quantity	partNum
150	ADR460
75	1350CNW

If you want to allow conditions to match different members, you can assign tags to the part of the attribute path that has multiplicity. Tags consist of a dollar sign followed by a letter, optionally followed by further numbers and letters. Assigning different tags to each condition removes the need for them to match the same object, so the following query would match the order described above.

```
orderLines[$a].quantity > 100 and orderLines[$b].partNum = '1350CNW'
```

Similarly, to find an order with one orderline for 150 x ADR460 and another for 75 x 1350CNW, the following query could be used:

```
orderLines[$x].quantity = 150 and orderLines[$x].partNum = 'ADR460' and
orderLines[$y].quantity = 75 and orderLines[$y].partNum = '1350CNW'
```

Sort Order

You can add an ORDER BY clause to the end of the DQL statement to specify what order the records should be returned in. The format of the ORDER BY clause is:

```
[ORDER BY column ASC|DESC[, column ASC|DESC][..]]
```

Each *column* specification is a multiplicity-single attribute path followed by one of the keywords ASC or DESC.

For example, to find all the customers with the name "Blogs" and sort by name then address:

```
name = "%Blogs" ORDER BY name ASC, address asc
```

Reserved Words

The following are reserved words in DQL:

- and
- asc
- between
- by
- desc
- lower
- not
- of
- or
- order
- size
- type
- upper

You can use a reserved word in a query expression if you wrap the reserved word in curly brackets.

For example, if you have an attribute called "type" and you want to find all objects whose "type" attribute has a value of 1, you must wrap the reserved word in curly brackets, as shown below.

```
{type} = 1
```

Similarly, if you have a parameter called "type", you must wrap the reserved word in curly brackets, as shown below.

```
myAttribute = :{type}
```

If an attribute name has a suffix such as [ALL] or [1], then the curly braces go around the name only. For example:

```
{type}[ALL] = 1
```

Attribute Paths

An *attributeName* in a DQL expression can be specified either as a simple name, or as an attribute path, which uses dot notation to navigate around the data model.

For example:

```
orders.orderLines.quantity = 1
```

An attribute path can be:

- the name of an attribute of the case object
- the name of a composition/association relationship of the case object
- the name of an attribute within its containment hierarchy, separated by dots

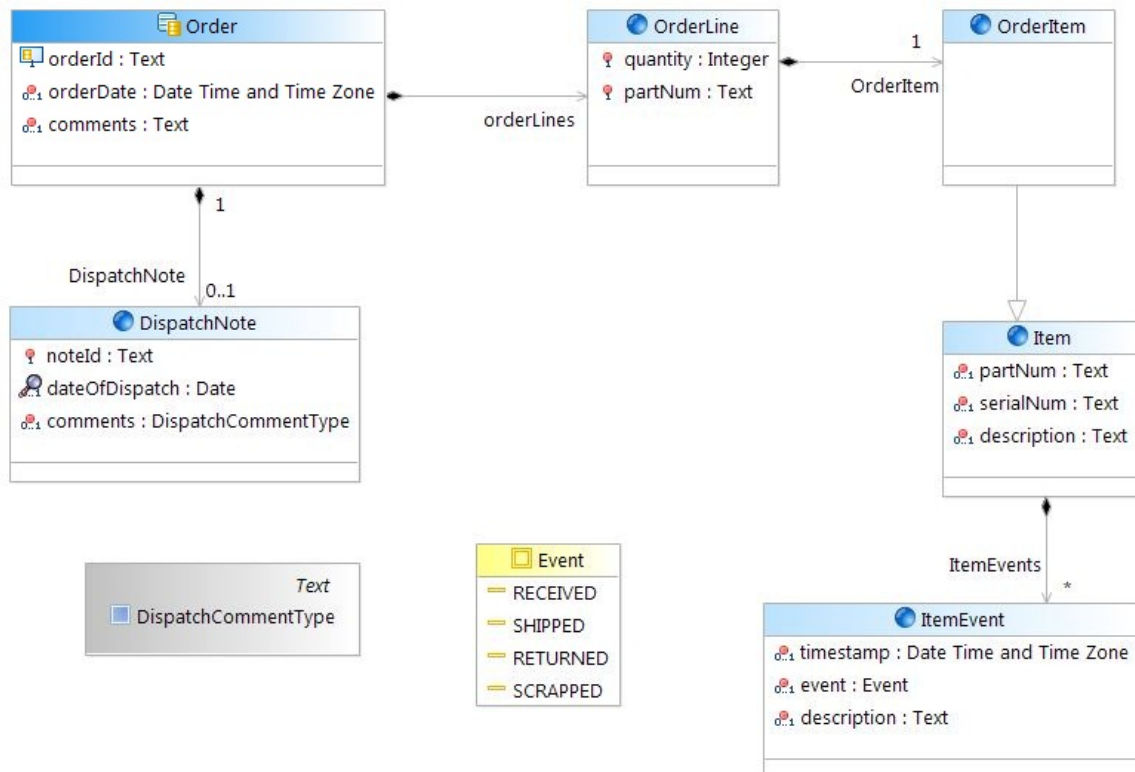


The path must contain the *name* of an attribute or relationship, *not* the label.

If an attribute is multiplicity-many, then it will, by default, match if any of its values satisfy the condition. Alternatively, '[ALL]' may be placed after the path fragment to indicate that all values must match.

A zero-based index number may be specified after the path fragment in the form '[i]' to require a given value within a list of multiple values to match.

For example, given the following model of an Order class:



The following table shows a number of example attribute paths and their meanings.

Attribute path	Meaning
orderId	The 'orderId' attribute.
comments	Any value of the multiplicity-many 'comments' attribute.
comments[ALL]	All values of the multiplicity-many 'comments' attribute.
comments[0]	The first (index zero) value of the multiplicity-many 'comments' attribute.
orderLines	The 'orderLines' composition relationship.
orderLines.quantity	The 'quantity' attribute of any of the OrderLine objects referred to by orderLines.
orderLines[ALL].quantity	The 'quantity' attribute of all of the OrderLine objects referred to by orderLines.
orderLines.orderItem.itemEvents[ALL].description	The 'description' attribute of all of the ItemEvent objects referred to by the OrderItem object of any OrderLine object referred to by orderLines. That is, there must be at least one OrderLine where all ItemEvents match.
orderLines[ALL].orderItem.itemEvents.description	The 'description' attribute of any of the ItemEvent objects referred to by the OrderItem object of all OrderLine objects referred to by orderLines. That is, every OrderLine must have at least one matching ItemEvent.
orderLines[ALL].orderItem.itemEvents[ALL].description	The 'description' attribute of all of the ItemEvent objects referred to by the OrderItem object of all OrderLine objects referred to by orderLines. That is, all ItemEvents must match.
dispatchNote	This is a reference to a global object, so only the =null and !=null operators are valid.

See Also

[Case Data Query Language \(DQL\)](#)

Operators

Operator	Description	Notes	Notes/Examples
=	Equal to	Can be used to match wildcard characters.	<code>name = 'John'</code> <code>age != 18</code>
!=	Not equal to	Can be used to test for null - for example, to test for attributes and associations that have not been set.	
◇	Not equal to	Can be used with the following BOM types: Text, ID, URI, Date, Time, Datetime, Datetimetz, Duration, Integer, Decimal, Boolean, Enumeration.	
>	Greater than	Can be used with the following BOM types: Date, Time, Datetime, Datetimetz, Duration, Integer, Decimal.	
>=	Greater than or equal to		
<	Less than		
<=	Less than or equal to		
BETWEEN	Test whether a field is between two values (inclusive).	The ',' character can be used instead of 'AND'. NOT BETWEEN can also be used. Can be used with the following BOM types: Date, Time, Datetime, Datetimetz, Duration, Integer, Decimal.	<code>quantity between 0 and 20</code> <code>quantity between 0,20</code> <code>quantity NOT between 0,20</code>
IN	Test if an attribute value matches one of a list of items.	If the attribute is a String type, then the values in the IN or NOT IN condition can contain wildcards. NOT IN can also be used. Can be used with the following BOM types: Text, ID, URI, Date, Time, Datetime, Datetimetz, Duration, Integer, Decimal, Boolean, Enumeration.	<code>name IN ("Tony", "Clint", "Eric")</code>

Operator	Description	Notes	Notes/Examples
TYPE OF	Can be used if processing a collection or association that contains a number of different types that come from a class hierarchy.	Can be used with the following BOM types: another Class.	<code>orderTakers type of 'com.example.ordermode1.Employee'</code>

See Also

[Case Data Query Language \(DQL\)](#)

Functions

Function	Description	Notes/Examples
SIZE	Find out how many objects there are in a containment.	Find orders that contain a single orderLine: <code>SIZE(orderLines) = 1</code>
UPPER	Convert Text, URI or ID attributes to upper case.	<code>UPPER(postcode) = 'SN%'</code>
LOWER	Convert Text, URI or ID attributes to lower case.	<code>LOWER(postcode) = 'sn%'</code>

See Also

[Case Data Query Language \(DQL\)](#)

Values

Values in DQL expressions can use wildcards, constants and parameters.

Wildcards

Use the '*' or '%' character to match any number of characters.

Use the '_' or '?' character to match any single character.

Use an '\' to escape these characters if you want to actually search for them.

For example:

```
name = "Fred*"
```

will match any records that have a name that starts with "Fred".

The string:

```
address = "*\?*"
```

will match any records that contain an address that contains a '?' character.

Constants

Type	Notes	Examples
Boolean	Must be written as TRUE or FALSE (case insensitive)	
String	<p>Must be either single-quoted or double-quoted, but not a mixture.</p> <p>Single quote characters in single-quoted strings, and double quote characters in double-quoted strings, must be escaped using a backslash character ('\').</p> <p>A backslash character in either type of string must be escaped using a second backslash character.</p> <p>Characters in a String constant value are case-sensitive. (if you require case-insensitive behavior, use the UPPER and LOWER functions).</p>	<pre>"fred" 'bill' "\fred\" 'fred' "fred\\bill" "\\'\\" '\\'\''</pre>
Numeric		<pre>0 -123 123.456</pre>
Date	<p>Must be specified as: YYYY-MM-DD</p> <p>A timezone can be specified by appending the date with either "Z", "+HH:MM" or "-HH:MM".</p> <p>When doing date range checking you can abbreviate dates, leaving off the the non-significant parts.</p>	<pre>2013-12-25 2013-12-25Z 2013-12+08:00 2013-05:00</pre> <p>The following expression searches for a date of 2013 or later.</p> <pre>dispatchNote.dateOfDispatch > 2012</pre>
Time	<p>Must be specified as: HH:MM:SS.SSS</p> <p>When doing time range checking you can abbreviate times, leaving off the the non-significant parts.</p>	<pre>14:23:56.123</pre> <p>The following expression searches for a time of 09:00:00 or later.</p> <pre>openingTime >= 09:00</pre>

Type	Notes	Examples
Datetime / Datetimez	<p>Must be specified as: YYYY-MM-DDTHH:MM:SS.SSS</p> <p>A timezone can (for a Datetime) or must (for a Datetimez) be specified by appending the date with either "Z", "+HH:MM" or "-HH:MM".</p> <p>When doing range checking you can leave off the the non-significant parts.</p>	<pre>2013-12-25T00:00:01Z 2013-12 2013-12-25T09+05:00 2013-12-25T00:00:01-08:00</pre>
Enum	Enum names must be quoted with either single or double quotes.	<pre>orderLines.orderItem.itemEvents.event = 'RECEIVED' orderLines.orderItem.itemEvents.event IN ("RECEIVED", "SHIPPED")</pre>

Parameters

Parameters can be used to supply a constant value at runtime, allowing the same query to be run with different values for the constants.

Specify a parameter name as an alphanumeric identifier, preceded by a colon character:

`:parameterID`

For example:

```
name = :customerName
```

At runtime, parameters can contain multiple values - for example:

```
name IN :customerList and orders.orderLines.quantity BETWEEN :quantityRange
```

Use the `setQueryParameter()` method to set parameter values in the Criteria object. (See [Finding Case Objects by Criteria](#).) The following table shows the different types of parameter value that can be assigned to different attribute types.

Attribute Type	Parameter Type
BigDec	BigDec, BigInt, Double, Int, String
BigInt	BigDec, BigInt, Double, Int, String
Double	BigDec, BigInt, Double, Int, String
Integer	BigDec, BigInt, Double, Int, String
Boolean	Bool, String
String	BigDec, BigInt, Double, Int, Bool, String, Date, Time, Datetime, Datetimez, Duration
Date	String, Date, Datetime, Datetimez
Time	String, Time

Attribute Type	Parameter Type
Datetime	String, Date, Datetime, Datetimez
Datetimez	String, Date, Datetime, Datetimez
Duration	String, Duration



If you use a String parameter for a non-String attribute the value of the String must be in the correct format - see [Constants](#).

Parameters can have lists of values assigned to them. (The IN and BETWEEN functions only have single parameters - when used with BETWEEN the parameter must have two values.)

Parameter values can be set to null to find attributes or associations that are not set.

See Also

[Case Data Query Language \(DQL\)](#)

Creating a Criteria Object

From within a script, use the createCriteria method on the appropriate case access class to define a reusable DQL query that can be used with findByCriteria or navigateByCriteria calls.

Method syntax	Description
<code>createCriteria(DQLString)</code>	Creates a criteria object that defines the specified <i>DQLString</i> as a query. When used in a subsequent <code>findByCriteria</code> or <code>navigateByCriteria</code> call, this object will return a list of case references that match the specified criteria.
<code>createCriteria(DQLString,index,pageSize)</code>	Creates a criteria object that defines the specified <i>DQLString</i> as a pre-paginated query. When used in a subsequent <code>findByCriteria</code> or <code>navigateByCriteria</code> call, this object will return a paginated list of case references that match the specified criteria.

where:

- *DQLString* is a [DQL string](#) that defines the query to be executed.
- *index* is the (zero-based) number of the first record to return in a paginated list.
- *pageSize* is the number of records to return in each page of a paginated list. A value of `-1` means return all records.



When creating a criteria object for use with a `navigateByCriteria` method, you must use the *target* case access class, not the source class. See [Navigating Association Links to find Related Case Objects](#).

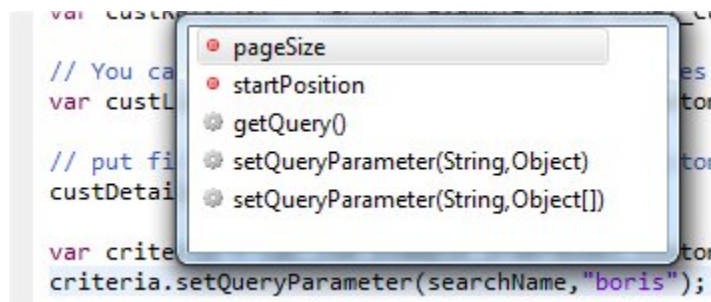
Criteria Object Methods and Attributes

A criteria object provides methods and attributes that allow you to return its DQL query string and its pagination details, and to set values for any parameters used in the query string.

Attribute	Description
startPosition	(Zero-based) number of the first record to return in a paginated list. This value defaults to 0 if the <i>index</i> value was not specified when creating the criteria object.
pageSize	Number of records to return in each page of a paginated list. A value of -1 means return all records. This is the default value if the <i>pageSize</i> value was not specified when creating the criteria object.

Method	Description
setQueryParameter(<i>paramName,value</i>)	Assigns a value to a named parameter that appears in the DQL query string. <i>paramName</i> is the name of a parameter defined in the DQL query string. <i>value</i> must be either a simple type or an enumeration.
setQueryParameter(<i>paramName,values</i>)	Assigns a list of values to a named parameter that appears in the DQL query string. <i>paramName</i> is the name of a parameter defined in the DQL query string. <i>values</i> must be a list of either a simple type or an enumeration.
getQuery()	Returns the DQL query string defined for this Criteria object.

For example, in the following screenshot the content assistance shows the methods and attributes available on a criteria object.



Example

This example shows how to use `setQueryParameter` to set a `dateRng` parameter on a criteria object. This object is then used in a query to find customers whose orders were dispatched in August 2013.

```
var startDate = DateTimeUtil.createDate("2013-08-01");
var endDate = DateTimeUtil.createDate("2013-08-31");

var dateRange = DataUtil.createList();
dateRange.add(startDate);
dateRange.add(endDate);

var criteria = cac_com_example_ordermodel_Customer.createCriteria(
    "orders.dispatchNote.dateOfDispatch between :dateRng");
criteria.setQueryParameter("dateRng", dateRange);

var customerList = cac_com_example_ordermodel_Customer.findByCriteria(
    criteria);
```

See Also

[Case Data Query Language \(DQL\)](#)

Case Reference Methods and Attributes

A case reference object provides attributes and methods that allow you to read the referenced case object, and navigate association relationships to linked case objects.

The **className** attribute provides the fully qualified class name of the referenced case object - for example, `com.example.ordermodel.Customer`.

The **simpleClassName** attribute provides the simple class name of the referenced case object - for example, `Customer`.

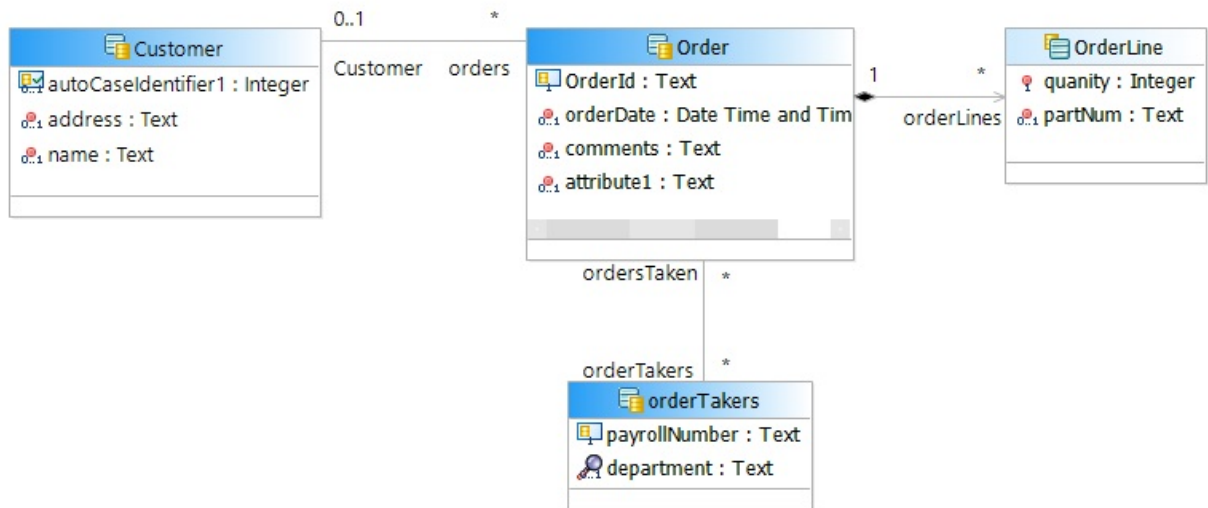
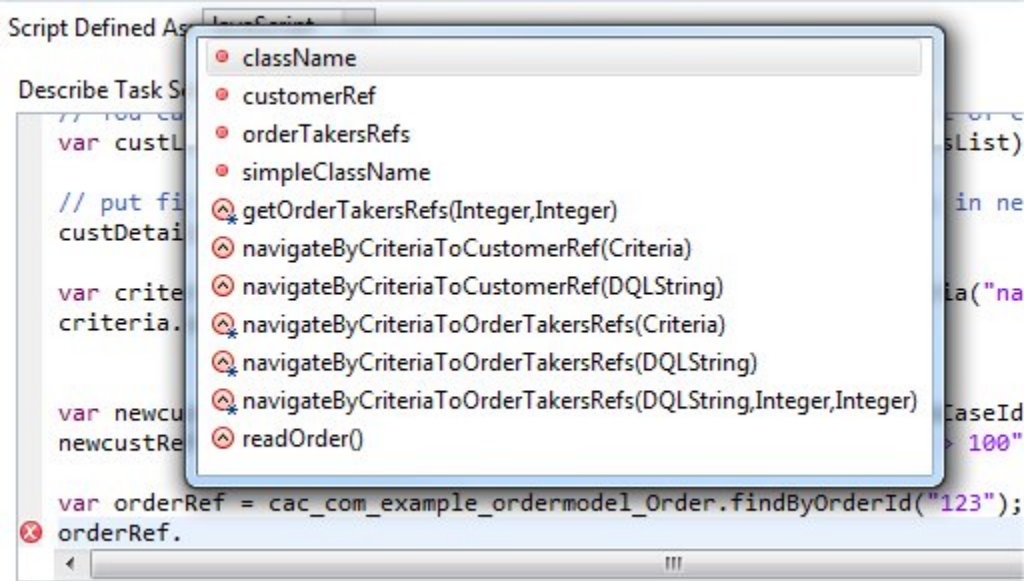


These attributes can be useful when processing lists of case references that contain a mixture of case class types. See [Handling Case Objects That Use Inheritance in Scripts](#).

Each case reference object also exposes a set of methods and attributes that allow you to:

- [navigate association links](#) from the referenced case object in order to find case objects in associated classes.
- [read the referenced case object](#).

For example, this is the content assistance that shows the methods and attributes available on the `orderRef` object shown below.



Navigating Association Links to find Related Case Objects

From within a script, each case reference provides a set of attributes and methods that you can use to navigate association links from the referenced case object to find related case objects.



You can only use these methods and attributes to navigate existing [association links](#). If you try to navigate using a link that either has not been created yet or has been deleted, execution of the script will fail.

The exact set of attributes and methods available depends upon the number of association links that exist from the referenced case class, and upon the multiplicity of those association links.

Zero-to-One Association Relationships

For each class that is related to this class by a 0-to-1 association relationship, the following attributes and methods are available.

Attribute	Description
<i>associationNameRef</i>	Returns the case reference object that is linked to this case reference object by the <i>associationName</i> relationship.

Method	Description
<i>navigateByCriteriaToassociationNameRef(criteriaObject)</i>	Returns the case reference object that is linked to this case reference object by the <i>associationName</i> relationship, and that matches the criteria specified in the <i>criteriaObject</i> .
<i>navigateByCriteriaToassociationNameRef(DQLString)</i>	Returns the case reference object that is linked to this case reference object by the <i>associationName</i> relationship, and that matches the criteria specified in the <i>DQLString</i> .

Zero-to-Many Association Relationships

For each class that is related to this class by a 0-to-many association relationship, the following attributes and methods are available.

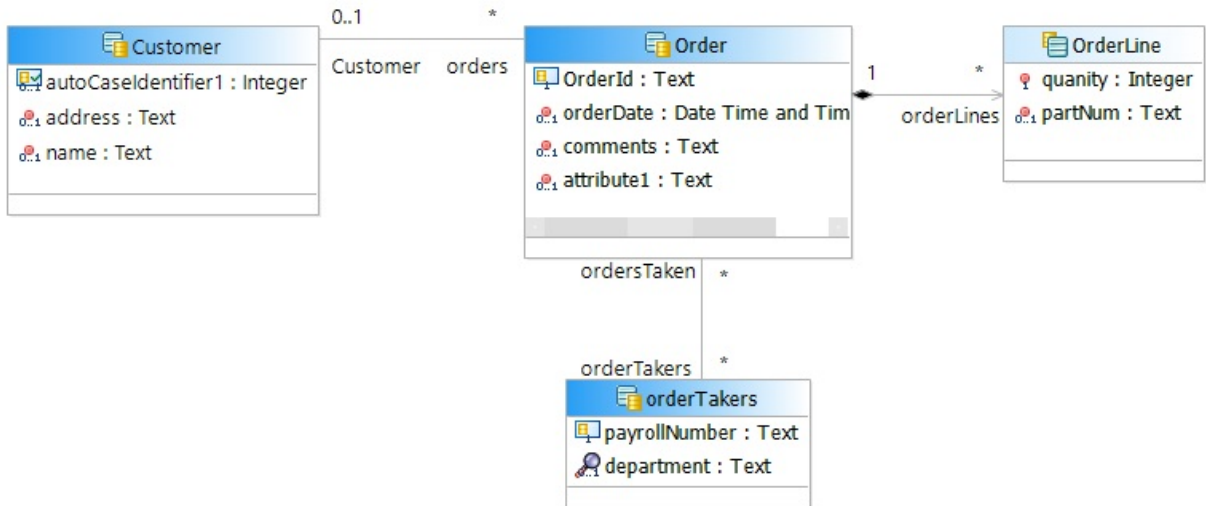
Attribute	Description
<i>associationNameRefs</i>	Returns a list of all case reference objects that are linked to this case reference object by the <i>associationName</i> relationship.

Method	Description
<i>getassociationNameRefs(index,pageSize)</i>	Returns a paginated list of case reference objects that are linked to this case reference object by the <i>associationName</i> relationship.
<i>navigateByCriteriaToassociationNameRefs(criteriaObject)</i>	Returns a paginated list of case reference objects that are linked to this case reference object by the <i>associationName</i> relationship, and that match the criteria specified in the <i>criteriaObject</i> .
<i>navigateByCriteriaToassociationNameRefs(DQLString)</i>	Returns a list of case reference objects that are linked to this case reference object by the <i>associationName</i> relationship, and that match the criteria specified in the <i>DQLString</i> .

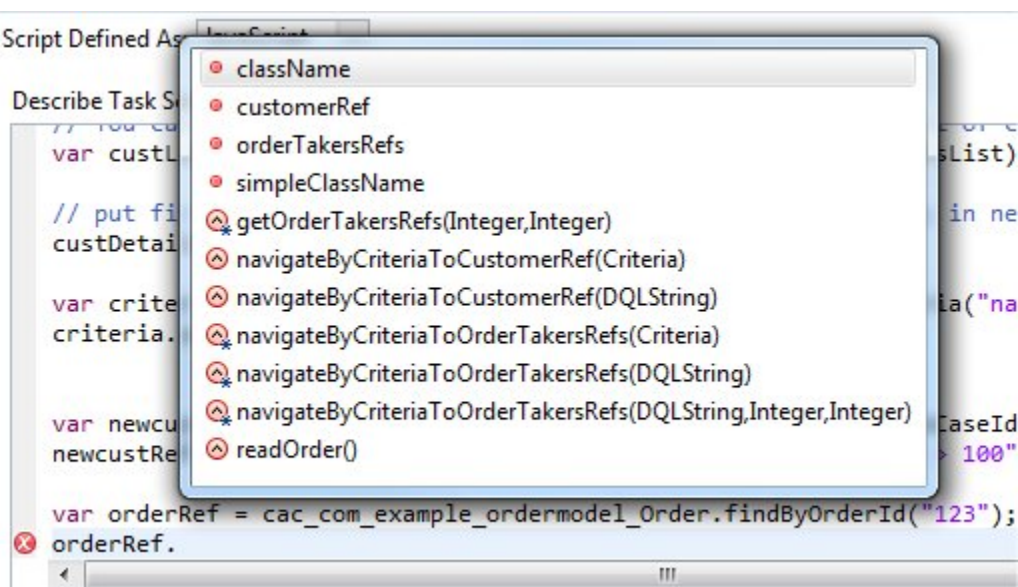
Method	Description
<code>navigateByCriteriaToassociationNameRefs(DQLString,index,pageSize)</code>	<p>Returns a paginated list of case reference objects that are linked to this case reference object by the <i>associationName</i> relationship, and that match the criteria specified in the <i>DQLString</i>.</p> <p><i>index</i> is the (zero-based) number of the first record to return.</p> <p><i>pageSize</i> is the number of records to return in each page of a paginated list. A value of -1 means return all records.</p>

Examples

The following examples are based on the following fragment of a case data model.



In the following screenshot the content assistance shows the methods and attributes available on an `orderRef` object.



This example demonstrates the use of the direct navigation methods and attributes. It shows how , given an order ID, to find the associated customer and the employee(s) who took the order.

```
// Find order number "123"
var orderRef = cac_com_example_ordermodel_Order.findByOrderId("123");

//Navigating a 0-to-1 relationship:
// Find the customer associated with this order
var customerRef = orderRef.customerRef;

//Navigating a 0-to-many relationship:
// Find the employees who took this order, either in a single list...
var employeeRefList = orderRef.orderTakersRefs;

// ...or, in a paginated list
var employeeRefList = orderRef.getOrderTakersRefs(0,5);
```

This example demonstrates the use of the navigateByCriteria methods. It shows how to navigate from a customer case reference object, returning orders with an orderLine quantity greater than 100.



The orderLines.quantity attribute path in the DQL criteria string is relative to the target class (Order) and not the source class (Customer).

```
orderRefList = customerRef.navigateByCriteriaToOrdersRefs("orderLines.quantity > 100");
```

This example demonstrates how to do the same thing using a criteria object. In this case, the createCriteria method for the target case access class - cac_com_example_ordermodel_Order - is used.

```
var criteria =
cac_com_example_ordermodel_Order.createCriteria("orderLines.quantity>100");
orderRefList = customerRef.navigateByCriteriaToOrdersRefs(criteria);
```

Association Links and Association Relationships

Association links allow you to navigate from one case object to another, using pre-defined association relationships between the corresponding case classes.

An association relationship exists between two case classes. You [create association relationships](#) when you create a case data model.

An association link exists between two case objects:

- An association link does not exist automatically because there is an association relationship between two case classes. You must [create an association link](#) from the source case object to the destination case object from within a business process using a **Global Data Operations** service task.
- An association link can only be created between case objects where there is an association relationship between the source and destination case classes.
- Once created, an association link exists until you [delete it](#), which you must do from within a business process using a **Global Data Operations** service task.



When using script methods and attributes to [navigate association links](#) from within a particular process, it is important to realize that association links can be created, navigated or deleted by any process that has access to the case reference for the source case object of the link. The process design may therefore need to account for the fact that a particular association link may or may not exist at runtime.

Creating an Association Relationship Between Two Case Classes

Association relationships define potential relationships between two case classes - for example, customers and orders. From within a business process, you can use these relationships to create links between case objects, and to navigate those links when searching for case objects.

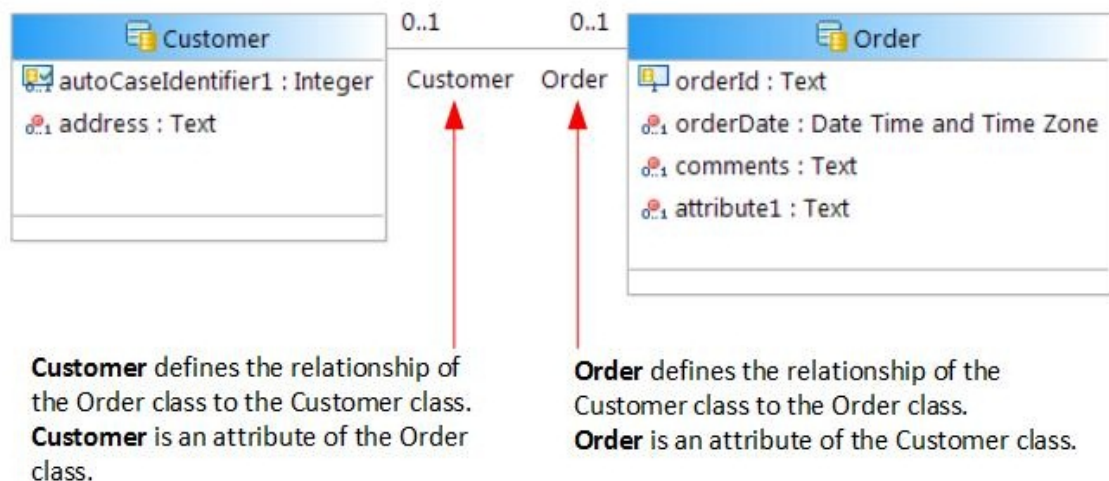
An association relationship must be bi-directional, and optional in each direction. (Neither end of the relationship can have a multiplicity value of "one" or "one or more".) You can define multiple associations between the same two classes, if appropriate.

Prerequisites

The case classes that you want to associate must already exist.

Procedure

1. In the BOM Editor, click **Association** in the Relationships section of the palette.
2. Click one case class and drag the relationship to the other case class.
This creates a bi-directional association relationship between the two case classes. By default, each end of the relationship has the name of the destination class, with a multiplicity of 0..1. For example:



3. For each end of the relationship, edit the **Multiplicity Value** and identifying **Label/Name** as needed to define the intended relationship. For example:

You can either select and edit the relationship's **Label** and/or **Multiplicity** directly on the canvas...

Label	Name	Type	Multipl...
autoCaseIdentifier1	autoCaseIdentifier1	Bom Primitive Types::Integer	0..1
address	address	Bom Primitive Types::Text	0..1
orders	orders	com.example.ordermodel::Order	*

... or edit them from the **Attributes** tab of the **Properties** view of the parent class.

In this example, the values for the Customer to Order relationship have been changed to show that a single Customer can have zero to many Orders.



It is good practise to keep a relationship's **Label** and **Name** the same. This makes it easier to avoid mistakes when constructing, for example, an attribute path in the DQL query on a navigateByCriteria method.

What to do next

From within a process that references this case data model, you can use association relationships to:

1. [create association links between case objects](#) of the related case classes. You can also [delete association links](#) when they are no longer required.
2. [navigate those links](#) when searching for case objects.

Using the relationship shown above, this would enable you to, for example, find all orders placed by a particular customer on a particular date.

Creating an Association Link

From within a process, use a **Global Data Operations** service task to link one case object to another. Association links allow you to navigate from one case object to another, from any process that has access to the case references of the source and destination case objects.

Prerequisites

An [association relationship](#) must exist between the source and destination case classes. Local data fields containing case references to the source and destination case objects must already exist.

Procedure

1. Add a service task at an appropriate point in the process.
2. On the **General** tab of the **Properties** view, set **Service Type** to **Global Data Operations**.
3. Select **Change or Delete Case Object(s) Using Case Reference Field**.
4. In the **Case Reference Field**, enter the name of the data field that references the source case object from which you want to add the link.
5. In the **Operation** field, select **Add Link(s) To Other Case Object(s)**.
6. In the **Case Class Association** field, enter the name of the association link that connects the source case class to the destination case class.
Content assist displays the association links that are available from the case class referenced in the **Case Reference Field**.
7. In the **Case Object Reference(s)** field, enter the name of the data field that references the destination case object for the specified **Case Class Association** field.

What to do next

1. (Optional) Add appropriate error handling to the service task boundary to deal with the following specific errors that may be returned by the **Add Link(s) To Other Case Object(s)** operation.

Error (Error Code)	Description	Possible solutions
CaseOutOfSyncError (CaseOutOfSyncError)	The source case object referenced from the Case Reference Field has been modified in some way since the field was last populated. That is, the case reference is stale. This could be either a change to the data held in the case object, and/or a change in the object's internal version number.	<ol style="list-style-type: none"> 1. Re-read the case object. 2. If necessary, examine the case data to check that no significant changes have occurred. 3. Retry the Add Link(s) To Other Case Object(s) operation.
UserApplicationError (CaseNotFoundError)	The source case object referenced from the Case Reference Field does not exist.	
UserApplicationError (AlreadyLinkedError)	The link to the destination case object cannot be created because it already exists.	
UserApplicationError (NoTargetsError)	The link cannot be created because the case reference specified in the Case Object Reference(s) field has a null value.	

2. (Optional) Modify any process that may need to do something if this association link is created:
 - a. Add a [case data signal event](#) to the process, so that the process will be notified when the association link is created.

- b. Add suitable business logic to perform any actions that the process needs to take as a result of the association link being created.

Deleting an Association Link

From within a process, use a **Global Data Operations** service task to delete an existing association link between two case objects. Association links allow you to navigate from one case object to another, from any process that has access to the case references of the source and destination case objects. You cannot delete a case object that has an association link to another case object.

Prerequisites

An [association link](#) must exist between the source and destination case objects. Local data fields containing case references to the source and destination case objects must already exist.

Procedure

1. Add a service task at an appropriate point in the process.
2. On the **General** tab of the **Properties** view, set **Service Type** to **Global Data Operations**.
3. Select **Change or Delete Case Object(s) Using Case Reference Field**.
4. In the **Case Reference Field**, enter the name of the data field that references the source case object from which you want to add the link.
5. In the **Operation** field, select **Remove Link(s) To Other Case Object(s)**.
6. In the **Case Class Association** field, enter the name of the association link that connects the source case class to the destination case class.
Content assist displays the association links that are available from the case class referenced in the **Case Reference Field**.
7. In the **Case Object Reference(s)** field, enter the name of the data field that references the destination case object for the specified **Case Class Association** field.

What to do next

1. (Optional) Add appropriate error handling to the service task boundary to deal with the following specific errors that may be returned by the **Remove Link(s) To Other Case Object(s)** operation.

Error (Error Code)	Description	Possible solutions
CaseOutOfSyncError (CaseOutOfSyncError)	The source case object referenced from the Case Reference Field has been modified in some way since the field was last populated. That is, the case reference is stale. This could be either a change to the data held in the case object, and/or a change in the object's internal version number.	<ol style="list-style-type: none"> 1. Re-read the case object. 2. If necessary, examine the case data to check that no significant changes have occurred. 3. Retry the Remove Link(s) To Other Case Object(s) operation.
UserApplicationError (CaseNotFoundError)	The source case object referenced from the Case Reference Field does not exist.	

Error (Error Code)	Description	Possible solutions
UserApplicationError (NotLinkedError)	The link to the destination case object cannot be deleted because it does not exist.	
UserApplicationError (NoTargetsError)	The link cannot be deleted because the case reference specified in the Case Object Reference(s) field has a null value.	

2. (Optional) Modify any process that may need to do something if this association link is deleted:
 - a. Add a [case data signal event](#) to the process, so that the process will be notified when the association link is deleted.
 - b. Add suitable business logic to perform any actions that the process needs to take as a result of the association link being deleted.

Reading Case Objects

From within a script, you can read case objects by using either the read method on the appropriate case access class, or the read*className* method on the appropriate case reference object.

Case Access Class Method	Description
read(List<classNameRef>)	Returns a list containing a local object of the <i>className</i> class for each case object identified by the specified <i>className</i> case references.

Case Reference Object Method	Description
read <i>className</i>	Returns a copy of the latest version of the referenced case object as a local object.


See Also

[Case Access Classes](#)

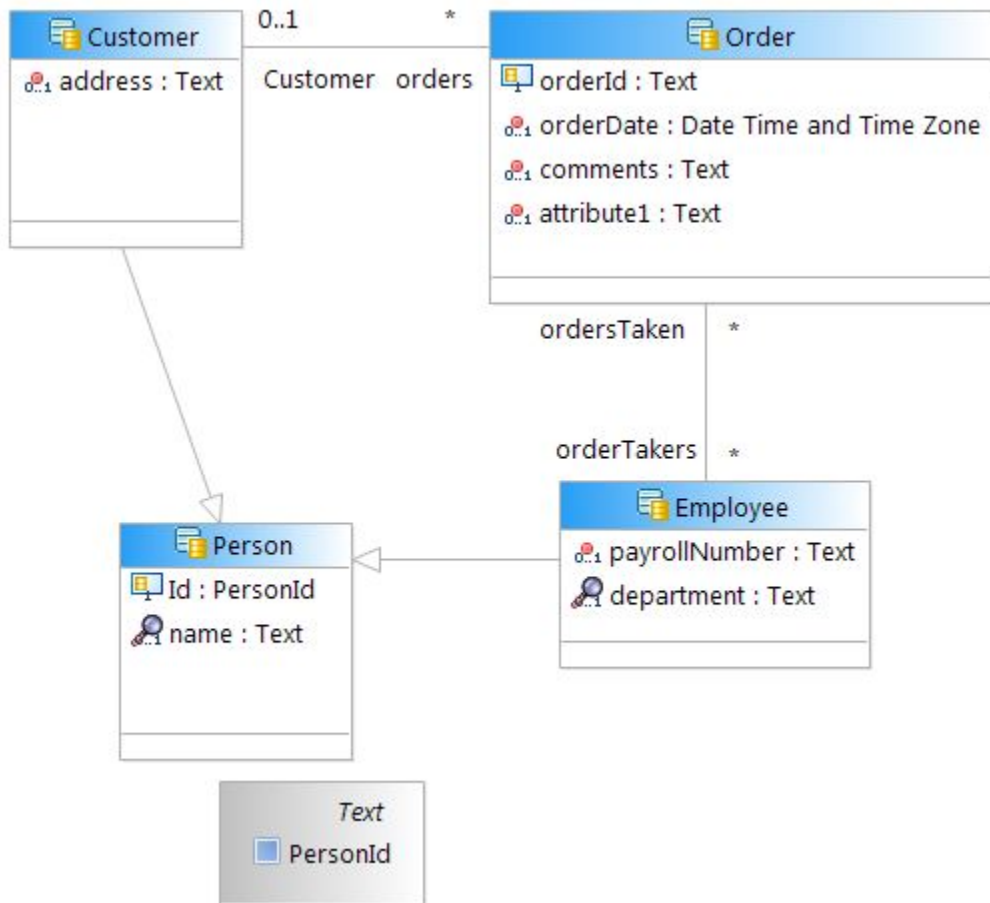
[Case Reference Methods and Attributes](#)

Handling Case Objects That Use Inheritance in Scripts

From within a script, use the *classNameRef* type-casting method for the appropriate case access class to return a case reference of the specialized class type from a case reference of the general class type.

Method	Description
<i>classNameRef</i> (<i>caseRef</i>)	Returns a case reference of the <i>className</i> type for the specified <i>caseRef</i> . <div style="display: flex; align-items: center; margin-top: 10px;">  <div style="border-left: 1px solid #ccc; padding-left: 10px;"> <p><i>caseRef</i> must belong to the parent (general) class type of which <i>className</i> is a sub-class. Invalid casts will result in exceptions at runtime.</p> </div> </div>

Case classes can use generalization relationships to provide specialization and inheritance. For example, in the following case data model the Employee and Customer classes are specializations of the (general) Person class. Note that they both inherit a searchable attribute (**name**) and their case identifier (**Id**) from the parent Person class.



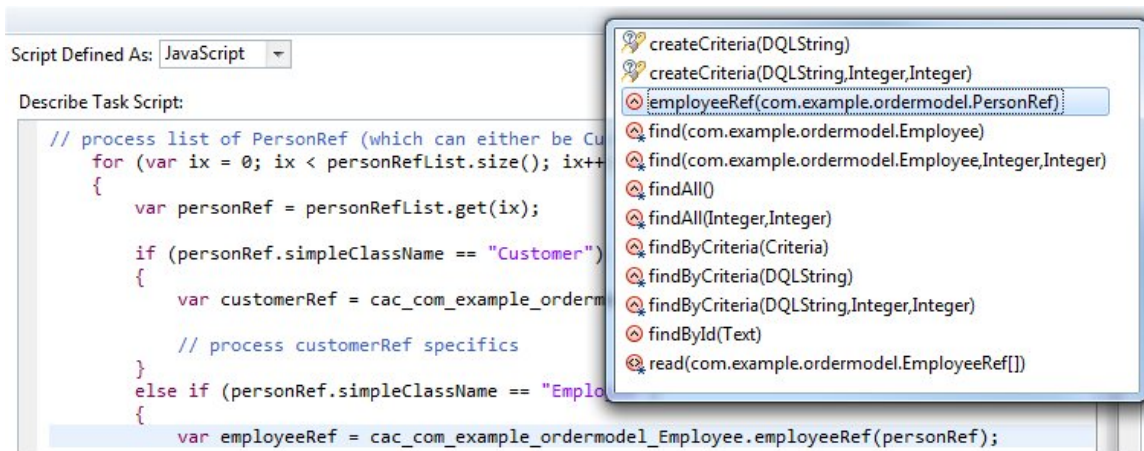
In a script, if you use the `findAll()` or `find()` methods to retrieve lists of case references for a general class type, each case reference in the list could be a reference to an object that belongs to either the general class itself, or to any specialized class derived from that class. In the example above, a list of Person case references could actually contain a mixture of Person, Customer and Employee references.

When processing the list, you may want to do different things depending on what type of class you are processing. To do this:

1. Use the `className` or `simpleClassName` attributes to determine which class type a particular case reference points to
2. Use the `classNameRef` type-casting method for the appropriate case access class to return a case reference of the specialized class type from a case reference of the general class type.
3. Use the type-specific methods provided by the appropriate case access class to perform the desired processing.



A `classNameRef` type-casting method is only provided on case access classes for specialized classes.



Example

This example shows how to take a list of PersonRef case references, determine whether each is an Employee or a Customer and provide appropriate processing for each.

```
var personRefList = cac_com_example_ordermodel_Person.findAll();

// Process the list of PersonRefs (each of which which can be either be a
CustomerRef or an EmployeeRef)
for (var ix = 0; ix < personRefList.size(); ix++)
{
    var personRef = personRefList.get(ix);

    // Use simpleClassName to determine whether this personRef is a CustomerRef or
EmployeeRef
    if (personRef.simpleClassName == "Customer")
    {
        var customerRef =
cac_com_example_ordermodel_Customer.customerRef(personRef);

        // Process customerRef specifics...
    }
    else if (personRef.simpleClassName == "Employee")
    {
        var employeeRef =
cac_com_example_ordermodel_Employee.employeeRef(personRef);

        // Process employeeRef specifics...
    }
}
}
```

Processing a Paginated List of Case References

When a paginated list of case references is returned by a method on a case access class or case reference, the list object has a `hasMoreResults` attribute that can be used to determine whether there are more pages of results that could be returned.

Paginated lists can be returned by the following case access class methods:

- [find\(\)](#)
- [findAll\(\)](#)
- [findByCriteria\(\)](#)

and by the following [case reference methods](#):

- [getassociationNameRefs\(\)](#)
- [navigateByCriteriaToassociationNameRefs\(\)](#)

Example

This example shows how to use a paginated list to iterate through a returned list of customers.

```
var maxOrderCount = 0;
var name = "";
var index = 0;
var pageSize = 100;

while (true) {
    var custRefs = cac_com_example_ordermodel_Customer.findAll(index, pageSize);
    for (var ix = 0; ix < custRefs.size(); ix++) {

        // Process the case reference...

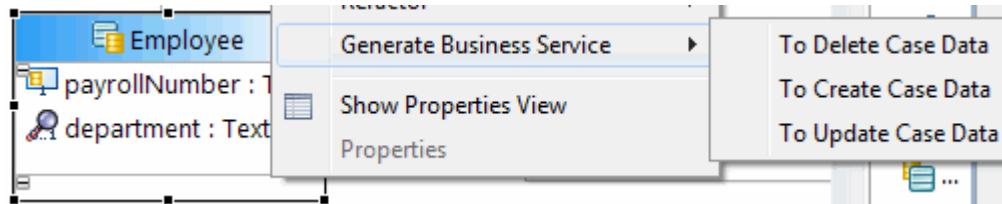
    }
}
if (custRefs.hasMoreResults) {
    index += pageSize;
} else {
    break;
}
}
```

Generating a Business Service to Create/Update/Delete Case Data

You can generate a business service from Case Class. The business service generated contains the relevant tasks to create, update or delete case data, depending on which option you selected when generating the business service.

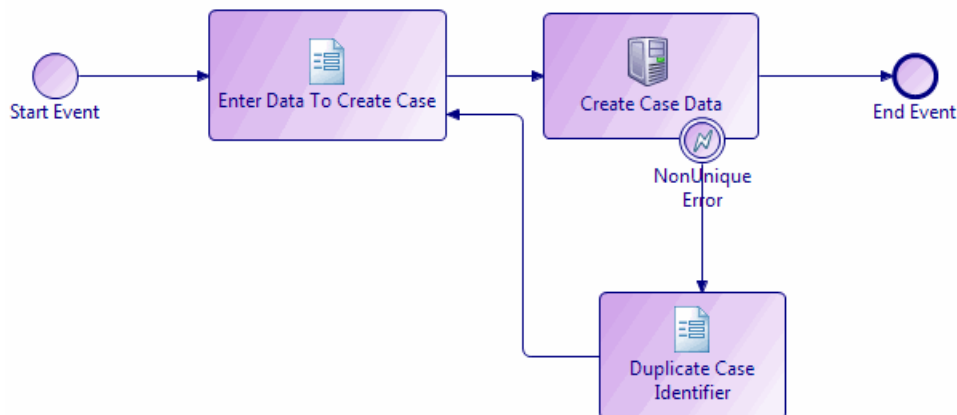
Procedure

1. Right-click on a Case Class in a BOM, and select **Generate Business Service**.

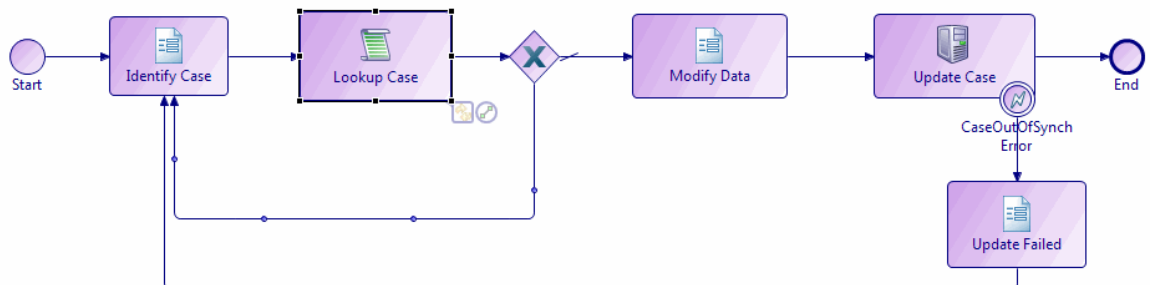


2. Select one of the following options:

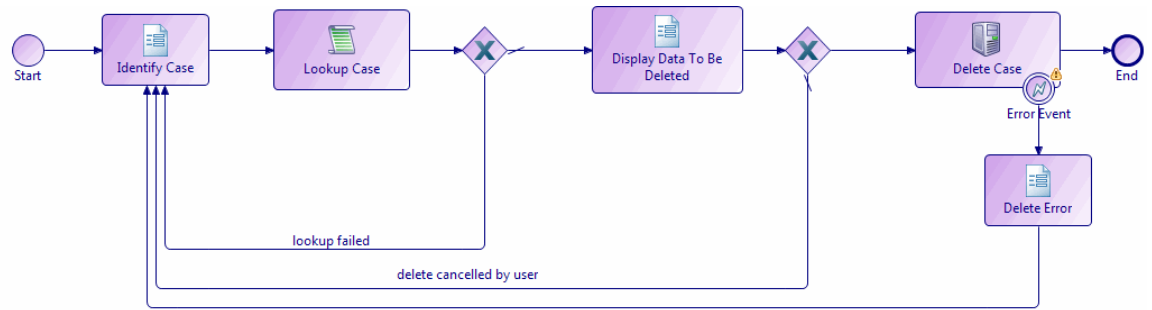
- **Create Case Data:** This creates a case data business service designed to allow you to create case data with appropriate runtime error handling, should a case object with the given case identification already exist.



- **Update Case Data:** This creates a case data business service designed to allow you to update case data with appropriate runtime error handling, should a case object with the given case identification not exist or have been updated since the lookup was performed.



- **Delete Case Data:** This creates a case data business service designed to allow you to delete case data with appropriate runtime error handling, should a case object with the given case identification not exist or have been updated since the lookup was performed.



- From the Generate Business Service dialog, open the project and Process Packages where you want to place the xpd. You can also select an existing xpd. Click **Finish**.

What to do next

When you have generated a case data business service, you can use it to edit case data. You are most likely to do this when you have an ad-hoc request not related to one of your normal business processes (for example, a customer has notified you of a change of address, and you want to make this change independent of any other changes to their order information. In this example you could use the Update Case Data business service to make the relevant changes to their data).

Using Case Data to Display Work Item Lists From Business Services Using TIBCO Openspace

In TIBCO ActiveMatrix BPM, you can specify case references in a business service and open a work item list associated with the specified case reference. This can be achieved using TIBCO Openspace.

TIBCO ActiveMatrix BPM allows you to use the case reference of the case data in a business service. You can then display a list of work items that are associated with the specified case reference. You can perform all the usual functions on the work items in the list in the normal way. For example, open, allocate, filter or sort work items. See *TIBCO Openspace User's Guide* for more information.



You must login in as a user that has the **ViewWorkList** system action assigned to it to view the work items in the list. This is because, when a business service is run and a work list displayed, it is not the work list of the currently logged in user. The work items in the work item list are associated with the specified case data. This means that some of the work items cannot be opened as only the work items that are offered or allocated to the currently logged in user can be opened.

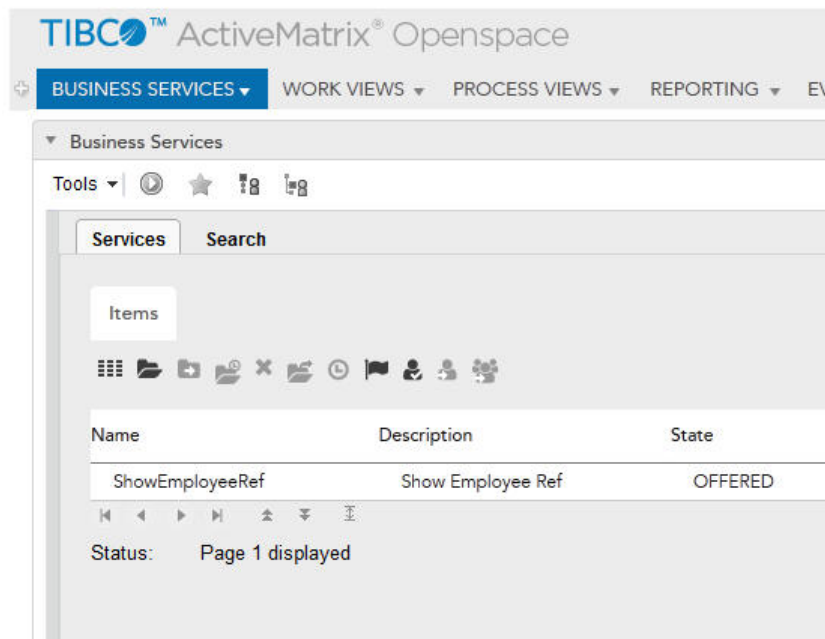
Once the work item list is closed, the current user task is completed and the business service progresses and completes, in the normal way.

This feature is provided as a component of the BPM Web Client API. See *TIBCO ActiveMatrix BPM Client Developer API Guide* for more information about the BPM Web Client API. TIBCO Business Studio uses Google Web Toolkit (GWT) that in turn uses the Web Client API to access the BPM runtime services.

Procedure

- In TIBCO Business Studio, create your BPM Developer Project in the normal way. See *TIBCO Business Studio Modeling Guide* for more information.
- Create a business service. Your business service can consist of as many activities as you require. See *TIBCO Business Studio Modeling Guide* for more information.
- Add a data field to your business service. The type of the data field must be TEXT and ensure its length is long enough for your case references. This data field is used to populate the business service with the case reference. The data field can be populated, either by the user or from another process. See *TIBCO BPM Implementation Guide* for more information.

4. Add the web component client template URL to the task that displays the work item list. This means that when this task is reached, a gadget is loaded and the work items associated with the specified case data display in the gadget. To do this:
 - a) Select the user task.
 - b) From the **General** tab, select **User-Defined Form**.
 - c) In the **Form Identifier** box, type the following URL: `bpm://worklist/getworkitemsforglobaldata/{datafield}` where *datafield* is the name of data field you defined in [Step 3](#)
5. Deploy the process as normal. See *TIBCO Business Studio BPM Implementation Guide* for more information.
6. From TIBCO Openspace, start the business service. Once you have entered the case reference, the work item list displays.



From this, you can perform all the functions that you can perform on a normal work item list. See *TIBCO Openspace User's Guide* for more information.

Using Case Data to Open Work Lists From Business Services Using Custom Clients

In TIBCO ActiveMatrix BPM, you can specify case data in a business service and open a work item list that belongs to the specified case data. If required, you can use this feature in your custom client. This feature is provided as a component of the BPM Web Client API.

TIBCO ActiveMatrix BPM allows you to use the case reference of the case data in a business service. You can then display a list of work items that are associated with the specified case reference. You can perform all the usual functions on the work items in the list in the normal way. For example, open, allocate, filter or sort work items. See *TIBCO ActiveMatrix BPM Client Developer API Guide* for more information.

When a business service is run and a work list displayed, it is not the work list of the currently logged in user. The work items in the work item list belong to the specified case data. This means that some of the work items cannot be opened as only the work items that are owned by the currently logged in user can be opened. You need to login as a user that has the **ViewWorkList** system action assigned to it to view the work-items.

This feature is provided as a component of the BPM Web Client API. See *TIBCO ActiveMatrix BPM Client Developer API Guide* for more information about the BPM Web Client API. Using TIBCO Business Studio,

you can develop web clients to be used with TIBCO ActiveMatrix BPM using the Web Client API. TIBCO Business Studio uses Google Web Toolkit (GWT) that in turn uses the BPM Web Client API to access the BPM runtime services.

The BPM Web Client API provides an API called `webCompRunner`. There are three methods that the `webCompRunner` API provides:

- The `initialize` method loads the client script and informs your custom client that the component is ready to be used.
- The `load` method loads the work item list associated with the case reference.
- The `unload` method provides any cleaning up and also allows you to load another work item list associated with a different case reference, if required.
- The `update_case_reference` method allows you to pass a new case reference.

Prerequisites

- To use this feature, you must have selected the **Openspace Gadget Development** installation profile when installing TIBCO Business Studio. The **Openspace Gadget Development** installation profile is available when you select **Customize Installation** during installation. See *TIBCO Business Studio Installation Guide*.
- The BPM Web Client API is provided as two jar files which are installed when you install the BPM runtime at `TIBCO_HOME\client-api\n.n\web-client-api` where *n.n* is the version of TIBCO ActiveMatrix BPM that you are using. For example, `C:\Program Files\tibco\amx-bpm\client-api\3.0\web-client-api`. The jar files are:
 - `bpm-web-client-api-1.4.jar`
 - `bpm-web-client-dep-1.4.jar`
 - Copy both the jar files to a folder which is on the machine hosting your development environment.
 - Edit the CLASSPATH to ensure that all the binary contents of the archive file are in the CLASSPATH. For example, add all the files available in `C:\temp\web-client-api` to the CLASSPATH.
- If you want to use TIBCO Forms in your in custom client, you must import the `formsclient.nocache.js` file into your web application project and include a script tag that points to this file. This provides the TIBCO Forms library. See *TIBCO Forms User's Guide* for more information about TIBCO Forms.
- On your hosting page, you need to have a reference to the `webcomponentclient.js` file.
- If you are developing a GWT app, you need to inherit the BPM web client API module.

Procedure

1. From TIBCO Business Studio, create your new BPM web application project. See *TIBCO ActiveMatrix BPM Client Developer API Guide* for more information.
2. Create and develop your gadget. See *TIBCO ActiveMatrix BPM Client Developer API Guide* for more information.
3. If you want your custom client to use TIBCO Forms, then you must import the `formsclient.nocache.js` into your web application project.
4. Once you have imported the `formsclient.nocache.js` file into your project, you must include a script tag in the `doctype.html` file in your custom client, as shown below:

```
<!doctype html>
<html lang="en">
<head>
```



```

<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>Openspace App Contribution</title>
<script type="text/javascript" language="javascript" src="bpmresources/formsclient/
formsclient.nocache.js"></script>
<link type="text/css" rel="stylesheet" href="gadget.css">
</head>
<body>
  <!-- OPTIONAL: include this if you want history support -->
  <iframe id="__gwt_historyFrame" tabindex="-1"
    style="position: absolute; width: 0; height: 0; border: 0"></iframe>
</body>
</html>

```

5. Use the `initialize` method to load the client script. The client script has to be loaded successfully for the `initialize` method to work. You need to use the `callBackHandler` provided by the BPM web client API to perform any further initialization. Any further initialization should be performed after the successful loading of the BPM Web Client API script.

For example:

```
WebCompRunner.setLoadReadyCallbackHandler (new LoadReadyCallbackHandler ())
```

6. The `Load` method loads the work item list associated with the specified case reference. The `Load` method has three arguments:
 - `WORK_LIST_CASE_REFERENCE`. The case reference of the case data.
 - The DOM Id. The location where the work item list should be loaded.
 - The BPM Web Client API class parameters. These are the parameters you can pass when loading the BPM Web Client API. The class parameters are:
 - `WebCompCallbackHandler`. The `WebCompCallbackHandler` notifies the BPM Web Client API that the BPM Web Client API class has loaded successfully or failed with an error.
 - `webCompCaseRef`. The `webCompCaseRef` identifies the case reference whose list of work items you want to view.

For example:

```
WebCompRunner.loadComponent(WebCompType.WORK_LIST_CASE_REF,
"WORK_LIST_CASE_REF.parent.element.id",
webCompLoadParams, new WebCompLoadCallbackHandler())
```

7. On the successful loading of the BPM Web Client API, you can call the `Unload` method. You can use this method to perform any required clean up. You can also use this method to call a new work item list for a new case reference. You can do this by passing a new case reference in the `webClientComponent UPDATE_CASE_REF` parameter.

For example:

```
webCompCaseRef.setUnloadCallbackHanlder (new UnloadCallbackHandler())
```


8. Deploy the process as normal. See *TIBCO Business Studio BPM Implementation Guide* for more information.
9. From your custom client, start the business service. Once you have entered the case reference, the work item list displays.


From this, you can perform all the functions that you can perform on a normal work item list. See *TIBCO ActiveMatrix BPM Client Developer API Guide* for more information.


Business Data Services Properties (bds.properties)

Configure where and how case data is created and managed.

Property	Default Value	Description
autogenerateDB	true	<p>Defines how the case data tables for a case data model are created, updated or deleted when that case data model is deployed or undeployed:</p> <ul style="list-style-type: none"> • <code>true</code>: ActiveMatrix BPM automatically creates, updates or deletes the required case data tables when a case data model is deployed or undeployed. • <code>false</code>: ActiveMatrix BPM generates SQL scripts to create, update or delete the required case data tables when a case data model is deployed or undeployed. A DBA must subsequently obtain and manually run these scripts.
caseDataStoreTableNameMaxLength	30	Defines the maximum number of characters in a case data table name. Changing this value may have an impact if you decide in future to migrate the case data store to a different database vendor (which may or may not support the maximum length that is specified here).
caseDataStoreColumnNameMaxLength	30	Defines the maximum number of characters in a case data column name. Changing this value may have an impact if you decide in future to migrate the case data store to a different database vendor (which may or may not support the maximum length that is specified here).
caseDataStoreIndexNameMaxLength	30	Defines the maximum number of characters in a case data index name. Changing this value may have an impact if you decide in future to migrate the case data store to a different database vendor (which may or may not support the maximum length that is specified here).
caseModelIdCacheSize	50	Defines the number of Ids (of each specific type) that will be returned in a single operation.
caseModelVersionIdCacheSize	50	
caseModelSpecIdCacheSize	50	
caseModelPackageSpecIdCacheSize	50	
caseModelTableMappingIdCacheSize	50	

Property	Default Value	Description
caseDataStoreIdCacheSize	50	
caseModelAuditIdCacheSize	50	
checkValidUpgradeChanges	true	<p>Defines whether ActiveMatrix BPM validates the changes made when an updated global data model is deployed:</p> <ul style="list-style-type: none"> • <code>true</code>. If you attempt to deploy an updated case data model that contains destructive changes as a new minor version, deployment will fail. • <code>false</code>. If you attempt to deploy an updated case data model that contains destructive changes as a new minor version, validation will not be performed. <p> TIBCO recommend that you do not use this option unless advised to do so by TIBCO Support.</p> <p>See "Upgrading a Case Data Model" in the <i>TIBCO ActiveMatrix BPM Case Data User Guide</i>.</p>
deleteFolderOnUndeploymentOrCaseObjectDeletion	true	<p>Defines how case folders are deleted:</p> <ul style="list-style-type: none"> • <code>true</code>: Folders are deleted implicitly by <code>deleteCase</code> or undeploying a model. • <code>false</code>: Folders are not deleted implicitly. A call to <code>deleteOrphanedFolders</code> is necessary to delete them. <p>This property is not added to the file on upgrade; it is added only on install. Therefore, if you want to change the default value of <code>true</code>, you must add the property to the file first.</p>
ensureReadConsistency	true	<p>Defines whether the operation that reads a case object from the database guarantees consistency. When the value is <code>true</code>, the operation detects if some other operation changes the case object while it is being read, thus causing the result of the read operation to be inconsistent. If the result is inconsistent, the read operation is retried until a consistent result is returned.</p> <p>Do not modify this value unless advised to do so by TIBCO Support.</p>

Property	Default Value	Description
onlyCheckNameForReferenceTypesOnUpgrade	false	<p>Disables checking that can, in particular circumstances, prevent upgrade of an already deployed Business Data project to a new minor version.</p> <p> You should only enable this property temporarily to deal with a specific upgrade problem - see "Upgrading a Business Data Project That References Another Business Data Project Fails" in <i>TIBCO ActiveMatrix BPM Troubleshooting</i> for more information about the problem and how to use this property to resolve it.</p>
pessimisticLockTimeoutMilliseconds	5000	<p>Defines the timeout period for a lock on a case object:</p> <ul style="list-style-type: none"> • 0. No timeout is set. An exception is thrown if BDS cannot get a lock on the requested case object. • >0. A timeout is set with the specified number of milliseconds. An exception is thrown if BDS cannot get a lock on the requested case object within this time period. <p>(This lock is used by User tasks.) Do not modify this value unless advised to do so by TIBCO Support.</p>
readConsistencyRetryLimit	1	<p>Defines the maximum number of times that the operation that reads a case object is retried until a consistent result is returned.</p> <p>Do not modify this value unless advised to do so by TIBCO Support.</p>

Property	Default Value	Description
supportTimeZoneInDateTime	true	<p>Controls time zone support for the "Date Time" data type, when that type is used in a case data model (and case objects created from that model).</p> <p>If supportTimeZoneInDateTime=true, or if supportTimeZoneInDateTime is not present in the <code>bds.properties</code> file (the default), time zone information is supported. In this case:</p> <ul style="list-style-type: none"> • A "Date Time" value that is written to a case object (for example, from a script or form) can optionally specify a time zone. If the value does not specify a time zone, it is assumed to be in UTC. In either case, the value written to the case object is stored internally (in the case data store) in UTC. • A "Date Time" value that is read from a case object is always qualified with the 'Z' suffix to indicate that it is in UTC. <p>If supportTimeZoneInDateTime=false, time zone support is disabled for the "Date Time" data type. In this case:</p> <ul style="list-style-type: none"> • A "Date Time" value that is written to a case object must not specify a time zone. If the value does specify a time zone, it will be rejected and the case object will not be created or updated. • A "Date Time" value that is read from a case object does not include a time zone. <p>To disable time zone support, add the following entry to the <code>bds.properties</code> file:</p> <pre>supportTimeZoneInDateTime=false</pre> <div style="border-left: 1px solid black; padding-left: 10px; margin-top: 10px;">  The supportTimeZoneInDateTime property does not affect the behavior of the "Date Time and Time Zone" data type. </div>
timerInterval	60	<p>Defines the period between internal process housekeeping checks.</p> <p>Do not modify this value unless advised to do so by TIBCO Support.</p>
useCMISRepository	false	<p>Defines whether BDS uses a CMS for the case folder feature.</p>

Configuring Whether Case Data Tables Are Managed Automatically or Manually

Whenever a case data model is deployed or undeployed, case data tables for that model must be created, updated or deleted (as appropriate). You can configure ActiveMatrix BPM to either do this automatically (the default option), or to generate SQL scripts that a DBA must subsequently run manually.

Automatic management is suited to a development environment, whereas manual management by DBA may be more suitable for or required in a production environment.



TIBCO recommend that you choose the appropriate setting before deploying the first case data model to the BPM system. Once any case data models have been deployed, you should only change this setting if advised to do so by TIBCO Support.

Procedure

1. Open the `CONFIG_HOME\bpm\bpmApplicationName\configuration\bds.properties` file.
2. Set the `autogenerateDB` property to one of the following values:
 - `true`: ActiveMatrix BPM automatically creates, updates or deletes the required case data tables when a case data model is deployed or undeployed.
 - `false`: ActiveMatrix BPM generates SQL scripts to create, update or delete the required case data tables when a case data model is deployed or undeployed. A DBA must subsequently obtain and manually run these scripts.



The case data model cannot be accessed by BPM process applications until this has been done.

3. Save the changes to the file.

Case Folders

Case folders are used to store, find, and retrieve content that is stored in a content management system (CMS). Each case folder, and its content, is associated with a "case," which is a case data object.

Your system can be configured so that whenever a case object is created, a case folder is automatically created. That case folder can then contain documents, or document references, relevant to the case.

The following describe various aspects of case folders:

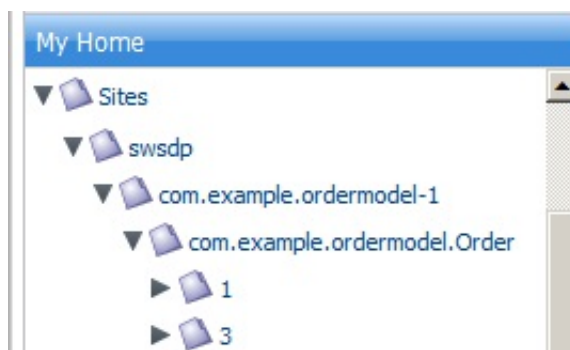
- Content Management System** - A content management system (CMS) must be available. This can be either the internal CMS provided with ActiveMatrix BPM (TIBCO DocumentStore), or an external CMS. For more information, see [Content Management System](#).
- Case Folder Configuration** - Before deploying projects that make use of case folders, you must configure your system, for example, to specify connection information to the CMS. For more information, see [Case Folder Configuration](#).
- Document Operations Service Type** - A service task of type Document Operations can be added to a business process or pageflow that can be used to perform case folder-related functions such as linking a document to a case, unlinking a document from a case, finding documents associated with a case, and so on. For more information, see [Document Operations Service Type](#).
- Using Case Folders** - A Case Management gadget is provided in TIBCO Openspace that can be used to perform case folder-related functions. Events related to case folder operation are also available through the Event Viewer in TIBCO Openspace. For more information, see [Viewing Case Folders in Openspace](#).
- Case Folder System Actions** - System actions are available that allow users who possess the appropriate actions to perform actions related to case folders. For more information, see [Case Folder System Actions](#).

Terminology for Case Folders

Specific terms are used with case folders.

Case Folder

A folder in the CMS that contains documents related to a case. Case folders have a specific hierarchy. The following is an example from the Alfresco CMS Explorer:



The folder hierarchy consists of the following parts:

- `Sites/swsdp` - This is the **repoRootFolderPath**, which can be specified using the TIBCO Configuration Tool (TCT) during installation of ActiveMatrix BPM, or in the `cm.properties` file -- see [Case Folder Configuration](#).
- `com.example.ordermodel-1` - This denotes the user application; it is created upon deployment.
- `com.example.ordermodel.Order` - This denotes the case class; there is one for each case class in a deployed BPM. This is created upon deployment. If your user application is later upgraded, case folders for new case classes introduced in the latest version are created upon upgrade.
- All of the subordinate folders are created when the first document is attached to a particular case. In this specific example, there are two orders (1 and 3) for which documents were attached.

Document Reference

A pointer to a document in the CMS.

Folder Reference

A pointer to a folder in the CMS.

Document Content

The actual binary content of a document.

Document Metadata

Information about a particular document, such as name, who created the document, date created, and so on.

Content Management System

A content management system (CMS) must be available. This can be either the internal CMS provided with ActiveMatrix BPM (TIBCO DocumentStore), or an external CMS.

The CMS must be installed and configured prior to deploying projects that use case folders. For information, see [Case Folder Configuration](#).

TIBCO DocumentStore

The TIBCO DocumentStore is an internal CMS, intended for light-weight usage, that is installed as part of ActiveMatrix BPM. It uses AtomPub binding and is CMIS 1.0 compliant.

All case folder functionality available in ActiveMatrix BPM can be performed using the DocumentStore, except for the following restrictions:

- The DocumentStore does not support versioning of documents.
- CMIS Query does not support full-text search.
- You cannot use the CONTAINS() function (used for full-text searches) when using the findDocuments operation to search for documents in a case folder.

External CMS

An external CMS must be compliant with Content Management Interoperability Services (CMIS), version 1.0. Case folder functionality has been verified using the following external content management systems:

- Alfresco 5 Community Edition
- Sharepoint 2013

Not all content management systems that claim to support CMIS support all features available through ActiveMatrix BPM. Check the vendor-specific documentation to verify which part of the CMIS specification their product supports.

Also, if ActiveMatrix BPM is configured to use Microsoft Sharepoint 2013 to provide its content management system (CMS), the following limitations apply to the use of the DocumentService in the ActiveMatrix BPM public API:

- The linkDocument and unlinkDocument operations cannot be used. This is because Sharepoint 2013 does not support multi-filing capability.
- You cannot use the LIKE operator when using the findDocument operation to search for documents in a case folder. This is because Sharepoint 2013 does not allow the use of the LIKE operator in the CMIS Query language.
- You cannot use the CONTAINS() function (used for full-text searches) when using the findDocuments operation to search for documents in a case folder. This is because Sharepoint 2013 mandates 'bothSeparate' query capabilities, but findDocuments performs a scoped search of documents (via the CMIS IN_TREE() clause) inside a case folder.

TIBCO DocumentStore

The TIBCO DocumentStore is available to test case folder functionality using the TIBCO ActiveMatrix BPM Development Server.

When an ActiveMatrix BPM Development Server is installed, the TIBCO DocumentStore is automatically installed on the server. The DocumentStore is an "internal CMS" that can be used to work with, and test, case folder functionality in a development / test environment. If case folders are to be used in the production system, an external CMS must be installed when moving to the production system. The DocumentStore cannot be used in a production environment.

The DocumentStore is automatically installed on the Development Server; no configuration is required.

The DocumentStore uses AtomPub binding and is CMIS 1.0 compliant

All case folder functionality available in ActiveMatrix BPM can be performed using the DocumentStore, except for the following restrictions:

- The DocumentStore does not support versioning of documents.
- CMIS Query does not support full-text search.

Case Folder Configuration

Case folders can be configured:

- when installing the ActiveMatrix BPM system, using the TIBCO Configuration Tool (TCT) **Create TIBCO ActiveMatrix BPM Server** wizard.
- after installation, by either:
 - using the TIBCO Configuration Tool (TCT) **Edit TIBCO ActiveMatrix BPM Instance** wizard, or
 - editing the `bds.properties` or `cm.properties` properties files.

The following table describes the configuration properties available for case folder functionality, as well as where each property is configurable.

Both the `bds.properties` and `cm.properties` files are located in the following directory on the machine in which ActiveMatrix BPM is installed:

`CONFIG_HOME/bpm/bpm_app_name/configuration`



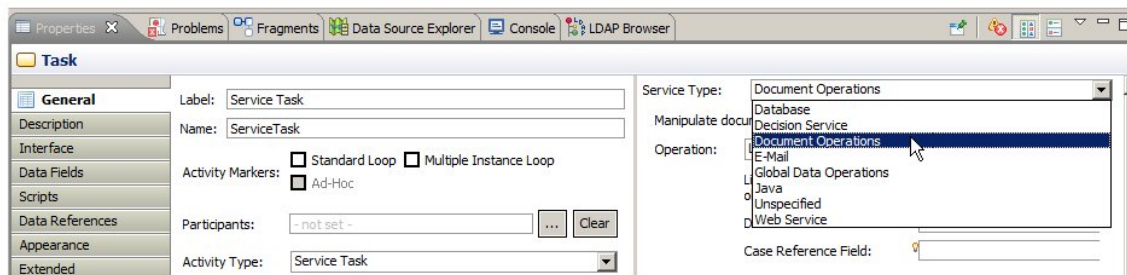
Property	Configured In	Description
<code>useCMISRepository</code>	TCT <code>bds.properties</code>	Enables the Document Service used for case folders. This property must be set to true to use any of the Document Service API calls. It also enables calling of the Document Service by core BDS (Business Data Services). If set to true, case folders are automatically created upon deployment. Default = false
<code>deleteFolderOnUndeployementOrCaseObjectDeletion</code>	<code>bds.properties</code>	If true, folders for a particular case are automatically deleted when the case is deleted, as well as when a model is undeployed. If this property is set to false, folders must be deleted using the <code>deleteOrphanedFolders</code> API call. Default = true
<code>repoConnectionURL</code>	TCT <code>cm.properties</code>	The URL used to connect to the CMS. This must be in the form: <code>http://CMSMachine:Port/cmsPath</code> where: <ul style="list-style-type: none"> • <i>CMSMachine</i> is the machine on which the CMS is running. • <i>Port</i> is the port using by the CMS. • <i>cmsPath</i> is the connection path used by your CMS. For example: <code>http://localhost:9080/alfresco/api/-default-/public/cmisis/versions/1.0/atom</code>
<code>repoBindingType</code>	TCT <code>cm.properties</code>	The binding type used by the CMS. Currently, this must be set to "atompub".
<code>repoID</code>	TCT <code>cm.properties</code>	The repository ID used by the CMS.
<code>repoRootFolderPath</code>	TCT <code>cm.properties</code>	The folder under which all sub-folders will be created in the CMS. If you are using an external CMS, you must ensure that this folder exists. Unlike the folders beneath it, this folder is not created on demand.
<code>repoConnectionTimeout</code>	<code>cm.properties</code>	The number of milliseconds after which a timeout will occur if a connection cannot be made with the CMS. This value can be increased if a "connection timed out" message appears in the <code>BPM.log</code> file. Default = 10000

Property	Configured In	Description
repoReadTimeout	cm.properties	The number of milliseconds after which a timeout will occur if a read operation cannot be completed. This value can be increased if a "read timed out" message appears in the BPM.log file. Default = 60000
trustStoreLocation	TCT cm.properties	The TrustStore file used when connecting to an HTTPS-enabled CMS. This is needed only if using HTTPS-based communications.
trustStoreType	TCT cm.properties	The TrustStore type. For example, JCEKS, JKS, etc. This is needed only if using HTTPS-based communications.
monitor.enable	cm.properties	Specifies whether or not the ActiveMatrix server will automatically detect and load changes made to this file at runtime. This should normally be set to true.
CMS Administrative username and password	TCT	These values are stored in the ECMCredentialsKeystore shared resource for security purposes. They can only be configured using TCT.

Document Operations Service Type

TIBCO Business Studio contains a Service Task of type "Document Operations" that provides functions such as finding, linking, and retrieving documents that are associated with a case data object.

The "Document Operations" Service Task is available in both business processes and pageflows.



The "Document Operations" Service Task can be configured to perform the following functions:

- [Delete Document](#) - Deletes the specified document from the CMS.
- [Find Documents Linked to Case Object](#) - Returns references to documents that are linked to a specified case.
- [Link Document to Case Object](#) - Links a document identified by document ID to a specified case.
- [Link System Document](#) - Links a document that is in the CMS to a specified case.
- [Move Document Between Case Objects](#) - Moves a document reference from one case to another case.
- [Unlink Document From Case Object](#) - Unlinks a document that is currently linked to a case.

When one of these document operations is chosen in TIBCO Business Studio, the input fields for that operation are displayed:

All input fields for all operations are required, and they all contain validations.

The input fields must be typed simple data fields / parameters. That is, they cannot be a complex data type.

Content assist is available on all of the input fields when defining the Document Operation. Place the cursor in the field, then press Ctrl+Space to display a list of the available input fields:

Delete Document

The Document Operation, Delete Document, deletes the specified document from the CMS. Deleting a "linked" document (that is, a document that is referenced from multiple folders) is deleted from *all* folders from which it linked.

Prerequisites

The following simple field/parameter (complex not allowed) must be defined before configuring this operation:

- **Document Reference Field** - Holds the document reference of the document to delete from the CMS.

Procedure

1. Add a Service Task to a process or pageflow.
2. In the **Properties** view, **General** tab, select "Document Operations" in the **Service Type** field.
3. In the **Operation** field, select "Delete Document".
The appropriate field for the selected operation is displayed under the **Operation** field.
4. In the **Document Reference Field** field, specify the field to hold the document reference.

Content assist is available; place the cursor in the field, then press Ctrl+Space to display a list of the available input fields.

Find Documents Linked to Case Object

The Document Operation, Find Documents Linked to Case Object, returns references to documents that are linked to a specified case.

Prerequisites

The following simple fields/parameters (complex not allowed) must be defined before configuring this operation:

- **Case Reference Field** - Holds the case reference of the case from which you are locating linked documents.
- **Return Document Reference Field** - When the operation is complete, this field will hold a document reference (or references) for the documents that are found.

Note that if the **Return Document Reference Field** is a single instance, and multiple documents are returned from the find operation, only the first document found is assigned to the **Return Document Reference Field**. Whereas, if the **Return Document Reference Field** is multiple instance, all returned documents (single or multiple) are assigned to the array field.

Procedure

1. Add a Service Task to a process or pageflow.
2. In the **Properties** view, **General** tab, select "Document Operations" in the **Service Type** field.
3. In the **Operation** field, select "Find Documents Linked To Case Object".
The appropriate fields for the selected operation are displayed under the **Operation** field.
4. Specify all of the input fields as described in the Prerequisites above.
Content assist is available on all of the input fields; place the cursor in the field, then press Ctrl+Space to display a list of the available input fields.
5. Specify how the user will be able to search for documents in the case using the following two options:
 - **Find By File Name (Field)** - Select this option if you want the user to be able to find documents based on a given document name.
 - **Find By CMISQL Query** - Select this option if you want to specify the specific CMIS properties on which the user can search for documents in the case. Use the table below this option to construct the query that the user can use in the application. For example, the following specification provides the user the ability to search based on the document name:

Find By File Name (Field):

Find By CMISQL Query:

Logical Gr...	(CMS Prop...	Operation	Value)
		cmis:name	LIKE	docName	

Note that the values specified are data fields / parameters that contain the value required by the query.

Not all content management systems support all of the query options detailed in the CMIS specification, therefore, check with your vendor's documentation.

Link Document to Case Object

The Document Operation, Link Document to Case Object, links a document identified by document ID to a specified case.

Prerequisites

The following simple fields/parameters (complex not allowed) must be defined before configuring this operation:

- **Document Reference Field** - Holds the document reference of the document to link to the specified case.
- **Case Reference Field** - Holds the case reference of the case to which the document will be linked.

Procedure

1. Add a Service Task to a process or pageflow.
2. In the **Properties** view, **General** tab, select "Document Operations" in the **Service Type** field.
3. In the **Operation** field, select "Link Document To Case Object".
The appropriate fields for the selected operation are displayed under the **Operation** field.
4. Specify all of the input fields as described in the Prerequisites above.
Content assist is available on all of the input fields; place the cursor in the field, then press Ctrl+Space to display a list of the available input fields.

Link System Document

The Document Operation, Link System Document, links a document that is in the CMS to a specified case.

Prerequisites

The following simple fields/parameters (complex not allowed) must be defined before configuring this operation:

- **Document Id Field** - Holds the document identifier of the document in the CMS.
- **Case Reference Field** - Holds the case reference of the case to which the document is to be linked.
- **Return Document Reference** - When the operation is complete, this field will hold the document reference of the document that was linked to the specified case.

Procedure

1. Add a Service Task to a process or pageflow.
2. In the **Properties** view, **General** tab, select "Document Operations" in the **Service Type** field.
3. In the **Operation** field, select "Link System Document".
The appropriate fields for the selected operation are displayed under the **Operation** field.
4. Specify all of the input fields as described in the Prerequisites above.
Content assist is available on all of the input fields; place the cursor in the field, then press Ctrl+Space to display a list of the available input fields.

Move Document Between Case Objects

The Document Operation, Move Document Between Case Objects, moves the specified document from one case folder to another case folder.

Prerequisites

The following simple fields/parameters (complex not allowed) must be defined before configuring this operation:

- **Document Reference Field** - Holds the document reference of the document to move.
- **From Case (Reference Field)** - Holds the case reference of the case from which the document is to be moved.
- **To Case (Reference Field)** - Holds the case reference of the case to which the document is to be moved.

Procedure

1. Add a Service Task to a process or pageflow.
2. In the **Properties** view, **General** tab, select "Document Operations" in the **Service Type** field.
3. In the **Operation** field, select "Move Document Between Case Objects".
The appropriate fields for the selected operation are displayed under the **Operation** field.
4. Specify all of the input fields as described in the Prerequisites above.
Content assist is available on all of the input fields; place the cursor in the field, then press Ctrl+Space to display a list of the available input fields.

Unlink Document From Case Object

The Document Operation, Unlink Document From Case Object, unlinks a document from a case, which removes the document from the case folder.

Note that this operation does *not* remove the document from the CMS; it only unlinks it from the case.

Prerequisites

The following simple fields/parameters (complex not allowed) must be defined before configuring this operation:

- **Document Reference Field** - Holds the document reference that is to be unlinked from the case.
- **Case Reference Field** - Holds the case reference of the case from which the document is to be unlinked.

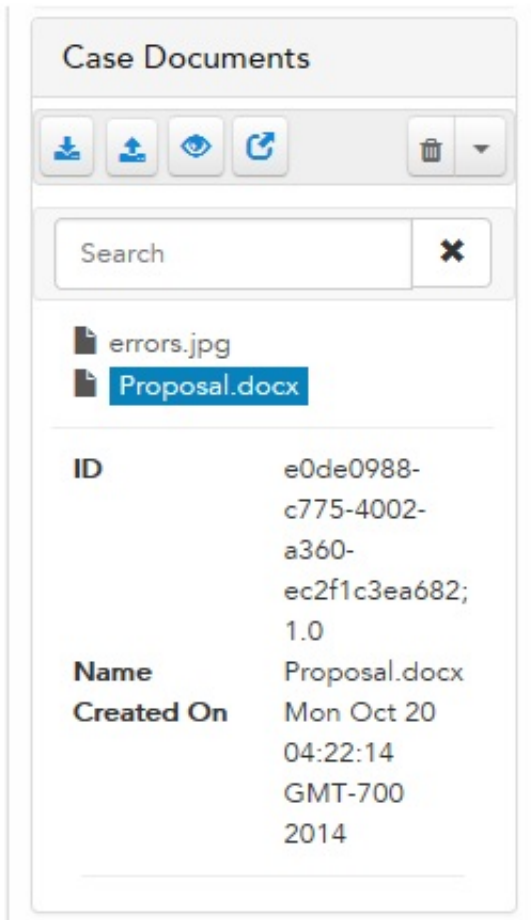
Procedure

1. Add a Service Task to a process or pageflow.
2. In the **Properties** view, **General** tab, select "Document Operations" in the **Service Type** field.
3. In the **Operation** field, select "Unlink Document From Case Object".
The appropriate fields for the selected operation are displayed under the **Operation** field.
4. Specify all of the input fields as described in the Prerequisites above.
Content assist is available on all of the input fields; place the cursor in the field, then press Ctrl+Space to display a list of the available input fields.

Viewing Case Folders in Openspace

Documents that are linked to cases can be viewed using TIBCO Openspace.

The Case Management gadget is used to view and work with case documents:



When one of the listed documents is selected, details about the document are displayed on the bottom of the **Case Documents** section. From this gadget, you can add or delete documents, view the contents of documents, and so on. For details, see the "Managing Case Documents" section in the *TIBCO Openspace User's Guide*.

Case Folder System Actions

There are two organization model-level system actions that control the rights to perform case folder-related operations.

These system actions, which are in the "Global Data" category, are:

- **Case Document Administration (cmisAdmin)** - This system action, which is "deny" by default, is required to:
 - Delete documents from the CMS
 - Delete orphaned case folders
 - Unlink a document from a case
- **Case Document User (cmisUser)** - This system action, which is "allow" by default, is needed for all other case folder-related operations (find, move, link, etc.) not controlled by the **cmisAdmin** system action.

TIBCO Documentation and Support Services

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [TIBCO Product Documentation](#) website, mainly in HTML and PDF formats.

The website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for TIBCO BPM Enterprise is available on the [TIBCO BPM Enterprise Product Documentation](#) page:

- TIBCO BPM Enterprise Release Notes
- TIBCO BPM Enterprise SOA Concepts
- TIBCO BPM Enterprise Concepts
- TIBCO BPM Enterprise Developer's Guide
- TIBCO BPM Enterprise Web Client Developer's Guide
- TIBCO BPM Enterprise Tutorials
- TIBCO BPM Enterprise Business Data Services Developer Guide
- TIBCO BPM Enterprise Case Data User Guide
- TIBCO BPM Enterprise Event Collector Schema Reference
- TIBCO BPM Enterprise - Integration with Content Management Systems
- TIBCO BPM Enterprise SOA Composite Development
- TIBCO BPM Enterprise Java Component Development
- TIBCO BPM Enterprise Mediation Component Development
- TIBCO BPM Enterprise Mediation API Reference
- TIBCO BPM Enterprise WebApp Component Development
- TIBCO BPM Enterprise Administration
- TIBCO BPM Enterprise Performance Tuning Guide
- TIBCO BPM Enterprise SOA Administration
- TIBCO BPM Enterprise SOA Administration Tutorials
- TIBCO BPM Enterprise SOA Development Tutorials
- TIBCO BPM Enterprise Client Application Management Guide
- TIBCO BPM Enterprise Client Application Developer's Guide
- TIBCO Openspace User's Guide
- TIBCO Openspace Customization Guide
- TIBCO BPM Enterprise Organization Browser User's Guide (Openspace)
- TIBCO BPM Enterprise Organization Browser User's Guide (Workspace)
- TIBCO BPM Enterprise Spotfire Visualizations
- TIBCO Workspace User's Guide

- TIBCO Workspace Configuration and Customization
- TIBCO Workspace Components Developer Guide
- TIBCO BPM Enterprise Troubleshooting Guide
- TIBCO BPM Enterprise Deployment
- TIBCO BPM Enterprise Hawk Plug-in User's Guide
- TIBCO BPM Enterprise Installation: Developer Server
- TIBCO BPM Enterprise Installation and Configuration
- TIBCO BPM Enterprise Log Viewer
- TIBCO BPM Enterprise Single Sign-On

How to Contact TIBCO Support

Get an overview of [TIBCO Support](#). You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the [TIBCO Support](#) website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to [TIBCO Support](#) website. If you do not have a user name, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

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

Legal and Third-Party Notices

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.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, TIBCO Administrator, Business Studio, Enterprise Message Service, Hawk, iProcess, JasperReports, and Spotfire are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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. 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.

Copyright © 2005-2022. TIBCO Software Inc. All Rights Reserved.