TIBCO BusinessEvents® Enterprise Edition

Data Modeling Developer Guide

Version 6.3.1 | September 2024

Contents

Database Concepts Overview

Database concepts are TIBCO BusinessEvents concepts that you create by mapping tables or views from a database to TIBCO BusinessEvents concepts.

One table or view maps to one TIBCO BusinessEvents database concept definition. A row in the table or view maps to one database concept instance. A column in a table or view gets mapped to a concept property.

Database concepts are created using the Database Import utility. This utility introspects the specified database schema and generates TIBCO BusinessEvents concepts. You can choose which tables or views to import.

When you import from a database, you can optionally generate relationships between database concepts based on database constraints. The utility imports any additional database entities that were not in the original selection but that must be imported because they have a relationship to the selected subset of entities.

A separate utility enables you to import domain models for database concepts from a source database column. You can then associate the domain model with the appropriate database concept.

Note: The database concepts are applicable only for JDBC stores.

Differences from Ordinary Concepts

The following are the differences between the database concepts and ordinary concepts:

- Assertion into Rete Network: Database concept instances returned by database query functions are not asserted into the Rete network automatically. You must assert them explicitly. To do so, use the RDBMS function `Database.assertDBInstance()`.
- History: Database concept properties do not support history tracking.

Minimum Permissions Required

Metadata access rights are required, because concepts and events are created using table metadata.

See Also

- *TIBCO BusinessEvents Architect Guide*, for understanding of database concepts and concept relationships.
- *TIBCO BusinessEvents Developer Guide*, for basic concept procedures and reference, and use of diagrams.
- *TIBCO BusinessEvents Configuration Guide*, for CDD configurations.
- *TIBCO BusinessEvents Administration* for deploy-time configuration.

Handling of Null Value Properties

To work with external databases, concept instances are serialized to XML. By default, when concept instance objects are serialized to XML, properties with null values are excluded. You can control handling of null properties in the XML representation of serialized concepts. Note that for numeric datatypes, some special handling may be required for interoperability. See the section **Handling Null Properties** in *TIBCO BusinessEvents Developer Guide* for details.

Table Constraints and Concept Relationships

When you import tables, optionally, you can import database constraints, that is, relationships between tables.

In the TIBCO BusinessEvents project the table relationships become relationships between concepts (see **Concept Relationships** in *TIBCO BusinessEvents Architect Guide*.) Database constraints are interpreted as contained or referenced concept relationships .

Note: The properties are always displayed as Concept Reference irrespective of containment or reference relationship. However, the actual relationship is stored in the `REL_TYPE` metadata property (which is a concept property level metadata property).

Tables Imported with Containment Relationships

After importing from tables, concept A contains concept B, if the following is true:

- Table A’s primary key is table B’s foreign key.
- And table A’s primary key is table B’s primary key in full or in part.

For example, cars contain tires:

- Table A is `Car`: Primary key is `carID`
- Table B is `Tire`: Primary key is (`carID, tireID`). Foreign key is `carID`.

After importing the `Car` and `Tire` tables to database concepts of the same names, the `Car` concept contains the `Tire` concept. `Car` has a `ContainedConcept` property called `Tire`, which points to `Tire`. (The `REL_TYPE` metadata property shows the type of relationship, as explained in the note.)

Tables Imported with Reference Relationships

After importing from tables, concept A references concept B if the following is true:

- Table A’s foreign key is Table B’s primary key.
- And table A’s primary key is not part of Table B’s primary key.

For example, orders reference sales representatives:

- Table A is `Order`: Foreign key is `repID`.
- Table B is `SalesRep`: Primary key is `repID`.

After importing the tables to database concepts of the same names, the `Order` concept references the `SalesRep` concept. `Order` has a `ConceptReference` property called `SalesRep` which points to `SalesRep`.

Note: The data types of the database source are imported to supported data types in TIBCO BusinessEvents. For example, an Oracle CLOB field is imported as a string property. You cannot use `queryUsingConceptProps` by specifying a value in the property that maps to a CLOB column.

You can also allow users to import tables selectively. For further information, see Importing Selected Database Tables.

Note: While importing from the IBM I/Series databases, you need to quote various strings internally. To do that, set the following property in the *BE_HOME*`/studio/eclipse/configuration/studio.tra` file:

```
be.dbconcepts.dbimport.use.quotes=true
```

Prerequisites for Importing Database Tables or Views

Setup the environment and resources before using the database import utility.

Configure the Environment

Copy the appropriate JDBC drivers file to *BE_HOME*`/lib/ext/tpcl`. You must restart TIBCO BusinessEvents Studio Explorer after copying the drivers file.

Note: The driver must be in the above location for the design-time **Test Connection** feature to work.

To use the debugger feature, you must also add build path information to pass to engines running inside TIBCO BusinessEvents Studio. See TIBCO BusinessEvents Developer Guide for details.

Create Destinations (as Needed)

If you use the option to create an event for each database concept, it’s a good idea to create a destination that you will specify as the default destination for the events. You can also create and configure the destinations later.

Set Up the JDBC Connection Resource

Add a JDBC Connection resource to the project and configure it for the database whose tables or views you want to import. To import concepts into the project from multiple data sources, set up a connection for each one. For details about adding a JDBC connection resource, see the TIBCO BusinessEvents Developer Guide.

You may also want to configure the connection using tuning settings in the Cluster Definition Descriptor (CDD). For more details, see Configuring the Database Connection.

Importing Database Tables or Views With the DB Import Utility

The DB Import utility allows you to connect to a database and import schemas.

You select the tables and views to import and the utility adds corresponding TIBCO BusinessEvents database concept definitions to the project. By default the utility names the database concepts using the table or view name. You can, however, provide different names as desired.

The utility provides an option to create one event for each table or view. The generated event's payload corresponds to the schema of the concept created for that table or view. These events can then be used when you perform database operations using provided functions, such as `queryUsingPrimaryKeys()`.

See Table Constraints and Concept Relationships to understand how table constraints affect concept relationships.

Note: Add the `be.dbconcepts.dburi.query.reuserefs` property, under the processing unit property group of the project CDD, to avoid duplicate Database concepts when two parents links to same row. When property is set to the value "true", and a query is made, then TIBCO BusinessEvents checks if the queried concept is already in cache. If the concept is already present in the cache, then that concept is used instead of creating a new concept.

Before you begin

Refer to Prerequisites for Importing Database Tables or Views for details configuration or setup required before importing the database tables or views.

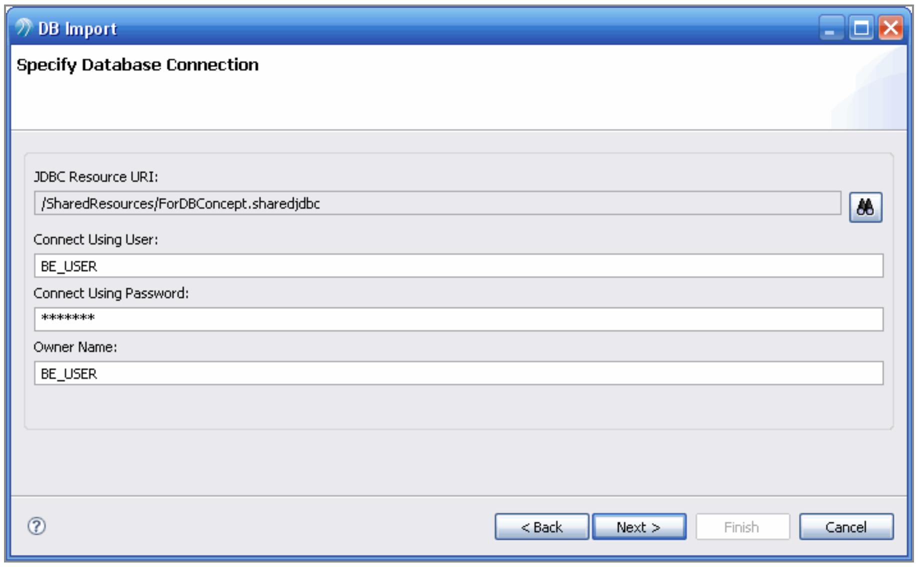
Procedure

1. Go to **File > Import > TIBCO BusinessEvents > Database Concepts.**

   The DB Import wizard displays, showing the Specify Database Connection dialog.

2. In the JDBC Resource URI field, click the **Browse** button and select the **JDBC Resource URI** for the database connection you added.

   The database connection details from the JDBC resource display.



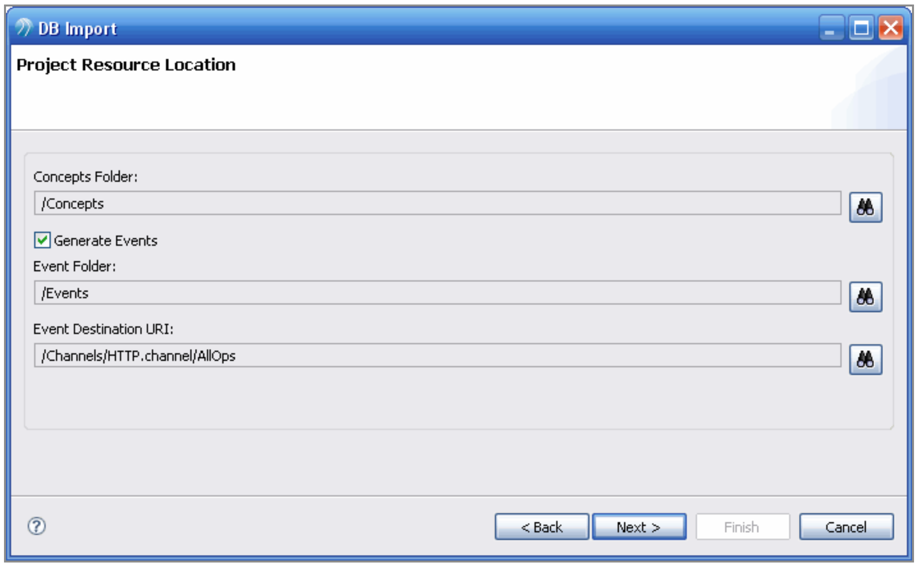
JDBC Resource URI: /SharedResources/ForDBConcept.sharedjdbc
Connect Using User: BE_USER
Connect Using Password: *******
Owner Name: BE_USER

3. In the **User SQL Query** field, type a query that returns the schema name and the table name that you want to import.

> **Note:** The **User SQL Query** field appears only if `dbimport.use.sql` is set to true in `studio.tra`. For more details, see Importing Selected Database Tables .

4. If needed, you can override the database connection details. Click **Next**.

   You see the Project Resource Location dialog.



Concepts Folder: /Concepts

☑ Generate Events

Event Folder: /Events

Event Destination URI: /Channels/HTTP.channel/AllOps

5. In the **Concept Folder** field, browse to and select the project folder that will contain the folder of database concepts.

   Note:
   The same folder name is used for concepts and for events. For example, if you are using a schema called HR, then HR is used as the project folder name containing the database concepts—and it is also the folder name used for their corresponding events.

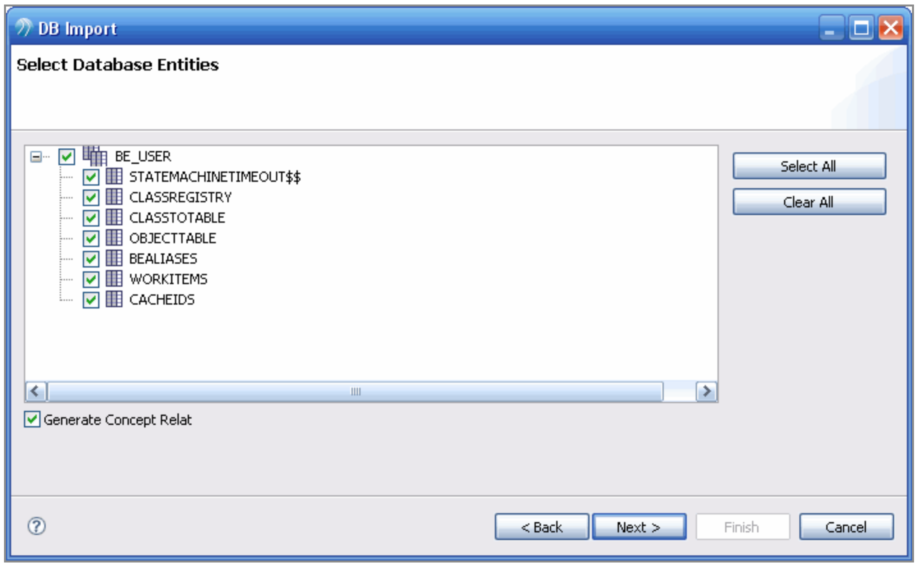
   If you specify the *same parent* folder for the database concepts and for the corresponding events, then the event and concept definitions are stored in the same folder.

6. (Optional) Check the **Generate Events** checkbox to create a simple event for each imported concept. When you check the Generate Events checkbox, additional fields appear for you to specify details for the events.

   - In the **Events Folder** field, browse to and select or create the project folder for the events.
   - In the **Event Destination URI** field, click the Browse button and select the default destination for the events.

7. Click **Next**.

   You see the Select Database Entities dialog.



DB Import

Select Database Entities

- [✓] BE_USER
  - [✓] STATEMACHINETIMEOUT$$
  - [✓] CLASSREGISTRY
  - [✓] CLASSTOTABLE
  - [✓] OBJECTTABLE
  - [✓] BEALIASES
  - [✓] WORKITEMS
  - [✓] CACHEIDS

[✓] Generate Concept Relat

8. Select the database entities (tables or views) from which you want to create database concepts (and events if you selected that option).

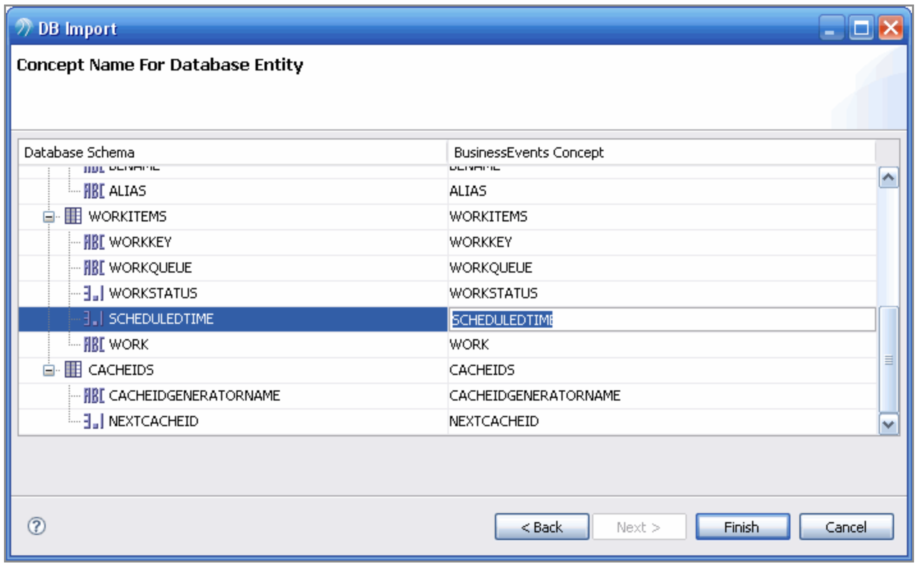
9. Select the **Generate Concept Relationships** checkbox to create relationships between concepts based on database constraints. See Table Constraints and Concept Relationships for more details about relationships.

   The utility imports any additional database entities that were not in the original selection but that must be imported because they have a relationship to the selected subset of entities.

   If this is the case, the message "*N* other related entities would be imported in addition to selected entities" displays.

10. Click **OK** to import the selected entities along with their related entities, and click **Next**.

11. At the Concept Name For Database Entity dialog, database schema names are provided as default names of the TIBCO BusinessEvents concepts.



Concept Name For Database Entity

Database Schema	BusinessEvents Concept
ALIAS	ALIAS
WORKITEMS	WORKITEMS
WORKKEY	WORKKEY
WORKQUEUE	WORKQUEUE
WORKSTATUS	WORKSTATUS
SCHEDULEDTIME	SCHEDULEDTIME
WORK	WORK
CACHEIDS	CACHEIDS
CACHEIDGENERATORNAME	CACHEIDGENERATORNAME
NEXTCACHEID	NEXTCACHEID

12. Edit the TIBCO BusinessEvents concept names as desired, then click **Finish**.

Result

The utility creates the concepts (and events if you chose that option). Browse your project tree to verify that the expected concepts and events have been created.

See Importing Domain Model Information from a Database Concept to import domain model entries for a property in the database concept.

Importing Selected Database Tables

By default, the DB Import utility imports all the schemas along with the tables within the schemas. Alternatively you can choose which tables to import.

Procedure

1. In *BE_HOME*/studio/eclipse/configuration/studio.tra, set the following property:

```
be.dbconcepts.dbimport.use.sql=true
```

2. Restart TIBCO BusinessEvents Studio for the change to take effect.

   Setting this property to true adds an additional field, **User SQL Query**, to the DB Import dialog.

   Specifying Tables to Import

3. In the User SQL Query field, enter a query that returns the schema name and the table name that you are interested in importing. You can use the `WHERE` clause appropriately in the query to selectively import database tables.

   > **Note:** While specifying the tables to be imported, you must specify the schema name first and then the table name. When you use User SQL Query, the **Owner Name** field has no effect.

   For example, when using Oracle 11g database, specify the following query to import the tables whose names start with ’TABLE_’:

   ```
   select schema_name, table_name from user_tables where table_name
   like 'TABLE_%’
   ```

   Below is an example of User SQL Query for IBM Z/ series database:

   ```
   Select creator, name from sysibm.systables where creator=’username’
   and name like ‘TABLE_%’
   ```

   Below is an example of User SQL Query for IBM I/ series database:

   ```
   select TABLE_SCHEMA,TABLE_NAME from sysibm.TABLES where TABLE_
   SCHEMA='schemaname' and TABLE_NAME like 'TABLE_%'
   ```

Importing Domain Model Information from a Database Concept

You can create a domain model by importing values from the database column that corresponds to a database concept property.

Domain models make data entry more reliable. See *TIBCO BusinessEvents Developer Guide* for details.

You can also import domain model information from a Microsoft Excel spreadsheet or from a database. Oracle Database is supported. You can also manually enter domain model information.

After you create the domain model, you associate it with the property you want to use it with. To do this you can right-click an entity property in TIBCO BusinessEvents Studio Explorer and select Associate Domains. Alternatively you can display the entity properties in the entity editor, and in the Domain field of a property, browse to the domain model you want to use.

Procedure

1. In TIBCO BusinessEvents Studio Explorer, do one of the following:
   - Right-click the folder where you want to create the domain model and select **Import > TIBCO BusinessEvents > Domain Model**.
   - In TIBCO BusinessEvents Studio Explorer, select any item in the project entity and select **File > Import > TIBCO BusinessEvents > Domain Model**.
2. Click **Next**.

   If you invoked the import wizard by right-clicking a folder, that folder is selected as the parent folder. You can choose a different one as desired.

   You see the Import Domain Model Wizard.
3. In the **Domain Import Source** field, select **DBCONCEPT**.
4. In the **File Name** field, enter a name for the domain model resource. Optionally enter a description.
5. In the **Data Type** field, select the appropriate data type for the domain model and click **Next**.
6. In the **Select DB Concept to Use** field, browse and select a database concept.
7. From the **Properties** drop-down list, select the property whose values you want to use as the domain model values.
8. Click **Finish**.

   You see a message, "Domain Import Successful."
9. Click **OK**.

   You see the Domain Model editor. The column values appear as domain entries. You can add, edit, duplicate, and remove entries as appropriate.

Configuring Database Concepts

You can configure database concept metadata properties, set in the concept editor. Also, you can configure options in the Cluster Deployment Descriptor (CDD) file, for example, to enable or disable database concepts in a processing unit.

For details on the concept editor, see *TIBCO BusinessEvents Developer Guide* and for configuring the CDD, see *TIBCO BusinessEvents Configuration Guide*.

Note: Database concepts cannot be enabled for inclusion in a backing store. Any backing store settings are ignored.

Configuring Database Concepts Metadata Properties

Database concepts and their properties have metadata properties. You may need to change the metadata properties in some cases.

For example, when the underlying column name in the database is changed later on, you need to change the property name or column name mapping.

The metadata properties for a concept are available on the Metadata Properties section of the concept resource. The metadata properties for a property are available on the shortcut menu of that property.

See Metadata Properties Reference for details on properties.

Tip: Metadata properties are disabled for concepts other than database concepts. If you want to use an existing concept as a database concept, you can enable the metadata properties so that you can manually configure them as needed. In the title bar, click the **Enable Metadata Configuration** ( ) button to enable metadata properties for editing.

Procedure

1. In TIBCO BusinessEvents Studio, open the database concept for editing.
2. Expand the Metadata properties section to see the settings, and set the following guidelines in Metadata Properties Reference.
3. To configure metadata properties for a property, in the **Properties** section, right-click in the row of the property whose metadata properties you want to display and select **Metadata Properties**.

   You see the Metadata Properties dialog, showing the property's metadata properties. Set the values following guidelines in Metadata Properties Reference.
4. Save the resource.

Metadata Properties Reference

Metadata Properties enable you to configure metadata information for the database concepts and their properties.

Metadata Properties for a Database Concept

Property Name	Global Variables supported?	Description
`OBJECT_NAME`	Yes	Name of the database table or view.
`PRIMARY_KEY_PROPS`	No	Displays a comma-separated list of names of database concept properties that are derived from primary key columns in the database, enclosed in square brackets, for example, `[ORDER_ID, ITEM_ID]`. The values are generated at import time. If there are no primary key columns, the square brackets are empty. The value of `PRIMARY_KEY_PROPS` is used to generate the `extId` (external ID) for a database concept instance. The value is overridden by the value of `EXTID_PROPS`.

Property Name	Global Variables supported?	Description
		Primary key information from `PRIMARY_KEY_PROPS` is also used in some RDBMS catalog functions. (The value is not overridden by `EXTID_PROPS` for this purpose.)
`EXTID_PROPS`	No	Not created by the import utility. Add as needed. If none of the columns imported for a database concept are primary keys (and so `PRIMARY_KEY_PROPS` has no value), or if you want to override the value of `PRIMARY_KEY_PROPS`, use the `EXTID_PROPS` metadata property to define an `extId` value. The value of these property names at runtime should uniquely identify a database concept instance. Set the value to a comma-separated list of database concept property names. The value is case sensitive. Unlike `PRIMARY_KEY_PROPS`, square brackets are not required. The value of `EXTID_PROPS` overrides the value of `PRIMARY_KEY_PROPS`.
`JDBC_RESOURCE`	No	A JDBC URI to which this database concept maps.
`OBJECT_TYPE`	Yes	• T for table • V for view
`SCHEMA_NAME`	Yes	Name of the database schema from where this concept originates.

Metadata Properties for a Database Concept Property

Property Name	Description
Primitive Properties	

Property Name	Description
`COLUMN_NAME`	Name of the column in the database.
`DATA_TYPE`	Data type as specified in the database.
`LENGTH`	Length as defined in the database.
`PRECISION`	Precision as defined in the database.
Relationship Properties	
`REL_TYPE`	C - If the concept held in this property is a contained concept. R - If it is a reference. **Note:** All relationships are modeled as references, even those defined as contained concepts.
`REL_KEYS`	A table containing join keys mapping source and target concepts. The **Name** and **Value** fields show the column names from the database tables that either form a Containment or Reference relationship. Name is the property name of the concept, and Value is the property name of the related concept.

Configuring the Database Connection

You can configure and tune the database connection or connections required for use of database concepts using various settings.

See Prerequisites for Importing Database Tables or Views for more database-related details.

Procedure

1. In TIBCO BusinessEvents Studio, open the project's Cluster Definition Descriptor and select the **Cluster tab > Database Concepts**.

2. In the **Database URIs** field, add all JDBC shared resources used for creating database concepts. Click the plus sign (+) to add a new row as needed, and add the project path to the resource.
3. As required, complete other configuration settings to tune performance, following guidelines in Database Connection Configuration Reference.
4. Save the resource.

Database Connection Configuration Reference

Configure the database connection for database concepts using the CDD file.

Note: The database concept settings and properties are applicable only for JDBC stores.

Studio Settings

Add the project paths to all JDBC resources used for database concepts, and configure other properties as desired. See Configuring the Database Connection for the procedure.

Property	Default Value	Description
`Check Interval`	`60`	The time interval (in seconds) between two consecutive checks of database connection status.
`Inactivity Timeout`	`0`	The time period (in seconds) after which the database connection in the pool closes unused connections.
`Initial Size`	`5`	The initial pool size in number of connections.
`Max Size`	`5`	Maximum number of connections allowed.
`Min Size`	`5`	Minimum number of connections to retain.
`Property Check Interval`	`900`	After this interval (in seconds), all connections which are already marked as inactive are flushed to the pool.
`Retry Count`	`-1`	Specifies behavior for reconnecting to the database. • If set to 0, the engine does not attempt to reconnect. • If set to -1, the engine retries with no limit. • If set to an integer greater than 0, the engine tries to connect for the specified number of times. When a connection is lost or stale, it cannot be recovered, and engine restart would be required.

Property	Default Value	Description
`Wait Timeout`	`1`	The maximum time (in seconds) to wait to retrieve a connection from the connection pool.
`Database URIs`		**Required**. Add the project path of all JDBC shared resources used for database concepts in the project. For example: `/SharedResources/DBConceptsConnection.sharedjdbc`

Properties

These configuration properties can be added in the CDD file. For more details on the CDD configuration see *TIBCO BusinessEvents Configuration Guide*.

Property	Default Value	Description
`be.dbconcepts.connections.checkall`	`false`	Specifies whether TIBCO BusinessEvents checks all connections for validity in DBConcepts. The values are: • true - All connections are checked in DBConcepts and if any connection fails then the connection pool is refreshed. • false - All connections are not checked in DBConcepts, and only if the test connection fails then the connection pool is refreshed.
`be.dbconcepts.connections.logall`		Specifies whether DBPool check operations are logged when `logLevel` is set to `debug`.
`be.dbconcepts.connection.check.interval`		Specifies the intervals in which the TIBCO BusinessEvents agent checks connections in the DB connection pool of DBConcepts. The agent only checks the connections that not in use currently.
`be.dbconcepts.connections.checkratio`	5	Specify the number of connections that the TIBCO BusinessEvents agent checks in each interval. Specify a number such that the following equation gives you the percentage of connections to check: `percent connections to check = (1/number)*100` For example, to check all connections on each interval set the property to 1. To check 25% of the connections set it to 4.

Property	Default Value	Description
`be.dbconcepts.oracle.pool.v12`	false	Specifies whether you can use the Oracle 12 driver for the new pool. Set this property to `true` to use the Oracle 12 driver. **Note:** For JDBC, copy `ojdbc7.jar` and `ucp.jar+` to the *BE_HOME*`\lib\ext\tpcl`.
`be.dbconcepts.use.oracle.pool`	false	For Oracle database connection, when the property is set to `true`, Oracle connection pool is used to maintain the database connection. When the property is set to `false`, BusinessEvents JDBC connection pool is used to maintain the database connection.
`be.dbconcepts.templates.jdbc.resultset.maxRows`	0	Specifies the maximum number of records (result set) that can be returned by an SQL query of database concepts. The default value `0` indicates there is no limit to the number of records that can be returned by an SQL query of database concepts.
`be.dbconcepts.query.sqltimeout`	0	Specifies the timeout value for database queries in seconds.
`be.dbconcepts.cursor.closeQuery.closeConnection`	true	Specifies whether the database transactions are auto committed when Database.closeQuery() function is executed. Set this property to `false` to prevent autocommit.
`be.dbconcepts.quote.column.names`	true	Specifies whether a case-sensitive search is enabled for names of entities like columns, tables, and schemas.
`be.dbconcepts.sqlquery.delimiters`	""	Specifies the delimiter for searching entities like columns, tables, and schemas for a case-sensitive search.

Setting the Object Management Mode of a Database Concept

The term mode refers to cache object management behavior. If a cache is enabled for the project, you can set objects (either globally or individually) to use one of three modes: Cache Only, Cache + Memory, or Memory Only.

The default mode for a cache cluster is Cache Only. However the default mode of a database concept is Memory Only. See *TIBCO BusinessEvents Developer Guide* for more details about modes.

Procedure

1. In TIBCO BusinessEvents Studio, open the project CDD file for editing.
2. In the Cluster tab, expand **Domain Objects > Overrides**.
3. Click **Add**. Select the database concept you want to override and click **OK**.
4. In the tree on the left, select the `/uri` entry for the database concept and select Cache Only or Cache+Memory.

   Note: Cache+Memory is recommended only for constants or concepts whose values change infrequently. It is not generally appropriate for database concepts. If you select Cache+Memory, additional configuration is required. See *TIBCO BusinessEvents Developer Guide* for details.

5. Save the resource.

   Note: Most other override properties relate to backing store. Database concepts cannot be enabled for inclusion in a backing store. Any backing store settings are ignored.

Enabling or Disabling Database Concepts in a Processing Unit

Configure the CDD file to enable or disable database concepts for individual processing units.

Procedure

1. In TIBCO BusinessEvents Studio, open the project’s CDD.
2. Select the **Processing Units** tab (at the bottom of the window). Select each processing unit in turn and do one of the following:
    - To enable database concepts, select the **Enable Database Concepts** checkbox.
    - To disable database concepts, clear the checkbox.
3. Save the resource.

RDBMS Catalog Functions

The RDBMS functions are used in TIBCO BusinessEvents rules and rule functions to connect to a database to perform database operations.

The database concepts feature has an object-to-relational mapping feature that enables you to act on the concepts, and delegate the persistence of these objects to the RDBMS catalog functions.

These functions allow you to perform basic Create, Retrieve, Update and Delete (CRUD) operations on database tables using database concepts, and perform related tasks. In this way, you can keep the database concepts synchronized with their database equivalents.

The RDBMS catalog has one category called Database, which contains the following functions:

- `getConnectionStatus()`
- `setCurrentConnection()`
- `unsetConnection()`
- `beginTransaction()`
- `commit()`
- `rollback()`
- `insert()`
- `update()`
- `delete()`
- `queryUsingConceptProps()`
- `queryUsingPreparedStmt()`
- `queryUsingPrimaryKeys()`
- `queryUsingSQL()`
- `assertDBInstance()`
- `executePreparedStmt()`

- `executeSQL()`
- `createQuery()`
- `closeQuery()`
- `getNextPage()`
- `getNextPageFromOffset()`
- `getPreviousPage()`
- `getPreviousPageFromOffset()`

These operations include inserting, updating or deleting database operations, or keeping the TIBCO BusinessEvents concepts and their database source synchronized.

The setCurrentConnection and unsetConnection Functions

Use `setCurrentConnection()` once before performing any database operation. Use `unsetConnection()` once after all database operations are performed.

In your rule or rule function, you must first connect to the database.

```
Database.setCurrentConnection ("/MyDbConnection");
Operations
Database.unsetConnection();
```

You must call `unsetConnection()` even in case of a failure within the rule function. If you do not unset the connection, the connection is not returned to the database connection pool.

Use of `setCurrentConnection()` may result in an exception if the underlying database is disconnected. You can first use `getConnectionStatus()` to determine if the database (specified by a JDBC connection resource URI) is connected.

Transactions

To commit multiple statements, group them using transactions.

By default, all database statements are individually committed. For example, if an insert call results in multiple insert statements, then each one gets committed individually. You can, however, use transactions. To group statements inside a transaction, call the `beginTransaction()` function before the statements, and call `commit()` after all the statements. Use `rollback()` to roll back the entire transaction in the event of an exception.

Example

```
try
{
  Database.setCurrentConnection ("/MyDbConnection");
  Database.beginTransaction ();
  Concept instance=Instance.createInstance("/someconcept");
  Database.insert(instance)
  Database.commit();
}
catch (Exception e)
{
  Database.rollback();
}
finally
{
Database.unsetConnection();
}
```

Insert Operations

The insert function `Database.insert()` inserts an object and its concept properties (if any) recursively into the database.

You can insert all related objects at once instead of performing individual inserts. The join keys are internally managed.

In the case of concept references, foreign keys in the referencing concept are updated with primary keys in the referenced concept.

In the case of contained concept properties, foreign keys in the contained concept are updated with primary keys in the container concept.

If columns in the database are modified by the database during inserts, these changes are also made in the concept instances. This is usually the case when primary keys are automatically generated or when columns have default values.

Using Generated Unique Primary Keys for Inserts

Many database products provide a mechanism for generating unique primary keys.

For each database concept that requires a primary key value to be provided in this way, you configure the mechanism provided by your database and provide it to TIBCO BusinessEvents in a stored procedure that you reference in an XML file. You can also reference an Oracle sequence in the same way. When a record is inserted into the database, the unique value generated for the specified primary key property is used. The database concept also uses the generated primary key value.

Procedure

1. Create an XML file with the extension `.sequences.xml`.

   Configure the XML file is as follows:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <unique_identifiers>
      <unique_identifier entity="ConceptURI"
                          property="PropertyName"
               unique_identifier="SequenceName"
                     stored_proc="StoredProcStatement"/>
   </unique_identifiers>
   ```

   Where, add a `unique_identifier` element for each database concept that will acquire a primary key value using this mechanism. Following are the parameters that you can define:

Parameter	Description
`entity`	The URI of the concept whose specified property will use the sequence or stored procedure.
`property`	Name of the property that holds the primary key.
`unique_identifier`	Name of the sequence (if you are using a sequence). Used for Oracle DBMS only. If both `unique_identifier` and `stored_proc` are present, `stored_proc` is used.
`stored_proc`	Name of the stored procedure (if you are using a stored procedure). The value must be a callable JDBC statement. The called stored procedure must take only one `OUT` type parameter. **Note** The syntax of the value for this attribute depends on the JDBC driver you are using for

Parameter	Description

	the database concept.
	
	For Oracle thin Driver use this syntax:
	
	`stored_proc="CALL *YourStoredProc* (?)"`
	
	For SQL Server use this syntax:
	
	`stored_proc="EXEC *YourStoredProc* ?"`

2. Add the XML file to the project as follows:
    a. Select **File > New**.
    b. Select **Other > XML > XML**.
    c. Click **Next**, type a name for the file, and click **Finish**.
3. You can also import an existing XML file. To do that, perform the following:
    a. Select **File > Import > File System**.
    b. Specify the file you want to import, and click **Finish**.

Update and Delete Operations

Perform the update or delete operations on row using their primary key.

Each instance of a database concept maps to one row in a database table. In order for the database to perform updates or deletes on the TIBCO BusinessEvents objects, or for TIBCO BusinessEvents to perform updates or deletes on the database tables, the software must be able to uniquely identify the row. Therefore, you can only perform delete and update operations if the table has at least one primary key. If you attempt to perform an update for a row that has no primary key, an exception is thrown.

To find out whether a table has primary keys or not, open the project in TIBCO Designer, and check the `PRIMARY_KEY_PROPS` metadata property, which is on the Metadata Properties tab for the concept. If this property has no value, no primary keys exist and you cannot perform update or delete operations.

update()

The `Database.update()` function updates the database with values contained in the concept.

Tip: XMLType In order to update an Oracle Database table column of datatype `XMLType` with a string value greater than 4,000 characters, you must add the following Oracle JAR files to *BE_HOME*`/lib/ext`:

- *ORACLE_HOME*`/rdbms/jlib/xdb.jar`
- *ORACLE_HOME*`/lib/xmlparserv2.jar`

Example

```
try
  {
   Database.setCurrentConnection ("/MyDbConnection");
   Database.beginTranscation ();
   /*the instance passed to update operation is an instance of the
     dbconcept*/
   Database.update(instance)
   Database.commit();
  }
catch (Exception e)
  {
    Database.rollback();
  }
finally
  {
    Database.unsetConnection();
  }
```

delete()

The `Database.delete()` function deletes a record corresponding to the concept instance in the database.

If `cascade` is set to `true`, it deletes all database records corresponding to the contained concept property references and nulls out foreign key references in database records corresponding to the concepts that refer to the concept being deleted.

Example

```
try
  {
   Database.setCurrentConnection ("/MyDbConnection");
   Database.beginTransaction ();
   /*the instance passed to delete operation is an instance of the
   dbconcept*/
   Database.delete(instance, false)
   Database.commit();
  }
catch (Exception e)
  {
   Database.rollback();
  }
finally
  {
   Database.unsetConnection();
  }
```

Query Operations

You can perform query operations on a database, to return cocepts matching those queries.

queryUsingConceptProps()

The `queryUsingConceptProps()` queries database using the property values in a concept instance.

Note: You cannot use `queryUsingConceptProps()` by specifying a value in the property that maps to a CLOB column.

Syntax

```
Concept[] queryUsingConceptProps(Concept qConcept, boolean
queryChildren)
```

In this example, `qConcept` indicates that the concept database is queried for matching values from this concept. If `queryChildren boolean` is set to true, concept properties are recursively queried. The function returns an array of concepts.

Example

```
Database.setCurrentConnection
("/SharedResources/MySQLConnection.sharedjdbc");
Database.beginTransaction();
  Concepts.test.test.test testConcept=Instance.createInstance("xslt://
{{/Concepts/test/test/test}}<?xml version=\"1.0\" encoding=\"UTF-
8\"?>\n<xsl:stylesheet xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsl=\"http://www.w3.org/1999/XSL/Transform\" version=\"1.0\"
exclude-result-prefixes=\"xsl xsd\">\n    <xsl:output method=\"xml\"/>\n
   <xsl:template match=\"/\">\n        <createObject>\n
<object>\n                <col_1>\n                    <xsl:value-of
select=\"1\"/>\n                </col_1>\n                <col_2>\n
              <xsl:value-of select=\"&quot;Hi&quot;\"/>\n
</col_2>\n            </object>\n        </createObject>\n
</xsl:template>\n</xsl:stylesheet>");
Concepts.test.test.test[] result=Database.queryUsingConceptProps
(testConcept,true);
for (int i=0;i<result@length;i++)
{
   System.debugOut("properties from DB are " + result[i].col_1 + " ," +
result[i].col_2);
}
Database.unsetConnection();
```

queryUsingPreparedStmt()

Use `queryUsingPreparedStmt()` to query the database using prepared statements.

Syntax

```
Concept[] queryUsingPreparedStmt(String conceptURI, String preparedStmt,
Object[] args,boolean queryChildren)
```

Here, `conceptURI` is the URI of the result concept type, `preparedStmt` is the prepared statement to be executed, and `args Object[]` are the positional values to be used for binding to the prepared statement. If `queryChildren boolean` is set to `true`, the concept properties are recursively queried. The function returns an array of concepts matching the query.

Example

```
Database.setCurrentConnection("/SharedResources/Mainframe");
Database.beginTransaction();
String stm="delete from ACME1.TABLE_BETA where ID1=?";
Object []o3 ={3};
int cnt_prepare_delete=Database.executePreparedStmt(stm,o3);
System.debugOut("output of function executePreparedStmt(delete) is " +
cnt_prepare_delete);
Database.commit();
Database.unsetConnection()
```

queryUsingPrimaryKeys()

Use the `queryUsingPrimaryKeys()` function to query the database using primary keys.

Syntax

```
Concept[] queryUsingPrimaryKeys(String conceptURI, SimpleEvent
pKeyEvent, boolean queryChildren)
```

Here `conceptURI` is the URI of the concept type, `pKeyEvent` is a simple event that contains primary key values to be used. Properties must match primary key properties in the result concept. If `queryChildren boolean` is set to true, concept properties are recursively queried. The function returns an array of result concepts.

Example

```
Database.setCurrentConnection("/SharedResources/ibmi");
Database.beginTransaction();
  EventsNew.ADBTEST1.TABLE_REFRING event = Event.createEvent("xslt://
{{/EventsNew/ADBTEST1/TABLE_REFRING}}<?xml version=\"1.0\"
encoding=\"UTF-8\"?>\n<xsl:stylesheet
xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsl=\"http://www.w3.org/1999/XSL/Transform\" version=\"1.0\"
exclude-result-prefixes=\"xsl xsd\">\n    <xsl:output method=\"xml\"/>\n
   <xsl:template match=\"/\">\n        <createEvent>\n
<event>\n                <ID2>\n                    <xsl:value-of
select=\"10\"/>\n                </ID2>\n            </event>\n
</createEvent>\n    </xsl:template>\n</xsl:stylesheet>");
Concepts.ADBTEST1.TABLE_REFRING [] parent =
Database.queryUsingPrimaryKeys("/Concepts/ADBTEST1/TABLE_
REFRING",event,true);
for(int k = 0 ; k < parent@length ; k++ ) {
   System.debugOut(Instance.serializeUsingDefaults(parent[k]));
}
Database.unsetConnection();
```

queryUsingSQL()

Use the `queryUsingSQL()` function to query the database using SQL statements.

Syntax

```
Concept[] queryUsingSQL(String conceptURI, String sql, boolean
queryChildren)
```

Where, `conceptURI` is the URI of the result concept type, and `sql` is the SQL statement to be executed. If `queryChildren boolean` is set to true, concept properties are recursively queried. The function returns an array of result concepts.

Example

```
Database.setCurrentConnection("/SharedResources/Mainframe");
Database.beginTransaction();
Concepts.BEUSER.CAR_TABLE []ct = Database.queryUsingSQL
("/Concepts/BEUSER/CAR_TABLE","select * from BEUSER.CAR_TABLE",true);
```

```
for(int i=0;i<ct@length;i=i+1)  {
   System.debugOut(""+ct[i].CAR_ID+ct[i].TIRE_TABLE[0].CAR_ID+ct
[i].TIRE_TABLE[0].TIRE_ID);
}
Database.unsetConnection();
```

Database Concept Assertion After Database Query

Several RDBMS catalog functions enable you to query the database. Concepts returned by a database query are not automatically asserted. They must be explicitly asserted.

To assert the database concept, use the following function:

```
Database.assertDBInstance(concept, deep)
```

This function returns void.

When a database concept is asserted with the *deep* parameter set to `true`, all the referenced and contained concepts are also asserted (concept properties are asserted recursively).

For example:

```
void assertDBInstance(Concept MyDBConcept, boolean deep)
```

Where, `MyDBConcept` is the instance of the concept to be asserted to working memory.

Prepared Statements

A prepared statement is an SQL statement whose values are determined at runtime. You can use prepared statements in the `queryUsingPreparedStmt()` and `executePreparedStmt()` functions.

When you use `executePreparedStmt()`, prepare the statement, for example:

```
String QueryCON="Insert into HR.COUNTRIES Values (?,?,?)"
```

Then associate the actual values at execution time, as shown in the following code snippet. It demonstrates use of `executePreparedStmt()` and `queryUsingPreparedStmt()`.

```
Database.setCurrentConnection("/JDBC Connection");
Database.beginTransaction();
String InsQuery="INSERT into HR.COUNTRIES Values (?,?,?)";
Object []InsObj={e.rate, e.CID,e.rank};
int insertCON=Database.executePreparedStmt(InsQuery,InsObj);
String UpdateQuery="UPDATE HR.COUNTRIES SET COUNTRY_ID=? WHERE COUNTRY_
ID=?"
Object [] update={Use.Updates,Use.UpdateV};
int updat=Database.executePreparedStmt(UpdateQuery,update);
String DeleteQuery="DELETE FROM HR.COUNTRIES WHERE COUNTRY_ID=?"
Object []DelObj={e.CID};
int deleteCON=Database.executePreparedStmt(DeleteQuery,DelObj);
String SelectQuery="Select * from HR.COUNTRIES where
HR.COUNTRIES.COUNTRY_ID>?"
Object []SelObj={e.CID};
Concept []SelectCON=Database.queryUsingPreparedStmt(
        "/Concepts/hr/HR/COUNTRIES",SelectQuery,SelObj,true);
Database.commit();
Database.unsetConnection();
```

Database Cursor Functions

Database cursor functions are useful when you want to process few records at a time in a large result set of queries. They help you to get database records, insert database records, and delete database records.

The following database functions are available in the catalog functions under RDBMS:

- `createQuery()`
- `getNextPage()`
- `getPreviousPage()`
- `getNextPageFromOffset()`
- `getPreviousPageFromOffset()`

- `closeQuery()`

Example of Database Cursor Functions

```
debugOut("Database Cursor Demo Starts");
Database.setCurrentConnection("/SharedResources/HR_DB_Conn.sharedjdbc");
String cursorName=Database.createQuery("/SharedResources/HR_DB_
Conn.sharedjdbc", "EmpCursor", "/Concepts/HR/EMPLOYEES", "select * from
employees", 10, null);
debugOut("Opened Cursor: " + cursorName);
try{
Concept[] empCepts=Database.getNextPage(cursorName, 10);
debugOut("Database.getNextPage() fetched " + empCepts@length + " rows");
 while(empCepts !=null && empCepts@length > 0){
empCepts=Database.getNextPage(cursorName, 10);
debugOut(" Database.getNextPage() fetched " + empCepts@length + "
rows");
for(int i; i < empCepts@length ; i=i+1){
      debugOut(" "+empCepts[i]);
      }
   }
debugOut("@ end");
} finally {
   if(cursorName !=null){
   Database.closeQuery(cursorName);
   }
}
```

createQuery()

You can open a database cursor for an SQL query by using the `Database.createQuery()` function. Once the cursor is open, you can retrieve large result sets from the database in pages.

Syntax

```
String createQuery(String jdbcURI, String cursorName, String
resultTypeURI, String sql, int pageSize, Object requestObj);
```

Parameters

Parameter	Type	Description
`jdbcURI`	`String`	The JDBC URI of the resource to be used for getting the connection.
`cursorName`	`String`	The name of the cursor to be opened.
`resultTypeURI`	`String`	The URI of the result type, which will be returned by the query. Possible values are Concept URI, Map, or Null. If it is of "Map" type, then each result set row is a Map object. In this case, you can retrieve the column values by specifying column names as the key. If Null, an `Object[]` is returned.
`sql`	`String`	The SQL query string. It can be a prepared statement query or a simple statement query.
`pageSize`	`Integer`	The number of concepts or records to be fetched from the database for each page.
`requestObj`	`Object`	If the SQL query specified is a prepared statement query, then the `requestObject` can be specified as a concept or an array of arguments.
`returns`	`String`	The name of the opened cursor.

Example

```
Database.setCurrentConnection("/SharedResources/Oracle.sharedjdbc");
String jdbcURI="/SharedResources/HR_DB_Conn.sharedjdbc";
String cursorName="EmpCursor";
String resultTypeURI="/Concepts/HR/EMPLOYEES";
String sql="select * from employees";
int pageSize=10;
Object requestObj=null;
String cursorName=Database.createQuery
(jdbcURI,cursorName,resultTypeURI,sql,pageSize,requestObj);
System.debugOut(" Opened Cursor: " + cursorName);
```

getNextPage() and getPreviousPage()

You can use the `Database.getNextPage()` and `Database.getPreviousPage()` functions to access next or previous pages respectively from the database cursor.

The `getPreviousPage()`function fetches records in the forward direction. That is, if the cursor is on the 11th record and you call this function, it returns records from 1 to 10, and not from 10 to 1.

The `getNextPage()` function returns an empty array when it reaches the end of the result set. Similarly, the `getPreviousPage()` function returns an empty array when it reaches the beginning of the result set.

Syntax

```
Object [] getNextPage(String cursorName, int pageSize)
Object [] getPreviousPage(String cursorName, int pageSize)
```

Parameters

Parameter	Type	Description
`cursorName`	`String`	The name of the database cursor.
`pageSize`	`Integer`	The number of concepts or records to be fetched from the database for each page. If the `pageSize` is -1, then the default page size given for the cursor is used.

Parameter	Type	Description
		The default is 500.
`return Object []`	`Object[]`	The array of the returned resultset data. If `resultConceptURI` is specified, the array is of `resultConceptType`. Else, it is in the form of n-tuple object array, where each tuple is an array of the values of the returned resultset data.

Example for getNextPage

```
Database.setCurrentConnection("/SharedResources/Oracle.sharedjdbc");
String jdbcURI="/SharedResources/HR_DB_Conn.sharedjdbc";
String cursorName="EmpCursor";
String resultTypeURI="/Concepts/HR/EMPLOYEES";
String sql="select * from employees";
int pageSize=10 ;
Object requestObj=null;
String cursorName=Database.createQuery
(jdbcURI,cursorName,resultTypeURI,sql,pageSize,requestObj);
System.debugOut(" Opened Cursor: " + cursorName);
   Concept[] empCepts={};
   empCepts=Database.getNextPage(cursorName, 10);
   System.debugOut(" Database.getNextPage() fetched " + empCepts@length
+ " rows");
```

Example for getPreviousPage

```
Database.setCurrentConnection("/SharedResources/Oracle.sharedjdbc");
String jdbcURI="/SharedResources/HR_DB_Conn.sharedjdbc";
String cursorName="EmpCursor";
String resultTypeURI="/Concepts/HR/EMPLOYEES";
String sql="select * from employees";
int pageSize=10 ;
Object requestObj=null;
String cursorName=Database.createQuery
(jdbcURI,cursorName,resultTypeURI,sql,pageSize,requestObj);
```

```
System.debugOut(" Opened Cursor: " + cursorName);
   Concept[] empCepts={};
   empCepts=Database. getPreviousPage (cursorName, 10);
   System.debugOut(" Database. getPreviousPage () fetched " +
empCepts@length + " rows");
```

getNextPageFromOffset() and getPreviousPageFromOffset()

While paging a result set, you can skip a number of rows or records. This set of records to be skipped is called an Offset. Use the `Database.getNextPageFromOffset()` and `Database.getPreviousPageFromOffset()` functions to get the next or previous pages respectively of the database cursor, starting from an offset.

Syntax

```
Object [] getNextPageFromOffset(String cursorName, int startOffset, int
pageSize)
Object [] getPreviousPageFromOffset (String cursorName, int startOffset,
int pageSize)
```

Parameters

Parameter	Type	Description
`cursorName`	`String`	The name of the database cursor.
`startOffset`	`Integer`	The start offset for the page.
`pageSize`	`Integer`	The number of concepts or records to be fetched from the database for each page. If the `pageSize` is -1, then the default page size given for the cursor is used.

Parameter	Type	Description
		The default is 500.
`return Object []`	`Object[]`	The array of the returned resultset data. If `resultConceptURI` is specified, the array is of `resultConceptType`. Else, it is in the form of n-tuple object array, where each tuple is an array of the values of the returned resultset data.

Example for getNextPageFromOffset

```
Database.setCurrentConnection("/SharedResources/Oracle.sharedjdbc");
String jdbcURI="/SharedResources/HR_DB_Conn.sharedjdbc";
String cursorName="EmpCursor";
String resultTypeURI="/Concepts/HR/EMPLOYEES";
String sql="select * from employees";
int pageSize=10 ;
Object requestObj=null;
String cursorName=Database.createQuery
(jdbcURI,cursorName,resultTypeURI,sql,pageSize,requestObj);
System.debugOut(" Opened Cursor: " + cursorName);
Concept[] empCepts={};
empCepts=Database.getNextPageFromOffset(cursorName,2,10);
System.debugOut(" Database. getPreviousPage () fetched " +
empCepts@length + " rows");
```

Example for getPreviousPageFromOffset

```
Database.setCurrentConnection("/SharedResources/Oracle.sharedjdbc");
String jdbcURI="/SharedResources/HR_DB_Conn.sharedjdbc";
String cursorName="EmpCursor";
String resultTypeURI="/Concepts/HR/EMPLOYEES";
String sql="select * from employees";
int pageSize=10 ;
Object requestObj=null;
String cursorName=Database.createQuery
(jdbcURI,cursorName,resultTypeURI,sql,pageSize,requestObj);
```

```
System.debugOut(" Opened Cursor: " + cursorName);
Concept[] empCepts={};
empCepts=Database.getPreviousPageFromOffset(cursorName,5,10);
System.debugOut(" Database. getPreviousPage () fetched " +
empCepts@length + " rows");
```

closeQuery()

You must close an open database cursor for an SQL query by using the `Database.closeQuery()` function.

If you do not close the cursor, it results in database level exception as an open cursor is a limited resource at database level. Even if there is no exception, you must close all open cursors.

Syntax

```
void closeQuery(String cursorName);
```

Parameters

Parameter	Type	Description
`cursorName`	`String`	The name of the cursor to be closed.

Example

```
Database.setCurrentConnection("/SharedResources/Oracle.sharedjdbc");
String jdbcURI="/SharedResources/HR_DB_Conn.sharedjdbc";
String cursorName="EmpCursor";
String resultTypeURI="/Concepts/HR/EMPLOYEES";
String sql="select * from employees";
int pageSize=10 ;
Object requestObj=null;
String cursorName=Database.createQuery
(jdbcURI,cursorName,resultTypeURI,sql,pageSize,requestObj);
```

```
System.debugOut(" Opened Cursor: " + cursorName);
Database.closeQuery(cursorName);
```

Oracle Catalog Functions

Oracle catalog functions are extended functions that can be used to perform database operations in rules and rule functions.

Add the following property in the `studio.tra` file to use the Oracle catalog functions.

```
TIBCO.BE.function.catalog.Oracle=true
```

Default value of the property is `false`. In the Built-in Functions Oracle catalog, this property enables/disables the following functions:

- `closeResultSet`
- `commit`
- `executeQuery`
- `getColumnValueByIndex`
- `getColumnValueByName`
- `getConnection`
- `getConnectionWithTimeout`
- `next`
- `registerConnection`
- `releaseConnection`
- `rollback`

The Oracle catalog function `executeQuery` is used only to select queries and not to perform the insert, update, and delete operations. Use the RDBMS catalog function to perform the insert, update, and delete operations, see Insert Operations and Update and Delete Operations.

The `registerConnection` function creates a pool of database connections and registers with the specified key. The syntax of the `registerConnection` function is:

```
void registerConnection(String key,String uri,int poolSize)
```

In the `registerConnection` function, the poolsize parameter defines the required number of database connections in the pool. The settings of the parameter depend on your project requirements, such as, using more connections improves the runtime performance when the registered connection is used in multiple rules.

Example

The following code shows the example usage of the Oracle catalog functions:

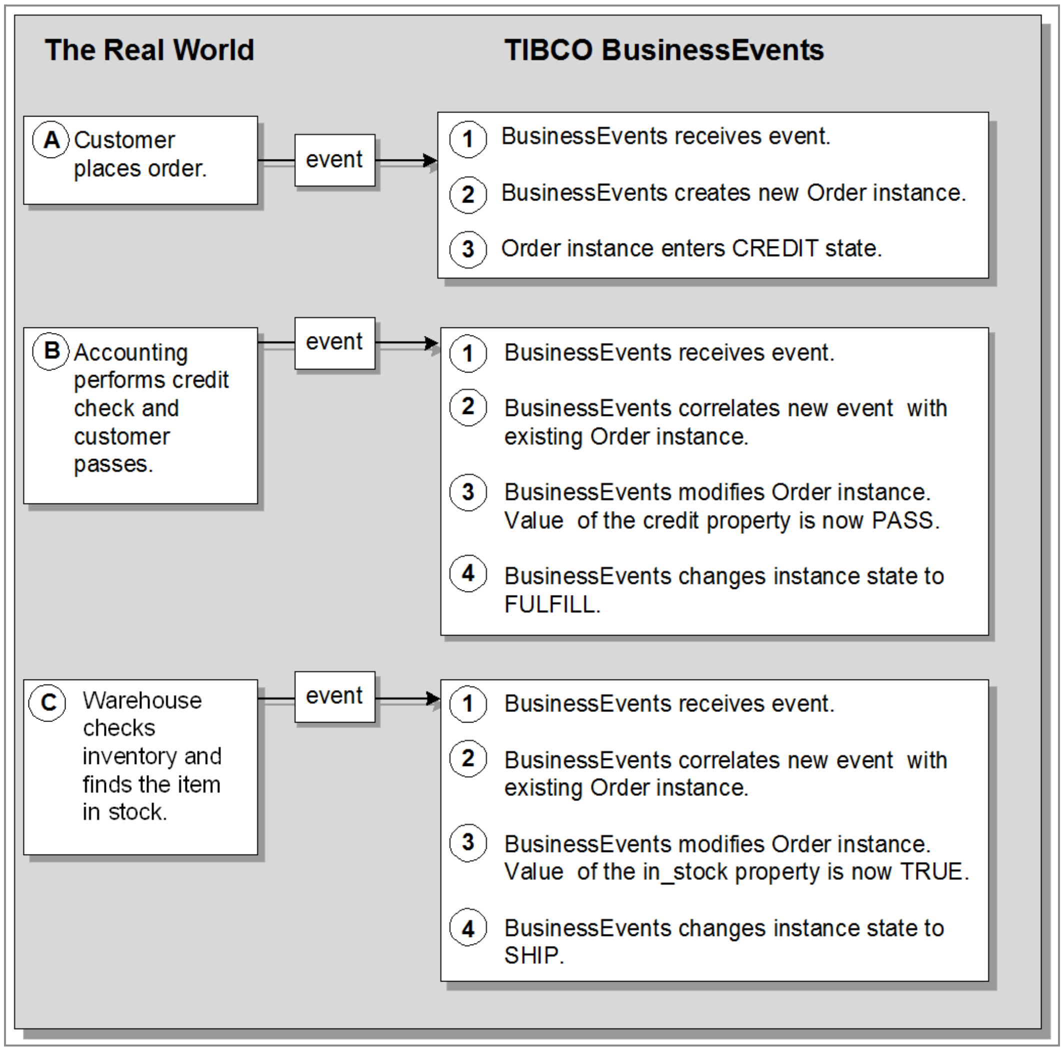
```
String connection ="/SharedResources/JDBCConnection.sharedjdbc";
Oracle.registerConnection("sstConnection",connection, 10);
Object cnx = Oracle.getConnection("sstConnection");
String sRequest =  "select to_char(count(*)) as NB from d_account";
String  nextResponse ;
Object[] args = {};
Object res = Oracle.executeQuery(cnx, sRequest, args);
if (res!= null){
   while(Oracle.next(res)) {
      nextResponse =  String.trim(Oracle.getColumnValueByName
(res,"NB"));
    }
}
Oracle.closeResultSet(res);
Oracle.releaseConnection(cnx);
```

State Modeler

State Modeler is based on the UML-standard definition for state models. A state model is a directed graph. States are represented by nodes and state transitions are represented by connectors.

A state model allows you to model the life cycle of a concept instance — that is, for each instance of a given concept, you can define what states the instance passes through and how it will transition from state to state.

For example, consider an order process. It is modeled using a state model that is associated with an Order concept.


The Real World
TIBCO BusinessEvents
A Customer places order.
event
1 BusinessEvents receives event.
2 BusinessEvents creates new Order instance.
3 Order instance enters CREDIT state.
B Accounting performs credit check and customer passes.
event
1 BusinessEvents receives event.
2 BusinessEvents correlates new event with existing Order instance.
3 BusinessEvents modifies Order instance. Value of the credit property is now PASS.
4 BusinessEvents changes instance state to FULFILL.
C Warehouse checks inventory and finds the item in stock.
event
1 BusinessEvents receives event.
2 BusinessEvents correlates new event with existing Order instance.
3 BusinessEvents modifies Order instance. Value of the in_stock property is now TRUE.
4 BusinessEvents changes instance state to SHIP.

In the above example, when a customer places an order, TIBCO BusinessEvents receives a simple event and creates a new instance of the Order concept. Through the life of the Order instance, it passes through a credit-check state, an inventory-check state, a fulfillment state, and so on until the customer has received and paid for the order.

As these order activities take place, TIBCO BusinessEvents receives new events, correlates the events to the existing instance, modifies one or more property values within the instance, and changes the state of the instance.

TIBCO BusinessEvents ships with a simple state model example, which you can open and examine. It also includes a document with instructions for creating the example yourself. This will help to familiarize you with the TIBCO BusinessEvents State Modeler. This and other examples are stored in the *BE_HOME*`/examples` directory.

State Modeler Functions

The State Modeler includes its own set of functions, which are located in the Standard function registry view, under `Instance > StateMachine`.

Only one is commonly used: `Instance.startStateMachine`. It is used only when auto start of state machines is disabled (see Controlling the Start of a State Machine).

State Models and Concepts

Each state model is owned by a concept. One concept can own multiple state models, but has only one *`main`* state model. The main state model can call the other state models.

Each state model begins with one *start* state and ends with one or more *end* states. Between the start and end states you can add *simple*, *composite*, and *concurrent* states as needed, connected by *transitions*. You can also add a Call State Model node to call another state model from within a state model.

Main State Model Inheritance

Each concept is allowed at most one main state model. However, a concept can inherit its main state model. At runtime, TIBCO BusinessEvents searches for a main state machine, starting with the concept instance. If it does not find one, it moves up the inheritance chain until it finds a main state machine. It creates an instance of the first main state machine it locates.

At runtime, if TIBCO BusinessEvents cannot locate a main state machine for the concept instance or for any concepts higher in the inheritance chain, it does not create an instance of any state machine for the concept instance.

A concept’s state machines can call any state machine that belongs to an ancestor of the concept. The state machine of a concept cannot call the state machine of a concept that is lower in the inheritance chain.

Call State Model

State model inheritance is used when you call another state model from within a state model: you can specify whether the call is an implicit or explicit call. See Call State Machine Resource.

State Models and Rules

State models define behaviors relating to a concept during its lifecycle. The TIBCO BusinessEvents rule language is used to define many of these behaviors.

Transitions are containers for rules, which control how each instance changes states and property values.

Entry Actions and Exit Actions

As a concept instance enters a state, it executes the associated entry action. While it is within a state, the state’s transition rules determine if, when, and how it will exit the state. As it exits a state, it executes the associated exit action.

Transition Rules

Transition rules determine when, or if, the transition is taken. If you define a rule for the transition, the transition does not occur until the rule executes successfully. See Transitions.

Timeout Expressions and Actions

States and the overall state model can have timeout expressions and actions. The rule language is used to define timeout expressions and timeout actions as explained in State Model Timeouts and State Timeouts.

State Machines at Runtime

At runtime a state model executes as a state machine, which begins in the start state.

By default a state machine starts when its concept instance is asserted. When a concept is asserted so are its contained concepts, if any, so all its contained concept state machines are also started.

However you can change this default behavior and instead use a function in your rules to start the state model, at some point after the concept is asserted. The function has a parameter that controls whether contained concept state models are also started.me

The state machine begins in the start state. If the transition to the next state has a rule, then the state stays at start until the rule's conditions are met. When they are met the following occurs:

- The start state's exit action executes (if one exists)
- The transition rule actions execute (if any exist)
- If the transition to the next state crosses more than one composite or concurrent state boundary before ending at its target, all entry actions associated with those boundaries execute.
- The next state's entry action executes (if it exists).

The same general process repeats as the machine moves from state to state, with the addition that there could be multiple exit actions, if the transition crosses any enclosing boundaries on its way to its target.

When a state machine exits an enclosing state, it exits all the enclosed states too.

If a state or state machine times out, then other actions occur as explained in State Model Timeouts and State Timeouts

Types of States

There are different types of states a concept instance passes through in the state model.

Note:

- Except for the concurrent state, all state types allow only exclusive-or (XOR) transitions, which go to one state only. To allow a state to go from one state to multiple states, use a *concurrent* state.
- You can set timeout expressions and timeout actions on all types of states except start and end states, and Call State Model nodes.

Start and End States

Each state model begins with a start state and ends with one or more end states. A start state has an exit action, but no entry action. An end state has an entry action but no exit action.

Composite and concurrent states also contain start and end states. You can delete the end state in a composite state, as it is optional. However, you cannot delete the end states in a concurrent state as they are mandatory.

Simple States

A simple state can include an entry action and an exit action.

Composite States

Composite states are like nested folders: they contain other states.

Composite states can contain simple states, other composite states, and concurrent states.

By default a composite state has a start and end state. However you can delete the end state if it is not needed.

Use a composite state to treat a group of states as a unit. For example, consider a state model for an order instance. The order instance may need to travel through complex credit check and fulfillment processes. You can group the complex credit-check process in one composite state and the order-fulfillment process in another composite state.

For information about composite state transition rules, see Transitions.

Concurrent State

A concurrent state allows multiple state flows to operate at the same time. By default, a concurrent state has two processing lanes, called *regions*. A region can contain simple states, other concurrent states, and composite states. You can add multiple regions to a concurrent state.

A state machine instance cannot exit a concurrent state until all its regions have finished processing, unless a timeout occurs.

For information about concurrent state transition rules, see Transitions.

Adding Regions in a Concurrent State

By default, a concurrent state has two regions, but more can be added to it.

Procedure

1. Open a state machine with a concurrent state.
2. Click to select the concurrent state to which you want to add regions.
3. In the Palette view, click **Region**.
4. Click inside the concurrent state.

   A new region appears.

Transitions

The states in a state model are connected by transitions.

Transition Rules

A transition can optionally be triggered by a rule. If a transition has no rule (a lambda transition) then as soon as the prior state's exit action (if any) has occurred, the state model moves to the next state (or to its entry action if it has one).

As with any rule, you can add advisory events, simple events, time events, concepts, and scorecards to the declaration section of a transition rule.

If you define a rule for the transition, the transition does not occur until the rule executes successfully.

Transition Rules Execute Once

In one respect, transition rules work like other TIBCO BusinessEvents rules: they participate in run-to-completion cycles. However, unlike regular rules, a transition rule executes at most only once.

For example, suppose a regular TIBCO BusinessEvents Studio rule has two entities in its scope (declaration) and has no condition. Such a rule might execute multiple times, for all matching tuples in the Rete network. However a transition rule executes only once, for the first matching tuple.

To understand more about rule behavior at runtime, such as the priority setting, the rule scope (declaration) and rule conditions, see TIBCO BusinessEvents Architect Guide. Rule configuration is documented in TIBCO BusinessEvents Developer Guide.

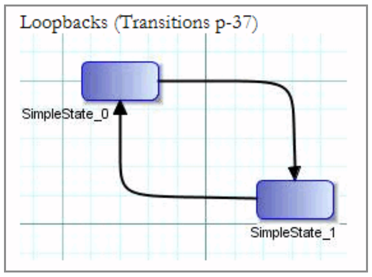
Self Transitions

Typically, transitions take an instance from one state to another state. Self transitions, however, connect a state to itself. Each time the instance loops back to the state, it triggers the entry action, and each time it leaves the state, it triggers the exit action.

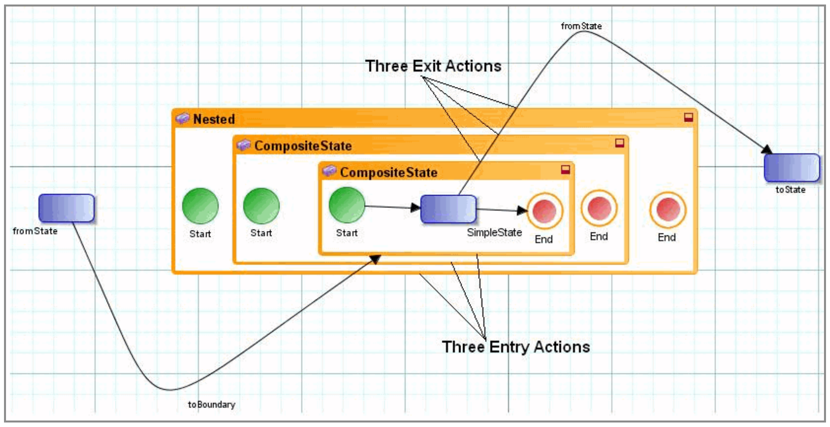
Start and end states cannot have self transitions.

Loopbacks

Loopbacks are allowed between any types of states except between start and end states.

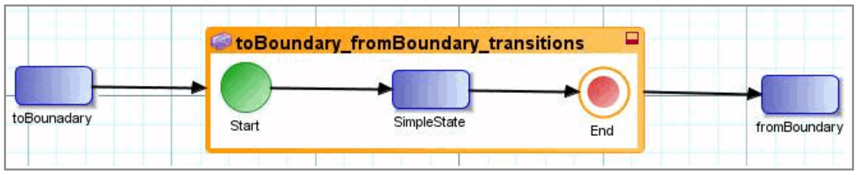


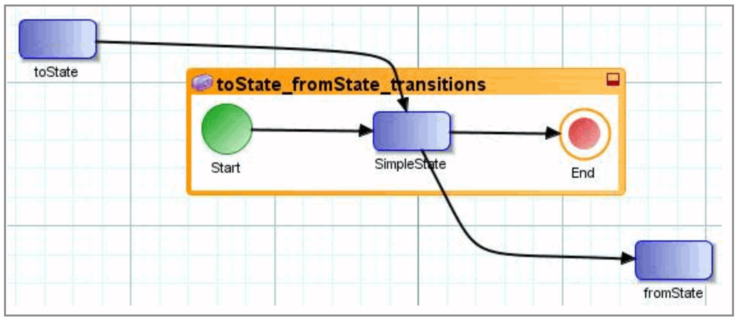
State Transitions



You can use state transitions (as opposed to Boundary transitions) for simple and for composite states. State transitions are not available for concurrent states.

For example:





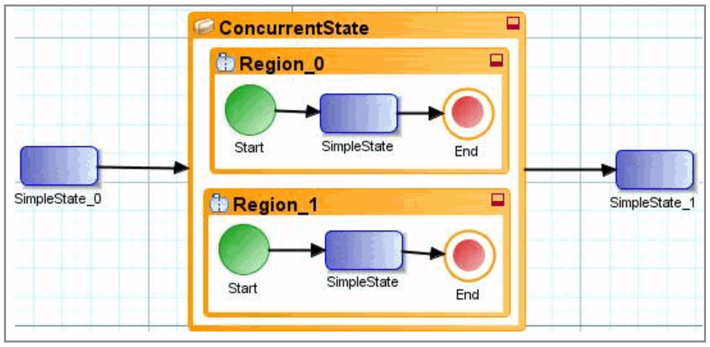
Transitions that Cross Multiple Boundaries

When a transition crosses the boundary of one or more nested states, the entry or exit actions of those states are executed.

Boundary Transitions

Composite states can use boundary transitions and Concurrent states must use them. Boundary transitions trigger entry and exit actions of the state whose boundary they are attached to.

Below is an example of boundary transitions used for a concurrent state.



Call State Machine Resource

To call a state machine from another state machine, at design time you add a Call State Model node. You insert the node like any other state in a state model and link it to other states using transitions.

When configuring a Call State Model node, you can specify any state model that is owned by the same concept as the main state model, or that is owned by any concept higher in that concept's inheritance chain (but not lower in the chain).

Note: At runtime, the state machine of a concept cannot call the state machine of a concept that is lower in the inheritance chain, and you cannot call a state machine recursively. That is, you cannot call a state machine from within itself either directly or indirectly.

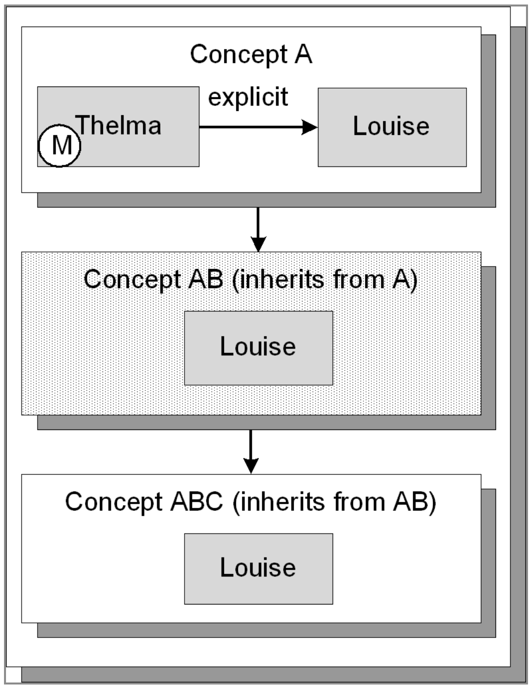
The concepts in an inheritance chain can own state models that have the same name. You can define the call as an explicit or virtual call. The **Call Explicitly** checkbox determines which of these same-named state models TIBCO BusinessEvents calls at runtime.

- **Explicit Call**: (Call Explicitly box checked.) TIBCO BusinessEvents makes an explicit call—it calls the state model specified (and at the level of the hierarchy specified.
- **Virtual Call**: (Call Explicitly box unchecked.) TIBCO BusinessEvents makes a virtual call—it calls the state model specified at the level of the asserted concept instance. If none is found at that level, then TIBCO BusinessEvents searches upwards in the concept hierarchy until it finds a state model of the specified name and it runs the first one it finds. This call is similar to the behavior of a Java or C++ virtual function.

You can use a Call State Model node even if you do not have concept inheritance relationship. In case of single concept, it does not matter if you check or uncheck the Call Explicitly checkbox.

Example of Explicit Call

In the following figure, Concept AB represents a concept instance that has been asserted at runtime. It inherits from Concept A. Concept ABC is another concept, lower in the inheritance chain than concept AB.



Concept A defines Thelma, a main state model and Louise, a state model that is *explicitly* called from the Thelma main state machine.

Note that Concept AB has no main state machine. It inherits the main state machine of Concept A.

At runtime, TIBCO BusinessEvents starts the Thelma main state machine for Concept AB. The Thelma state machine explicitly calls the Louise state machine defined for Concept A, and so TIBCO BusinessEvents starts the Louise state machine that is defined for Concept A.

If the call to Louise had not been defined as an explicit call, TIBCO BusinessEvents would have started the Louise state machine that was defined for Concept AB, because Concept AB is the concept that was asserted, and because it has a state machine of the same name as the one specified in the Thelma state machine.

Under no circumstances, however, would TIBCO BusinessEvents create an instance of `ABC.Louise` because it is below the concept instance in the inheritance chain.

State Model Timeouts and State Timeouts

You can define timeouts at the state model level and for each of its states, except for the start, end, and Call State Model nodes.

For the state machine, the timeout value specifies how long the state machine should wait before it completes. A timeout has an expression and an action. Both the expression and the action are defined using the TIBCO BusinessEvents rules language. See Defining Timeouts for complete details.

Note: TIBCO BusinessEvents supports only positive integers or zero as timeout values. It does not support negative numbers or decimal values.

The Timeout Period

A timeout expression specifies how long the state machine (or state) waits before it completes.

The timeout expression must evaluate to a number of time units (the units are defined in the timeout settings). If the time elapses before the state machine leaves the state, the state times out and TIBCO BusinessEvents executes the timeout action.

You can define the timeout period using different units in each state in a state model.

State Timeout Scheduler Controls

The `refreshAhead` and `pollInterval` properties controls the internal scheduler that manages state and state model timeouts.

The default value of the properties is 10000 ms. If you do not use default values, provide the same value for both properties. If they are set to different values, TIBCO BusinessEvents uses the higher value.

```
be.engine.cluster.smtimeout.refreshAhead
```

Time in milliseconds (into the future) used to pre-load the scheduled items from the backing store.

```
be.engine.cluster.smtimeout.pollInterval
```

TIBCO BusinessEvents checks the scheduler cache every *`pollInterval`* milliseconds, for scheduler work items whose timeout period expires in the current interval.

Runtime Behavior

At runtime, a state machine timeout timer starts when the state machine starts. The timer is cancelled when the state machine completes.

The state machine times out irrespective of its state at run-time when the state machine does not complete within the timeout period.

Similarly, a state timeout timer starts when the state machine enters the state. The timer is canceled when the state machine leaves the state, using any transition, including a self transition (in which the state machine exits and then re-enters the state).

The state times out when the state machine does not leave the state within the state timeout period.

Note: A timeout expression that evaluates to zero (For example: `return 0;`) means that the state model or state never times out.

Multiple Pending Timeouts

A state machine can have multiple pending timeouts. When the state machine exits an enclosing state, it exits all the enclosed states too. So if the result of a composite state timeout is to exit the state, then the state machine cancels all the enclosed timeouts (if any).

The Timeout Action

If a state model or state times out, TIBCO BusinessEvents executes the timeout action, defined using the TIBCO BusinessEvents rule language.

Once the state times out after the timeout expression, its timeout action gets executed, and then the timeout choices get executed. These choices are `current`, `all`, or `specified`, and are defined at design-time.

For the `specified` choice, you must also define what state to go to next in the event of a timeout. These additional options are explained in State Model Timeouts and State Timeouts.

State Timeout - Next State Choices

You can configure the Timeout Action setting for a state, which determines what state to go to next in the event of a state timeout.

The choices for the next state are:

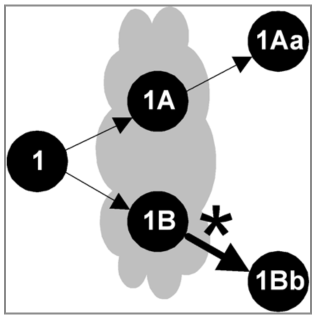
- **Current**: Remain in the current state. The timeout action executes. The timeout recurs. Entry and exit actions are ignored. However, the timeout period resets and it is possible for the state keep timing out.
- **Specified**: Go to the state specified in the Timeout State field. You can specify any top-level end state and any state that has the same parent as the state you are configuring.
- **All**: If the state has multiple possible next states, you can select this option to have TIBCO BusinessEvents prepare to go to any of the possible next states, that is, any state that has an incoming transition from the current state. (Note that one possible next state is the same state, if the original state transitions to itself.) See Timeout State Choice for States with Multiple Next States for details.

Timeout State Choice for States with Multiple Next States

When you choose the All timeout state choice, the state machine places the concept in a “pseudo state” consisting of all the possible next states, to indicate that it is not in the original state. However, the state machine does not execute any entry actions for these next states.

The next state is determined when something happens to determine which of the possible next states should execute.

For example, suppose a state 1 times out to "All." Its possible next states are 1A and 1B. 1A transitions to 1Aa and 1B transitions to 1Bb. The state machine waits until an outgoing transition from either 1A or 1B is triggered. Suppose the outgoing transition from 1B is triggered, then the state changes to 1Bb.



Note that in this case, none of the transition rules for the original state that timed out execute. Instead only the transition from 1B to 1Bb executes. In effect the state machine skips a state.

Locking (and Loading) in Timeouts

When TIBCO BusinessEvents uses Cache OM in a multi-agent or concurrent RTC application, locking must be used to ensure data integrity. When cache-only cache mode is used, you must also load concepts as needed.

For state transitions, locking and loading is handled in the normal way: the event that triggers a transition has an event preprocessor, where the locking is set to protect the concept from being modified by another action.

In multi-agent application, one inference agent at a time processes the timeout event of state machines. Once the current agent is stopped, next agent in the cluster resumes the processing.

To validate/confirm if an agent handles the state timeouts, enable JMX using the CDD property `be.engine.jmx.connector.port` in the ProcessUnit tab, open JVisualVM or JConsole and connect to the agent. Check for entry schedulers in the MBean com.tibco.be. The entry is available only for the InferenceAgent which handles the StateTimeouts.

Timeouts, however, are essentially time events, and time events do not go through an event preprocessor.

For this special case, therefore, you must implement locking (and loading as required) in a different way. Use the function `Cluster.registerStateMachineTimeoutCallback()` to register a rule function that acts as a preprocessor for state machine (and state) timeouts. You would typically call `Cluster.registerStateMachineTimeoutCallback()` in a startup function.

The signature is as follows:

```
registerStateMachineTimeoutCallback(String entityURI, String
ruleFunctionURI)
```

Where the `entityURI` is the URI of the state machine concept, and the `ruleFunctionURI` is the URI of the rule function that acts as a preprocessor.

See *TIBCO BusinessEvents Architect Guide* for details on this topic.

Working with State Modeler

A state model consists of nodes (states and Call State Model nodes) and transitions.

You can search for nodes or transitions using a specialized search feature. See Searching State Model Entities.

Nodes

A state model begins with one *start state* and ends with one or more *end states*. Between the start and end states you can add any number of other nodes: *simple*, *composite*, and *concurrent* states. Each state can have optional entry and exit actions.

Transitions

You connect the nodes with transitions, which can optionally depend on rules. You can define a transition that has no rule. This is called a lambda transition. If you configure a rule for a transition, the transition does not occur until the rule executes successfully.

Functions

The State Modeler includes its own set of functions, which are located in the Standard function registry under **Instance > StateMachine**. See Functions in *TIBCO BusinessEvents Architect Guide* and *TIBCO BusinessEvents Developer Guide* (and tooltips provided in TIBCO BusinessEvents Studio).

Note: Like all concepts, concepts with state machines aren't automatically deleted when their main state machines reach the end state. You must explicitly delete concept instances in your rules, using `Instance.deleteInstance()`.

Adding State Models

You can add complex states that allow you to nest nodes and transitions within them. You can also add *Call State Model* nodes, which allow you to call other state models.

For a guide to the settings see State Model Resource Reference.

Procedure

1. Do one of the following:
    - Right-click the folder where you want to store the state model and select **New > State Model**. You see the New State Model Wizard. In the **Owner Concept** field, specify the concept for this state model.
    - Right-click the concept that is to be the owner of this state model. You see the New State Model Wizard. The **Owner Concept** field contains the path to the concept you right-clicked.

      You can change the owner of a state model after it is created.
2. In the **File Name** field, type a name for the state model. In the **Description** field, type a description (optional).

    Note: You cannot change a new resource name in the editor after you click **Finish**. However, you can change the description. You can use a refactor operation to change the name later. (Right-click on the name in the explorer panel, and select **Refactor > Rename**.)
3. Click **Finish**. You see the state model editor, a diagram showing a start and an end state.
4. In the **Owner Concept** field, you can optionally change the owner concept for this state model. Click **Browse** to select a concept.
5. Check the **Main** checkbox if this is a main state model. Clear the **Main** checkbox if this is not a main state model.

Result

What you do next depends on the nature of the state model you want to define, how you prefer to work, and other actions you may want to do relating to the state model. See Outlining a State Model for suggestions.

The Studio Explorer view shows the state model in the folder where you add it. Note that, as a helpful reminder, the state model also appears as a child of the concept that owns it, but shortcut menu options do not appear here.

Removing and Changing State Model Ownership

A state model must be owned by a concept in order to execute. Depending on your ontology and rule logic, it may be possible to change the owner of a state model to another concept.

Removing a State Model from Ownership of a Concept

When you remove a state model from a concept, it becomes an orphan. You have the option to delete it entirely.

Procedure

1. Open the owning concept’s editor.
2. In the **Properties** view, select the state model from the State Models list and click **Remove.**
3. Click **Save**.

Reassigning a State Model to a Different Concept

Assign a orphan state model to a different concept owner, if you want to execute the state model.

Procedure

1. Open the state model’s editor.

2. In the **Properties** view, under the **General** tab, and in the **Owner Concept** field, browse to and select the desired concept.

   If the text in the **Owner Concept** field is red, this indicates that the ownership is not established, for example because the state model was removed from the concept, or because the concept was deleted.

   All occurrences of the previous owner concept in the **Declaration** field are replaced by the new owner concept. Entries in the State Models lists of the two respective concepts are adjusted automatically.

3. Edit the state model rules as needed, to ensure that they will work correctly with the new owner concept.

Outlining a State Model

Build a outline for the state model by adding the nodes you need and connecting them with transitions. Then you can fill in the details by configuring each node and transition.

Procedure

1. In TIBCO BusinessEvents Studio Explorer, double-click the state model name to open it.

   For instructions on adding a state model, see Adding State Models.

   Add Nodes

2. In the state modeler palette, select a type of state you want to add. Click in the canvas one or more times as needed to drop instances of the state onto the canvas or onto other states.

   Note: The selected type of state in the palette remains only if the preference for reset tool is unchecked.

   You can drop simple states, composite states, concurrent states and Call State Model nodes as follows:

   - Onto the canvas
   - Into a composite state
   - Into a region of a concurrent state

     You can drop regions only into concurrent states. (If you drop a region outside of a concurrent state, a blank virtual node appears. Delete this node.)

   Connect Nodes with Transitions

3. In the state modeler palette, select the transition icon and connect the states and state boundaries as needed to define the desired flow.

   - To add a transition from a state or boundary to a state or boundary, click the "from" state or boundary first, then the "to" state or boundary.
   - To add a self-transition, click the state or boundary, then click in an empty canvas area, then click the state or boundary again.

     See Guidelines for Adding Transitions for details about what you can and cannot connect.

4. As needed, cut, copy and delete states and transitions to complete the outline. In order to Cut, Copy, or Delete a node or transition perform the following steps:

   a. Right-click the node or transition and choose Cut, Copy, or Delete from the options menu.

   b. Right-click again and select Paste to paste a node or transition you cut or copied.

   Note:
   - A concurrent state must contain a minimum of two regions. You cannot cut or delete a region from a concurrent state with only two regions.
   - You cannot cut, copy, or delete a start state.
   - You cannot copy a state from one state model to a different state model.

   Configure the Details

5. When you are finished adding nodes and connecting them with transitions, click the Select arrow icon in the toolbar (or right-click in an empty part of the canvas). Select each node and transition in turn and configure them. See the following for details:

   - Configuring Nodes
   - Configuring Transitions

   Refine the Layout

6. Use the diagram options as desired to lay out the state model diagram as desired for viewing and sharing. See State Model Preferences.

Guidelines for Adding Transitions

You can connect one state to more than one other state, and you can connect to one state from more than one other state.

Note:

- When connecting from one state to multiple other states, only one transition will be taken when exiting from the state. (Use a concurrent state if you want concurrent behavior).
- TIBCO BusinessEvents Data Modeling does not allow transition between any state and its parent state.

You can connect to or from the following:

- Simple states
- Called state models (Call State Model nodes)
- Boundaries of concurrent states (but not to states or boundaries within them)
- Boundaries of composite states
- States and boundaries that lie *within* composite states (except states that are in a nested concurrent state)

  Note: A link can cross boundaries to connect nested states. When a link crosses a boundary, the entry or exit actions for that boundary execute.

  When you cross one or more boundaries to connect to or from a nested state or boundary, all entry or exit actions belonging to all crossed boundaries execute.

State Model Preferences

The State Model preferences are available from **Windows > Preferences > TIBCO BusinessEvents > Diagram > State Model**.

Reference to State Model Preferences

Option	Description
Fix node and edge labels	If enabled, displays the labels for nodes and edges at fixed positions. The users cannot move them manually. Default: Enabled.

Option	Description
Link Types	Shows the links as straight lines or curved lines. Default: Curved.
Grid	Displays the grid as lines or points, or does not display grid at all. Default: Lines.
Snap to grid	If you move nodes in a diagram and the grid is shown, the nodes snap to the grid lines if Snap to grid is enabled. Default: Disabled.
Layout Quality	Displays the layout in Draft, Medium or Proof quality. Default: Draft.
Layout Style	Defines if the layout will be Orthogonal or Hierarchical. Default: Orthogonal.
Orientation	Defines the general direction of the layout. Options are: Top to Bottom, Bottom to Top, Left to Right, and Right to Left. Default: Top to Bottom
Orthogonal Fix Node Sizes	This setting affects orientation behavior for the hierarchical layout option only. If enabled, node sizes do not change when you use the orientation feature. If disabled, orientation changes node sizes as needed for clarity. Default: Disabled.
Undirected Layout	Sets no orientation for the diagram. Default: Disabled.

Option	Description
Link Routing: Fix Node Sizes	This setting affects link routing behavior for all layout options except hierarchical layouts. If enabled, node sizes do not change when you use the link routing feature. If disabled, link routing changes node sizes as needed for clarity. Default: Disabled.
Link Routing: Fix Node Positions	This setting affects link routing behavior for all layout options. If enabled, node positions do not change when you use the link routing feature. If disabled, link routing changes node positions as needed for clarity. Default: Disabled.
Orthogonal or Polyline	Defines if the routing will be orthogonal or polyline (normal)
Link Routing:Orthogonal Fix Node Sizes	This setting affects link routing behavior for the hierarchical layout option only. If enabled, node sizes do not change when you use the link routing feature. If disabled, link routing changes node sizes as needed for clarity. Default: Disabled.

Configuring Nodes

Configure the state model and individual state properties to define action for each state or call for the state model.

It is assumed that nodes and transitions are laid out on the canvas as explained in Outlining a State Model.

Configuring a State

All state configurations are done using tabs in the Properties view for the state.

Procedure

1. In the diagram, select the state you want to configure.
2. In the **Properties** view, under the **General** tab, add a name and description as desired.
3. Depending on the type of state also configure the following:
   - Start states and simple states can have an exit action; end states and simple states can have an entry action. In the **Entry Action** or **Exit Action** tab, as appropriate, use the TIBCO BusinessEvents rule language to define any actions that will be performed when the state machine enters or leaves the state.
   - All states except start, end, and Call State Model nodes can have a timeout. See Defining Timeouts for background details about timeouts for state models and individual states.

     See State Resource Reference for details on all state configuration options.
4. In the **Properties** View, under the **Extended** tab, add (or remove) properties or groups of properties.
5. Click **Save**.

Configuring a Call State Model Node

The Call State Model resource allows you to call any state machine that is at the same level or higher in the inheritance chain.

See Call State Model Resource Reference for more details.

Procedure

1. In the diagram, select the state you want to configure.
2. In the **Properties** view, under the **General** tab, add a name and description as desired.

3. Select the **Call Explicitly** checkbox to make an explicit call. Clear the checkbox to make a virtual call. See Call State Model Resource Reference for an explanation of this setting.
4. In the **State Model Name** field, browse to and select the state model that is called at runtime.

Configuring Transitions

All transition configuration is done using tabs in the Properties view for the transition.

It assumes that nodes and transitions are laid out on the canvas as explained in Outlining a State Model.

Tip: A red arrow indicates a transition with a rule that has errors.

Procedure

1. In the diagram, select the transition you want to configure.
2. In the **Properties** view select the **General** tab, add a name and description as desired.
3. Do one of the following:
   - If the transition occurs automatically, select the **No Condition** checkbox. Configuration of this transition is complete.
   - If the transition occurs only when its associated rule executes, clear the **No Condition** checkbox and continue to the next step.
4. (If you did not check the No Condition checkbox.) Set a rule priority as desired.

   At runtime, when this transition rule is added to the rule agenda for a conflict resolution cycle, the priority determines its place in the order of execution.

5. Select the **Rule** tab and define the rule for this transition. Do any of the following as appropriate:
    - If the rule requires multiple entities to be present in working memory before it executes, add the entity or entities to the declaration section.
    - Define the conditions (if any) in the **Conditions** section.
    - In the **Actions** section define the actions (if any) that will occur when the rule executes.

Defining Timeouts

You can define timeouts for the overall state model and each of its states.

See State Model Timeouts and State Timeouts for an explanation of timeouts.

Defining Timeout for a State Model

Define a simple number of time units or more complex expression as timeout for the State Model in the Properties view.

Procedure

1. In the TIBCO BusinessEvents Studio Explorer, double-click the state model you want to work with.
2. In the **General** tab Units field, specify the time units for the state model timeout settings used in the timeout expression.
3. In the **Timeout Expression** tab, use the rule editor to enter an expression that defines the timeout period. The expression must resolve to a number of time units.
    - The default expression (`return 0;`) specifies that there is no timeout.
    - To define a specific timeout period, enter a positive value in the return statement, instead of 0.
    - You can enter a more complex expression as needed, using the TIBCO BusinessEvents rule language.

4. In the **Timeout Action** tab, use the rule editor to specify what is to happen if a timeout occurs, again using the rule language. For example, an event may go out to notify an employee of a problem, or to restock inventory and so on.

Defining Timeout for a State

Define the resulting state of a timeout and the actual timeout period using regular or complex expressions.

Procedure

1. In the TIBCO BusinessEvents Studio Explorer, double-click the state model you want to work with and in the diagram, click the state whose timeout you want to configure.
2. Select the **Timeout** tab and select the units to use: Milliseconds, Seconds, Minutes, Hours, Days, Weekdays, Weeks, Months. (This means you do not have to define units in the timeout expression.)
3. Select a Timeout State Choice value, to determine what state the machine goes to in the event of a timeout:
    - **Current**: Stays in the same state. The timeout action executes. Entry and exit actions are ignored. However, the timeout period resets and it is possible for the state keep timing out.
    - **All**: for states with multiple next states only, you can select this option to have TIBCO BusinessEvents prepare to go to any of the possible next states. For more details on this option, see State Model Timeouts and State Timeouts.
    - **Specified**: In the Timeout State field, select the state to go to in the event of a timeout.

    Note: When you choose All or Current, the value of the Timeout State field is grayed-out and is ignored. That field is used only for the Specified option.
4. In the **Timeout Expression** tab, enter an expression that evaluates to a number (that is, a number of the time units you chose in the General tab).
5. In the **Timeout Action** tab, use the TIBCO BusinessEvents rule language to add any actions that will be performed when the state times out.

Controlling the Start of a State Machine

By default, main state machines start when their associated concept instance is asserted. You can also configure TIBCO BusinessEvents not to start a concept's main state machine immediately after the concept is asserted.

Choose this mode by unchecking the concept’s Auto Start State Model checkbox. Note that this setting applies to concepts, not state machines, even ones that are inherited.

See *TIBCO BusinessEvents Developer Guide* for more information on concept and rule configuration settings.

Procedure

1. Open the editor for the state model’s owner concept.
2. In the Configuration section, clear the **Auto Start State Model** checkbox.
3. Open the rule editor for all appropriate rules and rule functions, depending on your project design, and add the following standard function:

   ```
   Instance.startStateMachine()
   ```

   This function takes two parameters: the concept name (specified using its project folder path), and a Boolean. The effect of the Boolean is as follows:

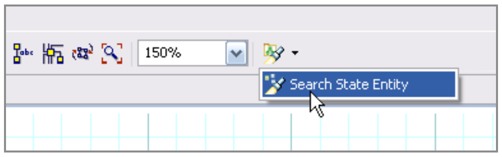
   - If set to true, then contained concepts’ main state machines are also started, if not already started (or if the contained concepts’ main state machines also have the Auto Start State Machine checkbox checked).
   - If set to false, then only the main state machine of the concept passed to the function is started.

Searching State Model Entities

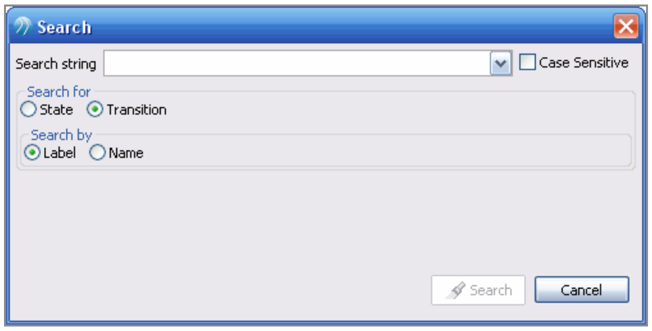
State models can become quite complex. You can search for specific states or transitions using the State Modeler Search Entity feature. It enables you to search either state nodes or transition edges in any one search.

Procedure

1. Open the state model in the editor. In the toolbar, click the drop-down list beside the **Search Diagram Entities** button, and select **Search State Entity**:

   You see the following search dialog:

2. Type a search string.

   Search text persists in the drop-down list until you dismiss the dialog.

3. Select how and where to search as desired:

   - To make the search case sensitive, select the **Case Sensitive** checkbox.
   - To search for a state, click the **State** button.
   - To search for a state transition, click the **Transition** button and then click the **Label** or **Name** button to select whether to search by label or by name.

4. Click **Search**.

   If a match is found, the matching state or transition is selected and the diagram centers on it.

State Model Resource Reference

State model resources are used to model the life cycle of a concept instance. Within a state model resource you configure states and transitions and calls to other state models.

General Tab

The General tab provides the generic information about the resource, such as, name, description, and so on.

Field	Global Var?	Description
Name	No	The name to appear as the label for the resource. Names follow Java variable naming restrictions. Do not use any reserved words. Names must be unique within a folder.
Description	No	Short description of the resource.
Owner Concept	No	The concept that owns this state model. Only one concept can own a state model. The owner concept can be changed as desired. See Removing and Changing State Model Ownership
Main	No	Select the checkbox if this is the main state model for the owner concept. The owner concept has a corresponding field, Main State Model. To associate this concept with a state machine, browse to and select the main state machine.
Units	No	Time units to use for the timeout expression. Possible values are: Milliseconds, Seconds, Minutes, Hours, Days, Weekdays, Weeks, Months.

Timeout Expression Tab

The Timeout Expression tab has a declaration section which references the owner concept. You cannot add anything to the declaration. The concept alias cannot be changed.

In the Timeout Expression tab Actions section, define the length of the timeout by entering an expression that evaluates to a number (that is, a number of the time units you chose in the General tab).

Timeout Action Tab

The Timeout Action tab has a declaration section which references the owner concept. You cannot add anything to the declaration.

In the Timeout Action tab, use the TIBCO BusinessEvents rule language to add any actions that will be performed when the state machine times out.

State Resource Reference

Simple, composite, and concurrent states, as well as regions within concurrent states have the full set of tabs. Start and end states, and Call State Model resources have a subset as noted.

See Types of States for a general description of each type of state: start and end, simple, composite, concurrent.

General Tab

All types of states have a General tab which provides a name and short description about the resource.

Field	**Global Var?**	**Description**
Name	No	The name to appear as the label for the resource. Names follow Java variable naming restrictions. Do not use any reserved words. Names must be unique within a folder.
Description	No	Short description of the resource.

Entry Action and Exit Action Tabs

Simple states, composite states, concurrent states, and regions within a concurrent state, can each have an entry and exit action.

A start state can have an exit action. An end state can have an entry action. A state's entry action executes when the state model transitions into that state; its exit action executes when the state model transitions out of the state.

Entry and exit actions execute even in the case of self transitions. Entry and exit actions have a declaration section which references the owner concept. You cannot add anything to the declaration. Entry and exit actions also have an Actions section, in which you use the TIBCO BusinessEvents rule language to define the action.

Timeout Tab

Simple states, composite states, concurrent states, and regions within a concurrent state, can each have a timeout, timeout expression and timeout action.

The Timeout tab has the following fields.

Field	Global Var?	Description
Units	No	The time unit used in the Timeout Expression tab. Possible values are: Milliseconds, Seconds, Minutes, Hours, Days, Weekdays, Weeks, Months.
Timeout State Choice	No	In the event of a timeout, this setting determines the next state: • **Current**: Stays in the same state. The timeout action executes. Entry and exit actions are ignored. However, the timeout period resets and it is possible for the state keep timing out.

Field	Global Var?	Description
		• **All**: for states with multiple next states only, you can select this option to have TIBCO BusinessEvents prepare to go to any of the possible next states. For more details on this option, see State Model Timeouts and State Timeouts. • **Specified**: In the Timeout State field, select the state to go to in the event of a timeout.
Timeout State	No	If the Timeout State Choice is Specified, then select the state to transition to in this field. In the event of a timeout, the state machine transitions to the specified state.

Timeout Expression Tab

Simple states, composite states, concurrent states, and regions within a concurrent state, can each have a timeout, timeout expression and timeout action.

The Timeout Expression tab has a declaration section which references the owner concept. You cannot add anything to the declaration.

In the Timeout Expression tab Actions section, you define the length of the timeout by entering an expression that evaluates to a number (that is, a number of the time units you chose in the Timeout tab).

Timeout Action Tab

The Timeout Action tab has a declaration section which references the owner concept. You cannot add anything to the declaration.

In the Timeout Action tab, use the TIBCO BusinessEvents rule language to add any actions that will be performed when the state times out.

Call State Model Resource Reference

The Call State Model resource allows you to call any state machine that is at the same level or higher in the inheritance chain.

Note: The state machine of a concept cannot call the state machine of a concept that is lower in the inheritance chain, and you cannot call a state machine recursively. That is, you cannot call a state machine from within itself either directly or indirectly.

Field	Global Var?	Description
Name	No	The name to appear as the label for the resource. Names follow Java variable naming restrictions. Do not use any reserved words. Names must be unique within a folder.
Description	No	Short description of the resource.
Call Explicitly	No	If checked, the call is an explicit call. If unchecked, the call is a virtual call. At design time, you may have concepts in the same inheritance chain that include same-named state machines. The **Call Explicitly** checkbox allows you to select the state machine you need. With the **Call Explicitly** check box unchecked, the call is similar to a Java or C++ virtual function call. See Call State Machine Resource for a more detailed explanation.
State Model Name		The name of the state model that is called at runtime.

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

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

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

Product-Specific Documentation

The documentation for this product is available on the TIBCO BusinessEvents® Enterprise Edition Documentation page.

To directly access documentation for this product, double-click the file at the following location:

TIBCO_HOME`/release_notes/TIB_businessevents-enterprise_6.3.1_docinfo.html`

where *TIBCO_HOME* is the top-level directory in which TIBCO products are installed. On Windows, the default *TIBCO_HOME* is `C:\tibco`. On UNIX systems, the default *TIBCO_HOME* is `/opt/tibco`.

Other TIBCO Product Documentation

When working with TIBCO BusinessEvents Enterprise Edition, you may find it useful to read the documentation of the following TIBCO products:

- TIBCO ActiveSpaces®: It is used as the cluster, cache, or store provider for the TIBCO BusinessEvents Enterprise Edition project.
- TIBCO FTL®: It is used as the cluster provider for the TIBCO BusinessEvents Enterprise Edition project.

How to Access Related Third-Party Documentation

When working with TIBCO BusinessEvents® Enterprise Edition, you may find it useful to read the documentation of the following third-party products:

- Apache Ignite
- Apache Kafka
- Confluent Kafka Schema Registry
- TIBCO Messaging - Schema Repository for Apache Kafka
- Apache Pulsar
- GridGain
- Apache Cassandra
- Grafana
- InfluxDB
- OpenTelemetry
- Control Plane
- Apache Maven

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the product Support website. If you do not have a username, 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

Cloud SG 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” file for the availability of a specific version of Cloud SG software on a specific operating system platform.