

TIBCO BusinessEvents™

Architect's Guide

*Software Release 4.0
May 2010*

The Power to Predict™



Important Information

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

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

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

TIB, TIBCO, TIBCO Software, TIBCO Adapter, Predictive Business, Information Bus, The Power of Now, The Power to Predict, TIBCO BusinessEvents, TIBCO ActiveSpaces, TIBCO ActiveMatrix BusinessWorks, TIBCO Rendezvous, TIBCO Enterprise Message Service, TIBCO PortalBuilder, TIBCO Administrator, TIBCO Runtime Agent, TIBCO General Interface, and TIBCO Hawk are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

EJB, Java EE, J2EE, JMS and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Excerpts from Oracle Coherence documentation are included with permission from Oracle and/or its affiliates. Copyright © 2000, 2006 Oracle and/or its affiliates. All rights reserved.

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

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README.TXT FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

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

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

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

This product is covered by U.S. Patent No. 7,472,101.

Copyright © 2004-2010 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

Contents

Figures	vii
Tables	ix
Preface	xi
Related Documentation	xii
TIBCO BusinessEvents Documentation	xii
TIBCO BusinessEvents Event Stream Processing	xiii
TIBCO BusinessEvents Decision Manager	xiii
TIBCO BusinessEvents Data Modeling	xiii
TIBCO BusinessEvents Views	xiv
Other TIBCO Product Documentation	xiv
Typographical Conventions	xv
How to Contact TIBCO Support	xviii
Chapter 1 Overview	1
What's Different About Complex Event Processing	2
Technical Requirements of a CEP System	3
A Model-Driven Approach	3
Stateful Rule Engine	5
Object Management Options	5
Main Product Components	6
Design-time Components	6
Administration Components	7
Design-time Resource Overview	9
Channels and Events	9
Concepts	10
Score Cards	10
Rules	11
Object Management and Fault Tolerance	11
State Modeler	12
Database Concepts	12
Query Language	12
Pattern Language	13

Chapter 2 Channels and Events	15
Channels and Events Overview	16
Event Preprocessors	17
Preprocessor Use Guidelines	17
Types of Channels	19
Default Destinations and Default Events	20
Message Acknowledgment	21
Types of Events	22
Simple Events	22
Time Events	23
Advisory Events	23
Simple Events — Time to Live and Expiry Actions	24
Event Expiration and Expiry Actions	25
Chapter 3 Concepts	27
Overview of Concepts	28
Concept Property History	30
History Size	30
History Policy	32
Concept Relationships	34
Inheritance Relationships	34
Containment Relationships	35
Reference Relationships	36
Rules Governing Containment and Reference Relationships	37
When a Contained or Referred Concept Instance is Deleted	39
Chapter 4 Rules and Functions	41
Rules	42
Inferencing Rules	42
Rule Priority and Rank	42
Organizing and Deploying Inferencing Rules	43
Rule Functions	44
Virtual Rule Functions and Decision Tables	44
Startup and Shutdown Rule Functions	45
When Startup Rule Functions Execute	45
Creating Entities With a Startup Action in a Multi-Engine Project	46
ActiveMatrix BusinessWorks Containers	46
Chapter 5 Run-time Inferencing Behavior	47
Runtime Architecture and Flow	48

Rule Evaluation and Execution	49
Understanding Conflict Resolution and Run to Completion Cycles	51
How the Rete Network is Built	53
Testing the Truth of a Rule's Conditions Using the Dependency Set	53
How a Rule Becomes Newly True	54
Order of Evaluation of Rule Conditions	54
Chapter 6 Object Management Options	57
Object Management (OM) Overview	58
The Cache Object Manager	58
The In Memory Object Manager	59
The Berkeley DB (Persistence) Object Manager	60
Summary of Object Management Features	60
Migrating to a Different Object Management Method	61
Berkeley DB Object Manager	62
Fault Tolerance With Berkeley DB Manager	63
Object Management and Fault Tolerance Scenarios	64
Cache OM with Memory Only Mode on All Objects and Fault Tolerance Scenarios	64
Berkeley DB Object Management and Fault Tolerance Scenarios	65
Cache Object Management and Fault Tolerance Scenarios	66
Chapter 7 Distributed Cache OM	69
Cache Object Management Feature Overview	70
Distributed Cache Characteristics	70
Scaling the System	70
Reliability of Cache Object Management	71
Concurrency — Multi-Agent and Concurrent Rete Features	71
Where Object Management is Configured	72
Characteristics of Distributed Caching Schemes	73
Failover and Failback of Distributed Cache Data	74
Limited and Unlimited Cache Size	74
Distributed Cache and Multi-Agent Architecture and Terms	76
Cache Clusters	77
Cache Cluster Processing Units (Nodes)	77
Inference Agents	77
Cache Agents (Storage Nodes)	78
Query Agents	78
Dashboard Agents	79
Cache Cluster Discovery	80
Cluster Member Discovery Using Multicast Discovery	80
Cluster Member Discovery Using Well-Known-Addresses	81
Discovery When Host Machines Have Multiple Network Cards	81

Load Balancing and Fault Tolerance of Inference Agents	82
Load Balancing of Inference Agents in a Group	82
Fault Tolerance Between Inference Agents in a Group	82
Cache OM with a Backing Store	84
Backing Store Write Options — Cache-aside and Write-behind	84
Cache Manager Options at the Entity Level	86
Between Cache and Backing Store: Preloading Options and Limited Cache Size	86
Between Rete Network and Cache: Cache Modes	87
Chapter 8 Cache Modes and Project Design	89
Working With Cache Modes	90
Cache Plus Memory — For Constants and Less Changeable Objects	91
In Memory Only — Useful for Stateless Entities	91
Cache Only Mode	92
Loading Cache-Only Objects into the Rete Network	93
Cache Load Functions	93
Loaded Objects are Not New and Don't Trigger Rules to Fire	94
Chapter 9 Concurrency and Project Design	95
Designing for Concurrency	96
Multi-Agent Features and Constraints	97
Concepts are Shared Across Agents Asynchronously	97
Scorecards are Local to the Agent	97
Events are Owned by the Agent that Receives Them	98
Multi-Agent Example Showing Event Handling	99
Using Locks to Ensure Data Integrity Within and Across Agents	101
Understanding Locking in BusinessEvents	101
When to Use Locking	102
Lock Processing Example Flow	102
Locking Functions	104
Tips on Using Locks	105
Avoiding Deadlock when Multiple Keys Protect One Object	106
Diagnosing and Resolving Lock Failures	106
Chapter 10 Deploying, Monitoring and Managing	109
Deploy-time Configuration and Deployment	110
Cluster Deployment Descriptor (CDD)	110
Deployment Methods	111
Hot Deployment	111
Monitoring and Management	111
Index	113

Figures

Figure 1 Channels and Destinations20

Figure 2 History Ring Buffer31

Figure 3 History Policy32

Figure 4 TIBCO BusinessEvents Architecture49

Figure 5 Run to Completion Cycle51

Figure 6 Cache Object Management and Fault Tolerance Architecture76

Tables

Table 1	General Typographical Conventions	xv
Table 2	Syntax Typographical Conventions	xvi
Table 3	Containment and Reference Concept Relationship Rules	38
Table 4	Cache OM with Memory Only Mode on All Objects and Fault Tolerance Scenarios	64
Table 5	Persistence and Fault Tolerance Scenarios	65
Table 6	Cache and Fault Tolerance Scenarios	66

Preface

TIBCO BusinessEvents™ allows you to abstract and correlate meaningful business information from the events and data flowing through your information systems, and take appropriate actions using business rules. By detecting patterns within the real-time flow of events, BusinessEvents™ can help you to detect and understand unusual activities as well as recognize trends, problems, and opportunities. BusinessEvents publishes this business-critical information in real time to your critical enterprise systems or dashboards. With BusinessEvents you can predict the needs of your customers, make faster decisions, and take faster action.

BusinessEvents
The Power to Predict™

Topics

- [Related Documentation, page xii](#)
- [Typographical Conventions, page xv](#)
- [How to Contact TIBCO Support, page xviii](#)

Related Documentation

This section lists documentation resources you may find useful.

TIBCO BusinessEvents Documentation

- *TIBCO BusinessEvents Installation*: Read this manual for instructions on site preparation and installation.
- *TIBCO BusinessEvents Getting Started*: After the product is installed, use this manual to learn the basics of BusinessEvents. This guide provides step-by-step instructions to implement an example project and also explains the main ideas so you gain understanding as well as practical knowledge.
- *TIBCO BusinessEvents Architect's Guide*: If you are architecting an application using TIBCO BusinessEvents, read this guide for overview and detailed technical information to guide your work.
- *TIBCO BusinessEvents Developer's Guide*: After the architect has designed the system, use this manual to implement the design in BusinessEvents Studio.
- *TIBCO BusinessEvents Administration*: This book explains how to configure, deploy, monitor, and manage a BusinessEvents application and the data it generates.
- Online References:
 - *TIBCO BusinessEvents Cache Configuration Guide*: This online reference is available from the HTML documentation interface. It provides configuration details for cache-based object management. Cache-based object management is explained in *TIBCO BusinessEvents Administration*.
 - *TIBCO BusinessEvents Java API Reference*: This online reference is available from the HTML documentation interface. It provides the Javadoc-based documentation for the BusinessEvents API.
 - *TIBCO BusinessEvents Functions Reference*: This online reference is available from the HTML documentation interface. It provides a listing of all functions provided with BusinessEvents, showing the same details as the tooltips available in the BusinessEvents Studio rule editor interface.
- *TIBCO BusinessEvents Release Notes*: Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

TIBCO BusinessEvents Event Stream Processing

This BusinessEvents add-on is available separately, and includes the BusinessEvents Query Language features and the Pattern Matching Framework.

- *TIBCO BusinessEvents Event Stream Processing Installation*: Read this brief manual for installation instructions. A compatible version of TIBCO BusinessEvents must be installed first.
- *TIBCO BusinessEvents Event Stream Processing Add-on Query Developer's Guide*: This manual explains how to use the object query language to query various aspects of the running system.
- *TIBCO BusinessEvents Event Stream Processing Pattern Matcher Developer's Guide*: This manual explains how to use the pattern matcher language and engine to correlate event patterns in a running system.
- *TIBCO BusinessEvents Event Stream Processing Release Notes*: Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

TIBCO BusinessEvents Decision Manager

This BusinessEvents add-on is available separately. It incorporates a decision modeling business user interface, and associated runtime.

- *TIBCO BusinessEvents Decision Manager Installation*: Read this brief manual for installation instructions. A compatible version of TIBCO BusinessEvents must be installed first.
- *TIBCO BusinessEvents Decision Manager User's Guide*: This manual explains how business users can use decision tables and other decision artifacts to create business rules. It also covers configuration and administration of Rules Management Server, which is used for authentication, authorization, and approval processes.
- *TIBCO BusinessEvents Decision Manager Release Notes*: Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

TIBCO BusinessEvents Data Modeling

This BusinessEvents add-on is available separately. It contains state models and database concept features.

- *TIBCO BusinessEvents Data Modeling Installation*: Read this brief manual for installation instructions. A compatible version of TIBCO BusinessEvents must be installed first.

- *TIBCO BusinessEvents Data Modeling Developer's Guide*: This manual explains data modeling add-in features for BusinessEvents. The database concepts feature enables you to model BusinessEvents concepts on Database tables. The state modeler feature enables you to create state machines.
- *TIBCO BusinessEvents Data Modeling Release Notes*: Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

TIBCO BusinessEvents Views

This BusinessEvents add-on is available separately. It includes graphical dashboard components for run-time event monitoring.

- *TIBCO BusinessEvents Views Installation*: Read this manual for instructions on site preparation and installation.
- *TIBCO BusinessEvents Views Developer's Guide*: This book explains how to use BusinessEvents BusinessEvents Views to create meaningful metrics that are presented to business users in real-time for proactive decision making.
- *TIBCO BusinessEvents Views User's Guide*: This book explains how to monitor metrics in BusinessEvents BusinessEvents Views and how to represent the business processes graphically.
- *TIBCO BusinessEvents BusinessEvents Views Release Notes*: Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

Other TIBCO Product Documentation

You may find it useful to refer to the documentation for the following TIBCO products:

- TIBCO ActiveSpaces[®]
- TIBCO Hawk[®]
- TIBCO Rendezvous[®]
- TIBCO Enterprise Message Service[™]
- TIBCO ActiveMatrix BusinessWorks[™]

Typographical Conventions

The following typographical conventions are used in this manual.

Table 1 General Typographical Conventions

Convention	Use
<i>TIBCO_HOME</i> <i>ENV_HOME</i> <i>BE_HOME</i>	<p>Many TIBCO products must be installed within the same home directory. This directory is referenced in documentation as <i>TIBCO_HOME</i>. The value of <i>TIBCO_HOME</i> depends on the operating system. For example, on Windows systems, the default value is C:\tibco.</p> <p>Other TIBCO products are installed into an installation environment. Incompatible products and multiple instances of the same product are installed into different installation environments. The directory into which such products are installed is referenced in documentation as <i>ENV_HOME</i>. The value of <i>ENV_HOME</i> depends on the operating system. For example, on Windows systems the default value is C:\tibco.</p> <p>TIBCO BusinessEvents installs into a directory within <i>ENV_HOME</i>. This directory is referenced in documentation as <i>BE_HOME</i>. The value of <i>BE_HOME</i> depends on the operating system. For example on Windows systems, the default value is C:\tibco\TIBCO BusinessEvents\4.0.</p>
code font	<p>Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example:</p> <p>Use MyCommand to start the foo process.</p>
bold code font	<p>Bold code font is used in the following ways:</p> <ul style="list-style-type: none"> • In procedures, to indicate what a user types. For example: Type admin. • In large code samples, to indicate the parts of the sample that are of particular interest. • In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, MyCommand is enabled: MyCommand [enable disable]

Table 1 General Typographical Conventions (Cont'd)




Convention	Use
<i>italic font</i>	<p>Italic font is used in the following ways:</p> <ul style="list-style-type: none">• To indicate a document title. For example: See <i>TIBCO BusinessWorks Concepts</i>.• To introduce new terms For example: A portal page may contain several <i>portlets</i>. Portlets are mini-applications that run in a portal.• To indicate a variable in a command or code syntax that you must replace. For example: <code>MyCommand <i>pathname</i></code>
Key combinations	<p>Key name separated by a plus sign indicate keys pressed simultaneously. For example: <code>Ctrl+C</code>.</p> <p>Key names separated by a comma and space indicate keys pressed one after the other. For example: <code>Esc, Ctrl+Q</code>.</p>
	The note icon indicates information that is of special interest or importance, for example, an additional action required only in certain circumstances.
	The tip icon indicates an idea that could be useful, for example, a way to apply the information provided in the current section to achieve a specific result.
	The warning icon indicates the potential for a damaging situation, for example, data loss or corruption if certain steps are taken or not taken.

Table 2 Syntax Typographical Conventions

Convention	Use
[]	<p>An optional item in a command or code syntax.</p> <p>For example:</p> <pre>MyCommand [optional_parameter] required_parameter</pre>
	<p>A logical 'OR' that separates multiple items of which only one may be chosen.</p> <p>For example, you can select only one of the following parameters:</p> <pre>MyCommand param1 param2 param3</pre>

Table 2 Syntax Typographical Conventions

Convention	Use
{ }	<p>A logical group of items in a command. Other syntax notations may appear within each logical group.</p> <p>For example, the following command requires two parameters, which can be either the pair param1 and param2, or the pair param3 and param4.</p> <pre>MyCommand {param1 param2} {param3 param4}</pre> <p>In the next example, the command requires two parameters. The first parameter can be either param1 or param2 and the second can be either param3 or param4:</p> <pre>MyCommand {param1 param2} {param3 param4}</pre> <p>In the next example, the command can accept either two or three parameters. The first parameter must be param1. You can optionally include param2 as the second parameter. And the last parameter is either param3 or param4.</p> <pre>MyCommand param1 [param2] {param3 param4}</pre>

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, please contact TIBCO Support as follows.

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

<http://www.tibco.com/services/support>

- If you already have a valid maintenance or support contract, visit this site:

<https://support.tibco.com>

Entry to this site requires a user name and password. If you do not have a user name, you can request one.

Chapter 1 **Overview**

This chapter provides an overview of TIBCO BusinessEvents, including a brief introduction to complex event processing and event-driven applications, and a description of the major product components

Topics

- *[What's Different About Complex Event Processing, page 2](#)*
- *[Main Product Components, page 6](#)*
- *[Design-time Resource Overview, page 9](#)*

What's Different About Complex Event Processing

Complex Event Processing (CEP) is a set of technologies that allows "events" to be processed on a continuous basis.

Most conventional event processing software is used either for Business Process Management (BPM), TIBCO iProcess for example, or for Service Oriented Architecture (SOA), for example TIBCO ActiveMatrix BusinessWorks software.

CEP is unlike conventional event processing technologies, however, in that it treats all events as potentially significant and records them asynchronously. Applications that are appropriate for CEP are *event-driven*, which implies some aspect of real-time behavior. To be more specific, the typical CEP application area can be identified as having some aspect of "situation awareness," "sense and respond," or "track and trace," aspects which overlap in actual business situations.

Situation awareness is about "knowing" the state of the product, person, document, or entity of interest at any point in time. Achieving this knowledge requires continuous monitoring of events to do with the entity, events that indicate what situation or state the entity is in, or about to be in. As an example, a dashboard indicates all performance indicators for a runtime production process. All the production plant events are monitored and the situation, or health, of the production process is determined via some performance indicators that are shown in real-time to one or more operators.

Sense and respond is about detecting some significant fact about the product, person, document or entity of interest, and responding accordingly. To achieve this result the software does the following:

- Monitors events that indicate what is happening to this entity.
- Detects when something significant occurs.
- Executes the required response.

As an example, you may monitor cell phone or credit card usage, detect that a cell phone or credit card is being used consecutively at locations that are too far apart for real-time person-to-business transactions. Detection of such transactions indicates that an act of fraud is in progress. The system responds accordingly, denying the transactions, and invoking the necessary workflow to handle the situation as defined in standard procedures.

Track and trace is about tracking the product, person, document or entity of interest over time and tracing pertinent facts like location, owner, or general status. An example would be tracking events from an RFID-enabled inventory control system where at any point in time you need to know how many widgets are in what location.

"Situation awareness," "sense and respond," and "track and trace" can all be classified as types of *activity monitoring*, for which the continuous evaluation of incoming events is suitable. For this reason, CEP is often described as a generalization of Business Activity Monitoring (BAM), although the event processing task may be only indirectly be related to business, as in the case of an engine monitoring application or process routing task.

Technical Requirements of a CEP System

CEP systems must be able to receive and record events and identify patterns of these events and any related data. CEP systems must also handle temporal or time-based constraints, especially for handling the non-occurrence of events. The following TIBCO BusinessEvents features satisfy these requirements:

- A rich event model, incorporating event channels (for different event mechanisms) and destinations (for different types of events).
- A pattern detection mechanism using a sophisticated, high performance, declarative rule engine.
- A state model mechanism that allows entities to be described in terms of state, and in particular allows modelling of time-out events to handle the non-occurrence of events. (State modeling requires TIBCO BusinessEvents Data Modeling, purchased separately.)

A Model-Driven Approach

TIBCO BusinessEvents can be described not only as a CEP engine but also as an event-driven rule engine or real-time rule engine. TIBCO BusinessEvents enables CEP problems to be solved through a model-driven approach, in which the developer defines the event, rule, concept (class) and state models which are then compiled so that at run-time incoming events are processed as efficiently as possible. The various models are as follows:

Event model The event model describes the inputs into BusinessEvents. Events provide information through their properties and (optionally) through an XML payload. The event model provides the primary interface between BusinessEvents and the outside world, for input as well as output. Typical event sources (or channels) are messages from TIBCO Rendezvous and TIBCO Enterprise Message Service middleware, events generated explicitly by BusinessEvents, and custom mechanisms for non-standard event sources. Events can be used to trigger rules.

Concept model The concept model describes the data concepts used in BusinessEvents, which may be mapped from events or their payloads, or loaded by some other mechanism into BusinessEvents. The concept model is based on standard UML Class and Class Diagram principles.

Rule model Rules provide one of the main behavioral mechanisms in BusinessEvents. Rules are defined in terms of *declarations* (events and concepts of interest), *conditions* (filters and joins on and between the attributes of the events and concepts), and *actions*. The underlying rule engine is based on an algorithm called the Rete algorithm, which mixes all rules together into a type of look-up tree, so that any additional concept instance or event can near-instantly cause the appropriate rule or rules to fire and invoke the appropriate actions. Rules are almost always defined in general terms (concepts or classes and events), so they apply to however many combinations of those events and classes exist in memory at any one time. The combination of rule declaration and condition defines the *event pattern* required for CEP operation. Rule actions that update other concepts may cause other rules to become available for firing, a process called *inferencing* or *forward chaining*. These types of rules are generally called *Production Rules*. The appropriate UML Production Rule Representation is still under development.

Rule functions Algorithms, procedures or functions may be usefully defined as parameterized rule functions and re-used as required in rule actions and other areas where a behavior can be specified.

State model An important item of metadata for a concept or object is its state. Typically a state model describes the states that an entity can hold, and the transitions between states that are allowed, and the conditions for such transitions. Internally the state model is just additional metadata, but it is more useful to describe the state model as a visual model of states and transitions. The state transition rules can be viewed as special customizations of standard rules. The state model is based on the standard UML State Model principles. Requires TIBCO BusinessEvents Data Modeling add-on software.

Query model Queries can provide both snapshot and continuous views of the data in a BusinessEvents cache. Queries can also provide continuous views of data arriving through channels. They are constructed and executed from rule functions in a specialized agent (called a query agent). Queries provide event stream processing or set operations to derive information that can then be used in rule functions, or shared (via events or the cache). Requires TIBCO BusinessEvents Event Stream Processing add-on software.

Stateful Rule Engine

At run-time, the rule engine executes rules based on new events and data sources on a continuous basis. The rule memory is never "reset" (unless by design), so that future events can always be compared to past events. For this reason, the rule engine is described as a *stateful rule engine*. If required, the working memory can be cleared and a new set of data asserted for each "transaction," in which case the engine is operating as a *stateless rule engine*.

Object Management Options

To ensure resilience, BusinessEvents provides two persistence mechanisms for the events and data loaded into the system. One is called Persistence. It is a checkpoint mechanism that causes the data in the working memory to be saved to a lightweight database engine, to be restored when and as required. The other is a high performance distributed cache that allows data to be persisted and removed from the Rete network or returned to the Rete network, as required to handle extremely large problem domains (that would not typically fit into a runtime memory model). A backing store can be added to provide additional reliability and object management options. Just as data can be moved between the Rete network and the cache, so can less used data be moved between the backing store and the cache, to balance storage, memory, and performance requirements.

These characteristics provide BusinessEvents with its enterprise and *extreme transaction processing* capabilities. Note that no rule operations are stored in the databases: this is because it is more efficient to simply rerun the rules and recreate the appropriate actions, than it is to persist the internal workings of the rule engine.

Main Product Components

This section presents the major components of TIBCO BusinessEvents and how they are used.

Design-time Components

Design time activities performed using the BusinessEvents resources include building an ontology — a set of concepts, scorecards and events that represent the objects and activities in your business — and building rules that are triggered when instances of ontology objects that fulfill certain criteria arrive in events. The output of the design-time activities is an enterprise archive (EAR) file, ready to deploy (or configure for deployment as needed).

See tutorials in *TIBCO BusinessEvents Getting Started* to learn more.

BusinessEvents Studio

Studio is an Eclipse-based project building environment. It organizes project resources and makes the project organization and the project resources visible in graphical ways.

Perspectives

The Eclipse plug-ins for BusinessEvents and for BusinessEvents add-ons provide these perspectives:

BusinessEvents Studio Development Provides resources for building BusinessEvents projects.

BusinessEvents Studio Debug Provides resources for debugging rules and rule functions in BusinessEvents projects.

BusinessEvents Studio Diagram Provides interactive graphical views of a project that allows you to see relationships between project resources and open editors for individual resources.

BusinessEvents Studio Decision Table Provides resources for building decision tables. (Available with TIBCO BusinessEvents Decision Manager.)

BusinessEvents Studio State Modeler Provides resources for building state models. It allows you to model states of ontology concept instances and use those states in rules. (Available with TIBCO BusinessEvents Data Modeling.)

Integration with TIBCO ActiveMatrix BusinessWorks

TIBCO BusinessEvents communicates with TIBCO ActiveMatrix BusinessWorks through a provided plug-in that contains a palette of ActiveMatrix BusinessWorks Activities. Details are provided in *TIBCO BusinessEvents Developer's Guide*.

Add-on Software

The following add-on products are separately available:

- TIBCO BusinessEvents™ Decision Manager provides a business rules application for business users and a rules management server.
- TIBCO BusinessEvents™ Data Modeling provides database concepts and state modeler features
- TIBCO BusinessEvents™ Event Stream Processing provides query and pattern matching features.
- TIBCO BusinessEvents™ Views provides visibility into the data flowing through a running BusinessEvents application, using meaningful metrics that are presented to business users in real-time for proactive decision making.

Administration Components

Administration of a deployed system involves management of objects generated by the inference engine, deploy-time configuration for tuning and other aspects of the system, deployment, management, and monitoring.

Object Managers

Your choice of an object manager depends on the need to persist objects generated by the rules executing in the inference engine. You can manage objects in memory only, or using a persistence database, or using a cache and backing store.

The recommended way to manage objects for most production needs is to use a cache and a backing store. When Cache Manager is used, agents of different types co-operate to provide efficient object storage and access, with options to use load balancing and fault tolerance of data and engine processes.

Object management is partly a design-time and partly an administration topic, because your choice of object management method can affect how you design rules. For example, you may have to retrieve objects if they are stored only in the cache or only in the backing store, so they can be used in the Rete network. See [Chapter 6, Object Management Options, on page 57](#) for an introduction to these topics.

Deploy-time Configuration Using a Cluster Deployment Descriptor (CDD)

The CDD editor enables you to define all the deploy-time properties for the entire cluster, from cluster-wide settings dealing with object management, through processing unit settings (that is, those at the BusinessEvents engine level), to agent class and agent instance settings. At deploy-time, you specify an EAR file, which contains project resources, and also a CDD file and a processing unit (a unit that deploys as an engine). The CDD defines all the deploy settings that apply to the specified unit. Thus, there is no need to maintain separate configuration files for each separate engine (processing unit), and you can change configuration settings without having to rebuild the EAR file. Deploy-time settings include object manager configuration, processing unit and agent class definition, specification of the channels and rules to enable for a particular agent class, and what startup and shutdown functions to run.

Deployment Topology Configuration Using a Site Topology Editor

If you will deploy using the BusinessEvents Monitoring and Management component (see next), then you will use the canvas-based site topology editor to configure the deployment topology. In the topology editor you configure deployment units that deploy to host machines, and you bind them to host machines. Each deployment unit contains one or more processing units (generally one), and each processing unit contains one or more agent classes. The processing units and agent class definitions are read from the CDD file, and you add deploy-time configuration settings.

BusinessEvents Monitoring and Management (BEMM)

BEMM can use the topology file definitions to manage the deployment — deploying, starting, stopping engines and so on. It also makes various methods available for controlling the deployment at different levels. You can also add more engines that are not predefined in the topology file and monitor and manage them (but you can't restart them using BEMM).

BEMM can also monitor the health of the deployment, based on health metric and alert thresholds you configure and display the metrics using a web-based graphical UI. It can send out email when certain conditions occur or take other action. BEMM has a profiler and can generate other helpful reports. BEMM monitoring features enable you to easily spot bottlenecks or other troublespots in the system so you can address any issues.

Designtime Resource Overview

In a rule engine, the things that the project works with such as employees, inventory, parts, and so on are *concepts* in the *ontology* (knowledge model) of the project, as are *scorecards*, which hold metrics. When *TIBCO BusinessEvents Data Modeling* software is used, a database concept feature enables you to create concepts from database data, and a state modeler feature enables you to model the behavior of concepts given certain occurrences.

Events such as flight take-off, purchase of a mortgage, sale of stock, and so on are also part of the ontology. Events can be created from messages arriving through channels, and generated internally, for use in the engine and to send out messages to external systems.

Rules are triggered by events and by changes in concepts and scorecards. For example, rules might cause a baggage to be rerouted if there is a certain problem at the airport. Rule functions are functions written in the rule language that can be called from rules or other rule functions. Some rule functions serve special purposes at startup, shutdown, and in preprocessing events. When *TIBCO BusinessEvents Decision Manager* software is used, its *decision tables* also provide rules. These, however, are *business rules*, and are triggered only indirectly by the inferencing engine.

When *TIBCO BusinessEvents Event Stream Processing* software is used, you can design complex *queries* that provide information on the event stream or on cached objects that can in turn be fed into rules. You can also design event patterns to watch for, and take certain actions when they occur or don't occur.

Designing the ontology and the rules well is key to a good CEP (complex event processing) project.

The sections below describe the features mentioned above in greater detail, with the exception of the features provided in add-on software which are documented in their respective manuals.

Channels and Events

Channels (except for local channels which communicate between agents), represent physical connections to a resource, such as a Rendezvous daemon, JMS server, or HTTP server or client.

A channel has one or more *destinations*, which represent listeners to messages from that resource. Destinations can also be used to send messages to the resource.

Messages arriving through channels are transformed into *simple events*. Conversely, simple events sent out of BusinessEvents are transformed to the appropriate type of message.

BusinessEvents processes three kinds of events. Only simple events are used in channels.

- **Simple Event** A representation of a single activity (usually a business activity) that occurred at a single point in time.
- **Time Event** A timer. Generally created and used to trigger rules.
- **Advisory Event** A notice generated by BusinessEvents to report an activity in the engine, for example, an exception.

BusinessEvents creates instances of simple events and time events based on user-configured event definitions.

See [Chapter 2, Channels and Events, on page 15](#).

Concepts

A *concept definition* is a definition of a set of properties that represent the data fields of an entity. Concepts are equivalent to UML Classes: they represent class-level information, and at runtime the instances of concepts are called objects.

Concepts can describe relationships among themselves. For example, an order concept might have a parent/child relationship with an `item` concept. A department concept might be related to a `purchase_requisition` concept based on the shared property, `department_id`.

With the TIBCO BusinessEvents Data Modeling (purchased separately), concepts can include a state model. Also with the TIBCO BusinessEvents Data Modeling, you can create concepts can be created by importing table and view data from databases, and you can update the database definitions using RDBMS functions. These concepts are called *database concepts*.

Concept properties can be updated by rules and rule functions (including rule functions whose implementation is provided by decision tables).

See [Chapter 3, Concepts, on page 27](#)

Score Cards

A score card serves as a static variable that is available throughout the project. You can use a ScoreCard resource to track key performance indicators or any other information. Use rules to view a scorecard value, use its value, or change its value. Note that unlike concepts and event definitions, which describe types of instances, each scorecard is both the description and the instance.

A score card is similar to a global variable, except that with multiple active inference agents, the value is local to the agent, and you can update the value of a scorecard in rules, but not the value of a global variable.

See [Designing for Concurrency on page 96](#) for some important points about score cards.

Rules

Rules define what constitutes unusual, suspicious, problematic, or advantageous activity within your enterprise applications. Rules also determine what BusinessEvents does when it discovers these types of activities. You can execute actions based on certain conditions which are defined using simple events, concept instances, events, score cards, or a combination of these objects.

Functions

BusinessEvents offers the following types of functions for use in rules:

- **Standard** — These functions are provided with BusinessEvents.
- **Ontology** — BusinessEvents generates these functions based on the resources in your project.
- **Custom** — You can write custom functions using Java and integrate them into BusinessEvents for use in rules.
- **Rule Function** — In addition to Java-based custom functions, you can use rule function resources to write functions using the BusinessEvents rule language.

Standard functions include a set of temporal functions, which allow you to perform calculations based on a sampling of a property's values over time. These functions make use of the history for that property.

See [Chapter 4, Rules and Functions, on page 41](#)

Object Management and Fault Tolerance

An important aspect of most BusinessEvents applications is management of the objects created and modified at runtime.

Although configuring object management is an administrative task, it is important to consider the effect of object storage options when designing projects.

Different projects have different object management requirements. For some, it is acceptable to destroy the objects once the rule engine cycle that needs them has completed. They require only memory-based object management. For others, the instances have longer term value and need to be persisted.

Related to object management is configuration of fault tolerance features. BusinessEvents supports a variety of object management and fault tolerance options.

Cache Object Management and Multi-Engine (Multi-Agent) Mode

Cache object management enables BusinessEvents to run in multi-agent mode, also known as multi-engine mode. In this mode, load balancing, parallel processing, and rule chaining features are available at the agent level.

See [Chapter 6, Object Management Options, on page 57](#) and chapters following

State Modeler

The State Modeler feature is available only with the TIBCO BusinessEvents Data Modeling add-on software. State Modeler is based on the UML-standard definition for State Models. It allows you to model the life cycle of a concept instance — that is, for each instance of a given concept, you can define which states the instance will pass through and how it will transition from state to state.

States have entry actions, exit actions, and conditions, providing precise control over the behavior of BusinessEvents. Transitions between states also may have rules. Multiple types of states and transitions maximize the versatility and power of State Modeler.

See *TIBCO BusinessEvents Data Modeling Developer's Guide*.

Database Concepts

The database concepts feature is available only with the TIBCO BusinessEvents Data Modeling add-on software. Database concepts are BusinessEvents concepts with database behavior. They are created using a utility that enables you to map tables or views from a database to BusinessEvents concepts.

See *TIBCO BusinessEvents Data Modeling Developer's Guide*.

Query Language

Available with TIBCO BusinessEvents Event Stream Processing add-on software, the query features enable you to perform set operations against cached concepts as well as against incoming event streams. Queries can obtain information at a point in time (snapshot queries, for cache queries only). They can also listen to a message stream and collect information continuously.

Queries use an object-oriented SQL-like query language within rule functions. Query results can then be passed using events, or can be shared in cached concepts to be used in other rules or rule functions.

See TIBCO BusinessEvents Query Developer's Guide

Pattern Language

Available with TIBCO BusinessEvents Event Stream Processing add-on software, the Pattern Matcher add-on provides pattern-matching functionality, complementing TIBCO BusinessEvents rule processing and query processing features. Pattern Matcher consists of an easy-to-use language and a service that runs in a BusinessEvents agent. It addresses some of the simpler and more commonly occurring problems in complex event processing such as patterns in event streams, correlation across event streams, temporal (time based) event sequence recognition, duplicate event suppression, and implementation of "Store and Forward" scenarios.

See TIBCO BusinessEvents Query Developer's Guide

Chapter 2 Channels and Events

This chapter explains how messages arrive and leave through channels, and are transformed to and from events.

Topics

- [Channels and Events Overview, page 16](#)
- [Event Preprocessors, page 17](#)
- [Types of Channels, page 19](#)
- [Default Destinations and Default Events, page 20](#)
- [Message Acknowledgment, page 21](#)
- [Types of Events, page 22](#)
- [Simple Events — Time to Live and Expiry Actions, page 24](#)

Channels and Events Overview

Channels (except for local channels) represent physical connections to a resource, such as a Rendezvous daemon, JMS server, or HTTP server or client.

Destinations in a channel represent listeners to messages from that resource, and they can also send messages to the resource. All destinations for a particular channel use the same server.

Arriving messages are transformed into simple events, using message data and metadata. Simple events sent out of BusinessEvents are transformed to the appropriate type of message.

In addition to simple events, which work with incoming and outgoing messages of various sorts, BusinessEvents uses a special-purpose event type called `SOAPEvent`, which inherits from `SimpleEvent`. It is used for sending and receiving SOAP messages in an HTTP channel. Two other types of events are also used: time events and advisory events. These event types are described in [Types of Events on page 22](#).

Event Preprocessors

Event preprocessors are rule functions with one argument of type simple event. (Event preprocessors are not used for time or advisory events.) Event preprocessors are multithreaded. They perform tasks after an incoming message arrives at the destination and is transformed into a simple event, but before it is asserted into the Rete network (if it is — events can be consumed in the event preprocessor).



Time events do not go through an event preprocessor. If you are using cache-only cache mode, take care when designing rules that execute as a result of a time event. For example, a rule that has a join condition using a time event and a concept would not execute if the concept is not loaded in the Rete network.

You can aggregate events, edit events, and perform other kinds of event enrichment in a preprocessor. You can also use preprocessors as explained below.

Setting locks if
concurrency
features are used

You must set locks in the preprocessor when concurrency features are used to protect concept instances during RTC. Locking ensures that updates to concept instances during an RTC do not conflict with another set of updates to the same concept instances in another RTC. Locks are released at the end of the RTC.

Loading cache
only entities

If you are using the Cache Only cache mode for any entities, you must also load the relevant entities from the cache using an event preprocessor.

Improving project
efficiency

You can also use preprocessors to improve performance by avoiding unnecessary RTCs in the inference engine. For example you can consume events that are not needed. Another way to use the preprocessor for efficient processing is to transfer an event's contents to a new concept that is not processed by the agent's set of locally active rules. Such a concept is automatically asserted, does not trigger rules, and is saved into the cache (depending on OM configuration) where it is available for processing by any agent as needed.

Preprocessor Use Guidelines

Consuming events in a preprocessor is allowed It can be useful in some applications and reduces the flow of messages into the Rete network.

You can only modify events before they are asserted into the Rete network Rule evaluation depends on event values at time of assertion, so values can be changed only before assertion, that is, in the preprocessor.

You can create concepts but not modify or delete existing concepts Modifying or deleting concepts that already exist in the system could disrupt an RTC. You can modify or delete concepts that were created in the same preprocessor, however.

See Also

[Loading Cache-Only Objects into the Rete Network on page 93](#)

[Using Locks to Ensure Data Integrity Within and Across Agents on page 101](#)

Types of Channels

BusinessEvents provides the following types of channels:

- **Local channels** Connect co-located agents. One use for local channels is to enable a query agent to make use of a co-located inference agent for additional processing.
- **TIBCO Rendezvous channels** Connect TIBCO BusinessEvents to TIBCO Rendezvous sources and sinks.
- **HTTP channels, including SOAP support** An HTTP channel acts as an HTTP server at runtime, enabling BusinessEvents to serve requests from clients, as well as to act as a client of other servers
- **JMS channels** Connect TIBCO BusinessEvents to TIBCO Enterprise Message Service provider sources and sinks.
- **TCP channels** connect to data sources not otherwise available through channels, using a catalog of functions.



Each JMS Input Destination Runs a Session

Every JMS destination that is configured to be an input destination runs in its own JMS Session. This provides good throughput on queues and topics for processing, and less connections.

Support for SOAP Protocol

Support for SOAP protocol is provided by these features (using SOAP over HTTP):

- A specialized event type whose payload is configured as a skeleton SOAP message
- A set of functions for extracting information from SOAP request messages and constructing response messages.
- A utility that constructs project resources based on the SOAP service's WSDL file (document style WSDLs with literal encoding are currently supported).

Default Destinations and Default Events

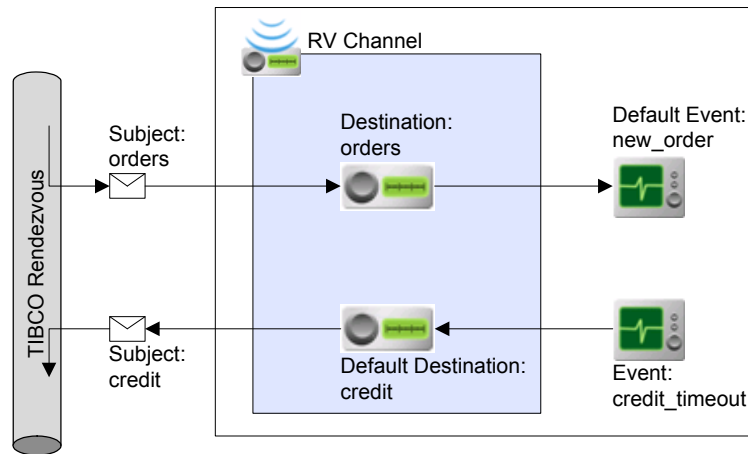
Using default destinations and default events simplifies project configuration for many scenarios.

Incoming Messages

Incoming messages can be mapped to a default event that you specify when you configure the destination. All messages arriving at that destination are transformed into the default event type, unless they specify a different event.

For example, in [Figure 1](#) the channel is configured to listen to the flow of messages on Rendezvous. The *orders* destination directs BusinessEvents to map messages coming in on the subject, *orders*, to the *new_order* simple event.

Figure 1 Channels and Destinations



You can map incoming messages to specified event types. The technique is explained in Mapping Incoming Messages to Non-default Events in *TIBCO BusinessEvents Developer's Guide*.

Outgoing Messages

Outgoing messages can be sent to a default destination. When the destination is not otherwise specified (for example in rules or rule functions), events are sent to the destination you select as their default destination.

For example, in [Figure 1](#) the event *credit_timeout* is sent out through its default destination *credit*.

You can send an event to the default destination of its event type using the `Event.sendEvent()` or `Event.replyEvent()` functions.

You can send an event to a specified destination using the `Event.RouteTo()` function.

Message Acknowledgment

For each message type (that is, each type of channel), BusinessEvents acknowledges the receipt of messages according to the protocol of the messaging system. Some messages do not require acknowledgement. For example reliable Rendezvous messages do not require acknowledgment.

Message Acknowledgment Timing

An event is acknowledged as follows:

- In a preprocessor: Immediately after the event is consumed.
- During a run to completion (RTC) cycle:
 - With Cache OM, during the post RTC phase.
 - With Persistence OM, after the event is written to disk, during a checkpoint.
 - With In Memory OM, during the post RTC phase, but only if the event has been explicitly consumed.

Types of Events

BusinessEvents processes three kinds of events:

- **Simple Event** A representation of a single activity (usually a business activity) that occurred at a single point in time. The `SOAPEvent` event type inherits from `SimpleEvent`.
- **Time Event** A timer. Time events can be configured to repeat at intervals, or they can be scheduled using a function in a rule or rule function.
- **Advisory Event** A notice generated by BusinessEvents to report an activity in the engine, for example, an exception.

BusinessEvents creates instances of simple events and time events based on user-configured event definitions. The following sections provide more detail on each type of event.



Inheritance You can use inheritance when defining simple events.

Attributes In addition to user defined properties, events have built-in attributes for use in rules and rule functions. For example, simple events have these attributes: `@id`, `@extId`, `@ttl`, and `@payload`. Concepts and scorecards also have built-in attributes. See *TIBCO BusinessEvents Developer's Guide* for details.

Simple Events

A *simple event definition* is a set of properties related to a given activity. It includes information for evaluation by rules, meta-data that provides context, and a separate payload -- a set of data relevant to the activity.

For example, suppose you are interested in monitoring the creation of new employee records. You might create a simple event definition that includes important fields from the employee record, perhaps the social security number, department, and salary. You could then write a rule to create an instance of this simple event each time a new employee record is created.

A *simple event* is an instance of a simple event definition. It is a record of a single activity that occurred at a single point in time.

Just as you cannot change the fact that a given activity occurred, once an event is asserted into the Rete network, you can no longer change it. (Before assertion you can use an event preprocessor to enrich the event, however.) Simple events expire when their time to live has elapsed, unless BusinessEvents has instructions to consume them prior to that time.

Example 1: A temperature sensor records a reading that is above a predefined limit. The payload might include the temperature-sensor ID, the reading, and the date and time. This simple event might trigger a complex event that would immediately notify a manager.

Example 2: A customer purchases four airline tickets from San Francisco, California to San Jose, Costa Rica. The payload might include the date and time of purchase, the date and time of the flight, the purchase price, the credit card number, the flight number, the names of the four passengers, and the seat assignments. This simple event alone may include no exceptions. However, it is possible that when examined within the context of other related events, an exception may arise. For example, one or more of the passengers may have booked tickets on another flight during the same time period.

Time Events

A time event is an event definition that triggers the creation of event instances based on time. There are two ways to configure a time event:

- **Rule based** A rule schedules the creation of a time-event instance at a given time.
- **Time-interval based (Repeat Every)** `BusinessEvents` creates a time-event instance at regular intervals.

Advisory Events

Advisory events are asserted into the Rete network automatically when certain conditions, for example, exceptions, occur. Add the `AdvisoryEvent` event type to rules to be notified of such conditions. An advisory event expires after the completion of the first RTC cycle (that is, the time to live code is set internally to zero). The types of advisory events are described next.

Exception The `BusinessEvents` engine automatically asserts an advisory event when it catches an exception that originates in user code but that is not caught with the `catch` command of the `BusinessEvents` Exception type. (For information on working with exceptions, see Exception Handling in *TIBCO BusinessEvents Developer's Guide*.)

BusinessEvents-ActiveMatrix BusinessWorks Integration Advisory events are also used in the container mode `BusinessEvents-ActiveMatrix BusinessWorks` integration feature `invokeProcess()` function. Such events are asserted when the `ActiveMatrix BusinessWorks` process fails or times out (or is cancelled).

Engine Activated Advisory Event An advisory event is asserted when an engine has finished starting up and executing startup functions (if any).

Simple Events — Time to Live and Expiry Actions

Events have a time to live (TTL) setting. Events can't be modified after they are initially asserted, but they can continue trigger rules during their time to live.

Events Recovered From Cache or Persistence Store

When Cache object management is used, events with a sufficiently long time to live (TTL) setting are cached. Similarly, when Persistence object management is used, such events are persisted in the data store.

With Persistence and Cache OM types, the TTL period is re-evaluated when an event is recovered from the persistence database or reloaded from cache. For example, if the TTL is 60 minutes and event is recovered 30 minutes after it is asserted, then its remaining TTL is 30 minutes.

Using TTL Options to Trigger Rules Correctly

Set the event's time to live so that it can trigger the rules you intend. If a rule correlates different events, you must ensure that those event instances are all in the Rete network concurrently. Time to live options are as follows:

- Zero (0): the event expires after the completion of the first RTC cycle. Do not set to 0 if you want to correlate the event with a future event or other future occurrences of this event, as explained below.
- One or higher (>0): the event expires after the specified time period has elapsed. The TTL timer starts at the end of the action block of the rule or preprocessor function in which the event is first asserted.
- A negative integer (<0): the event does not expire, and must be explicitly consumed.

Example Consider the following example:

- A process sends eventA, eventB, and eventC.
- The TTL for all three simple events is 0.
- Rule 1 has the condition: `eventA.id == eventB.id`.
- Rule 2 has the condition: `eventC.id != null`.

At runtime, BusinessEvents behaves as follows:

1. BusinessEvents receives eventA. Because there is no eventB in the Rete network, eventA doesn't trigger any rules. BusinessEvents consumes eventA.

2. BusinessEvents receives eventB, but eventA has been consumed — there is no eventA in the Rete network. So eventB does not trigger any rules. BusinessEvents consumes eventB.
3. BusinessEvents receives eventC, which triggers Rule 2 because Rule 2 depends only on eventC.

To trigger Rule 1, you must configure the time to live for eventA and eventB to ensure that both events will be in the Rete network concurrently. You can trigger Rule 1 in these ways:

- If you know that eventA is sent before eventB, set the TTL for eventA to a time greater than the maximum period that can elapse between sending eventA and sending eventB.
- If you don't know the order in which eventA and eventB are sent, set the TTL for both simple events to a time greater than the maximum time between the occurrence of the two simple events.

Event Expiration and Expiry Actions

After the time to live (TTL) period, the event expires and is deleted from the Rete network. Any expiry actions are taken.

With Persistence object management, the expired event is marked for deletion.

With Cache object management, events TTL is evaluated when the event is retrieved from the cache.

Expiry Actions

For each simple event definition, BusinessEvents allows you to specify one or more actions to take when the event expires, using the BusinessEvents rule language. For example, you can write an action that routes the simple event to a different destination, sends a message, or creates a new event. This action can be anything that is possible with the rule language.

An expiry action can be inherited from the event's parent.



If an event is explicitly consumed in a rule, BusinessEvents does not execute the expiry action.

Chapter 3 **Concepts**

This chapter discusses BusinessEvents concepts, which are created from information in messages or built in rules.

Topics

- [Overview of Concepts on page 28](#)
- [Concept Property History on page 30](#)
- [Concept Relationships on page 34](#)

Overview of Concepts

Concept types are descriptive entities similar to the object-oriented concept of a class. They describe a set of properties. For example, one concept might be Department. The Department concept would include department name, manager, and employee properties.

Runtime Behavior of Concepts

Rules at runtime can create instances of concepts. For example, when a simple event arrives, a rule can create an instance of a concept using values present in the event. Rules can also modify existing concept instance property values.

Concepts must be explicitly deleted from working memory when no longer needed or they will steadily increase memory usage. Use the function `Instance.deleteInstance()` to delete concept instances.

Depending on other factors, adding, modifying, and deleting concept instances can cause BusinessEvents to evaluate or re-evaluate dependent rules, as explained in [Understanding Conflict Resolution and Run to Completion Cycles on page 51](#).



Concepts are automatically asserted into the Rete network when created, except in the following cases:

- Database concepts returned by database query operations (requires TIBCO BusinessEvents Data Modeling).
- Concepts passed to a rule function in the context of ActiveMatrix BusinessWorks integration projects.

Understanding Concept Property History

Each concept property includes a history, the size of which is configurable. The history size determines how many previous values BusinessEvents stores for that property. See [Concept Property History on page 30](#).



Database concept properties do not support history tracking (Available in TIBCO BusinessEvents Data Modeling).

Concept Relationships

Concepts can have inheritance, containment and reference relationships with other concepts. See [Concept Relationships on page 34](#).

Exporting Concepts to XSD Files

You can export concept and event types to XML Schema Definition (XSD) files. XML schemas are used for interoperability between BusinessEvents and third party tools or SOA platforms that use well-defined XML for message communication, transformation, and validation.

Concept Property History

Each concept property includes a history, the size of which is configurable. The history size determines how many previous values BusinessEvents stores for that property. You can also set the history policy to record all values or only changed values.



For ContainedConcept and ConceptReference properties History is tracked when a contained or referenced concept instance changes to a different concept instance. History is *not* tracked, however, when a contained or referenced concept's properties change. See [Concept Relationships on page 34](#) for more on containment and reference relationships.

Database concepts Database concept properties do not support history tracking.

History Size

If you set the history size to one or more, BusinessEvents stores the property value when the property changes, along with a date and timestamp, up to the number specified. When the maximum history size is reached, the oldest values are discarded as new values are recorded.

If you set the history size to 0, BusinessEvents does not store historical values for the concept. It stores the value without a time and date stamp.

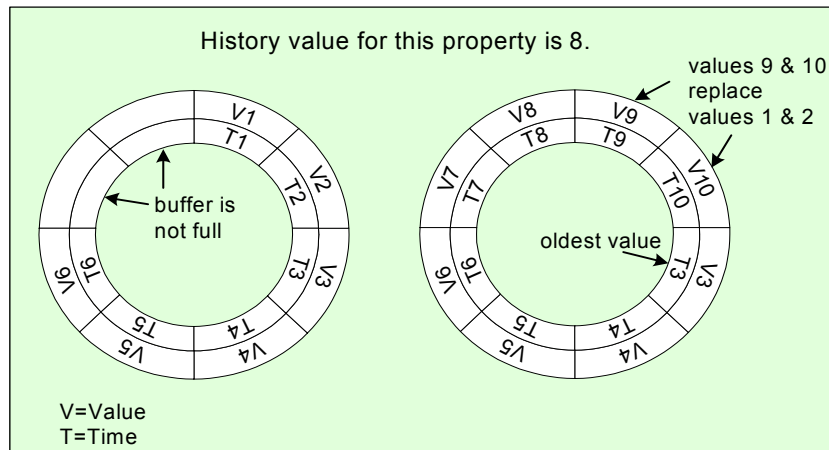
For example, consider a Customer concept:

Property Name	History	Comments
customer_name	1	These properties tend to be very stable and you may have little interest in tracking a history for them.
customer_address	1	
city	1	
state	1	
zip	1	
account_number	0	With history size 0, BusinessEvents does not record the timestamp when the value is set.
credit_limit	4	Credit limit may change more frequently and you may have an interest in tracking the changes.

Historical Values are Stored in a Ring Buffer

The historical values for a concept property are kept in a ring buffer, as illustrated in [Figure 2](#).

Figure 2 History Ring Buffer



The ring buffer stores both the value and the time at which the value was recorded. After the ring buffer reaches maximum capacity, which is eight in this example, BusinessEvents begins replacing older values such that it always stores the n most recent values, where n is the history size. Note in [Figure 2](#) in the ring buffer on the right, after the buffer reached maximum capacity, V9 replaced V1, then V10 replaced V2, making V3 the oldest value stored in the ring buffer.

History Policy

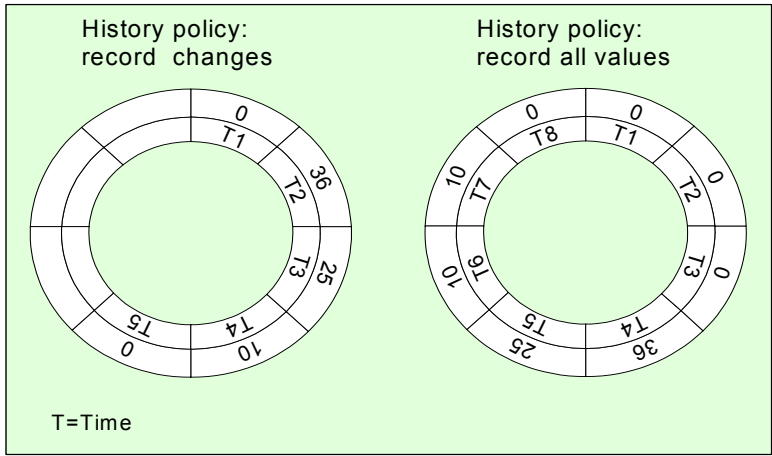
BusinessEvents can record values using either of these policies:

- **Changes Only** BusinessEvents records the value of the property every time it changes to a new value.
- **All Values** BusinessEvents records the value of the property every time an action sets the value even if the new value is the same as the old value.

Which you choose depends on what you are tracking. For example, if you are setting the history for a property that tracks how many people pass a sensor every five minutes, All Values might be the best policy. However, if you are setting the history for a property that tracks the level of liquid in a coffee pot, Changes Only might be more appropriate.

For example, look at the two ring buffers in [Figure 3](#). In both cases, the same series of values is set to the same property, but the history policy is different.

Figure 3 History Policy





History Policy and Rule Evaluation The history policy affects how frequently BusinessEvents re-evaluates rules that are dependent on the property. Each time BusinessEvents records a value, it reevaluates rules that are dependent on that property. If you track changes only, rules are re-evaluated less frequently than if you track all values.

Concept Relationships

Concepts can have inheritance, containment and reference relationships with other concepts.

Inheritance Relationships

Definition: One concept inherits all the properties of another concept, similar to Java, where a subclass inherits all the properties of the superclass that it extends. You can define a hierarchy of inheritance, where each concept in the hierarchy extends the concept above it.

The relationship is defined by the Inherits From field in the concept resource editor.

Concepts that are related to each other directly or indirectly by inheritance cannot have distinct properties that share a common name. Therefore, the following restrictions apply:

- If two concepts are related by inheritance, you cannot create a new property in one with a name that already exists in the other.
- If two unrelated concepts have properties that share a name, you cannot create an inheritance relationship between the two concepts.

Also, BusinessEvents does not allow you to create an inheritance loop; for example, if Concept A inherits from Concept B, Concept B cannot inherit from Concept A.

At runtime, a rule on a parent concept also affects all its child concepts. For example, suppose the concept Coupe inherits from the concept Car. A rule on Car is therefore also a rule on Coupe.

Containment Relationships

Definition: One concept is contained inside another concept. You can define a hierarchy of containment, where each concept in the hierarchy is contained by the concept above it.

The relationship is defined using a `ContainedConcept` property in the container concept.



When working with container and contained concepts in the rule editor, the XSLT mapper and XPath builder show the entire hierarchy of properties.

In the rule editor, you can also use the `@parent` attribute to get the parent of a contained concept instance.

Deep containment relationships can also cause memory issues. When `BusinessEvents` retrieves a concept from cache, its child concepts are also retrieved. When you modify a child concept, its parent concepts are considered to be modified. It is recommended that you keep concept relationships shallow. See [Table 3, Containment and Reference Concept Relationship Rules, on page 38](#) for these and other rules governing the behavior of concepts linked by containment, and also reference. The table can also help you to choose which is the appropriate type of relationship for your needs.

Containment Example: Car with Wheels, Doors, and Engine

The following example illustrates some of the rules that are listed in [Table 3](#). To configure a concept `Car` to contain a concept `Wheel`, you add a `ContainedConcept` property, `Wheels` for example, whose value is an instance of the concept `Wheel`. The `Wheels` property provides the link between the container and contained concept:

Car (Concept) — Wheels (property) — Wheel (Concept)

The concept `Car` contains four instances of the contained concept `Wheel`, so you define the property as an array.

The concept `Car` could also contain other concepts, such as `Door` and `Engine`, defined in a similar way using `ContainedConcept` properties.

However, the contained concepts — Wheel, Door, and Engine — cannot be contained by any other concept type. They can only be contained by the Car concept. For example, the concept Wheel cannot be contained in the concept Motorbike, if it is already contained by the concept Car.



A container concept can link to a contained concept using only *one* ContainedConcept property. You can use inheritance, however, to achieve a result similar to that gained by the general programming technique of linking to multiple contained class properties. Suppose you extend the concept Wheel by creating child concepts CarWheel and MotorcycleWheel. You can then use CarWheel as the concept contained by Car, and MotorcycleWheel as the concept contained by Motorcycle. Rules that apply to Wheel also apply to CarWheel and MotorcycleWheel, because of inheritance.

Depending on your needs, another option would be to use a reference relationship instead of a containment or inheritance relationship.

Reference Relationships

Definition: One concept instance references another concept instance. A concept that is the target of a reference can itself refer to one or more other concepts. Reference relationships are not, however, hierarchical.

The relationship is defined by a `ConceptReference` property in the referring concept.

See [Table 3, Containment and Reference Concept Relationship Rules, on page 38](#) for rules governing the behavior of concepts linked by containment or reference. The table also helps you to choose which is the appropriate type of relationship for your needs.



Properties of concept references cannot be used in a condition.

Reference Example: Order with SalesRep and Customer

The following example illustrates some of the rules that are listed in [Table 3](#).

To configure a concept `Order` to reference a concept `SalesRep`, you add a `ConceptReference` property, `Rep` for example, whose value is the ID of concept `SalesRep`. The `Rep` property provides the link between the referring and referenced concepts:

`Order (Concept)—Rep (property)—SalesRep (Concept)`

You can also define additional reference relationships such as:

`Order (Concept)—BackupRep (property)—SalesRep (Concept)`

`Order (Concept)—Lines (property array)—LineItem (Concept)`

`Order (Concept)—Cust (property)—Customer (Concept)`

`Customer (Concept)—Orders (property)—SalesRep (Concept)`

Reference Examples: Self Reference

A concept definition can have a reference relationship to itself. This is generally because the instances of one concept definition can have reference relationships to other instances of the same definition. For example:

- A `ListItem` concept has a `next` property which is a reference to a `ListItem` concept.
- A `Person` concept has a `spouse` property which is a reference to a `Person` concept.
- A `Person` concept has a `children` property which is an array of references to `Person` concepts.

Rules Governing Containment and Reference Relationships

[Table 3](#) presents the rules governing design-time and runtime use of containment and reference relationships. By comparing the rules, you can decide which type of relationship to use in a particular case.

Table 3 Containment and Reference Concept Relationship Rules

Containment One concept is contained in another	Reference One concept points to another
Designtime Rules	
One container concept can contain multiple different contained concepts, and a contained concept can itself also act as a container concept.	
One referring concept (that is, the concept that has the <code>ConceptReference</code> property) can have a reference relationship with multiple referenced concepts, and a referenced concept can also refer to other concepts.	
A container concept can link to a contained concept using only one <code>ContainedConcept</code> property. (Some other object oriented languages do allow you to reuse object types in parent object properties.)	A referring concept link to a referenced concept using multiple <code>ConceptReference</code> properties. (That is, multiple <code>ConceptReference</code> properties can reference the same referenced concept.)
A contained concept can have only one container concept.	A referenced concept can be referred to by multiple referring concepts
Runtime Rules	
When one contained instance is replaced with another , <code>BusinessEvents</code> deletes the instance that it replaced automatically. You do not have to delete the replaced instance explicitly.	When one referenced instance is replaced with another , <code>BusinessEvents</code> does <i>not</i> delete the instance that it replaced automatically. It may not be appropriate to delete the referenced instance. If you want to delete the referenced instance, do so explicitly.
When a contained instance is modified , the container instance is also considered to be modified. The reasoning can be seen by a simple example: a change to the wheel of a car is also a change to the car. Rules that test for modified instances would return the <code>Car</code> concept instance as well as the <code>Wheel</code> concept instance.	When a referenced instance is modified , the referring instance is <i>not</i> considered to be modified. The reasoning can be seen by a simple example: a change to the support contract for a customer is not a change to an order that references that customer.
When a container instance is asserted or deleted , the contained instance is also asserted or deleted, along with any other contained instances at lower levels of the containment hierarchy.	When a referring instance is asserted or deleted , the referenced instance is <i>not</i> also asserted or deleted.

When a Contained or Referred Concept Instance is Deleted

This section highlights an important difference in behavior when history is tracked for an array property, and when history is not tracked, or the property is not an array.

Property Settings (ContainedConcept or ConceptReference Property)	Effect of deleting a Contained or Referenced Concept:
Single value property, regardless of history setting.	The value of the ContainedConcept or ConceptReference property becomes null.
Multiple-value property (array), with History is set to 0 or 1 (that is, historical values are not tracked).	The array entry that held the deleted concept is removed, reducing the array size by one.
Multiple-value property (array), whose History is set to 2 or more (that is, historical values are tracked).	The array entry that held the deleted concept remains and its value is set to null, so that history can be tracked.

For more on history, see [Concept Property History on page 30](#).

Chapter 4 **Rules and Functions**

This chapter considers the part that rules and functions play in a BusinessEvents application.

Topics

- [Rules, page 42](#)
- [Rule Functions, page 44](#)
- [Startup and Shutdown Rule Functions, page 45](#)

Rules

Most rules in BusinessEvents are used for inferencing. However, regular business rules also have a role to play.

Inferencing Rules

Inferencing rules are at the heart of BusinessEvents. Inferencing rules are declarative, and at runtime are executed based on the outcome of each conflict resolution cycle (see [Understanding Conflict Resolution and Run to Completion Cycles on page 51.](#))

A rule includes the following parts:

- A declaration of entity types
- (optionally) one or more separate conditions, which evaluate to true or false
- An action, which is eligible to execute only when all conditions evaluate to true.

Statements in a rule action might create or modify concept instances, create and send simple events, call functions and rule functions, and so on depending on need.

Rule Priority and Rank

For each RTC, the rule agenda is sorted by priority and then within priority by rank, for those rules that use the same ranking mechanism. Use of priority and rank is optional. You can also use priority without using rank.

TIBCO recommends that you use priority and rank features only as needed; that is, unless there is reason to set priority (or priority and rank), let the rule engine determine the sequence of execution. This lessens the complexity of rule maintenance, and takes advantage of the power of the inferencing engine.

Rule Priority

Because BusinessEvents rules are declarative rather than procedural, there is no inherent order for processing. However, a priority property allows you to specify the order in which rules in one RTC execute.

Rule Rank Within the Same Priority

If you want to also control the order in which rules with the same *priority* execute,

you can use the rule rank feature. The value for the Rank property is a rule function that returns a double. The larger the return value, the higher the ranking. You can specify the same rule function in different rules to perform ranking across tuples of those rules.

Other Rules

Not all rules in BusinessEvents are inferencing rules. Rules in decision tables are business rules, executed only when the table is invoked.

Organizing and Deploying Inferencing Rules

You can organize rules depending on your project and project maintenance needs. Rules are organized in folders. At deploy time you can select folders of rules or individual rules (or both) for deployment.

Rule Functions

A rule function is a function written in the BusinessEvents rule language. All rule functions created for a project are available project-wide.

Rule functions can take arguments and can return a value. The return type can be set to void, indicating that the rule function does not return anything.

Like other types of functions, you can use rule functions in rule conditions and rule actions.

You can use project settings to use rule functions as preprocessors (see [Event Preprocessors on page 17](#)) and as startup and shutdown actions.

Virtual Rule Functions and Decision Tables

Decision tables are available with TIBCO BusinessEvents Decision Manager add-on software.

A virtual rule function (VRF) has arguments but no body or return type. The implementation of a virtual rule function is a decision table. Business users can create decision tables in Decision Manager stand-alone business user interface. Decision tables can also be created in the BusinessEvents user interface.

Users start by selecting a virtual rule function. They drag and drop entities from an argument explorer to form rows in a decision table. Each row forms a business rule, for example the condition area might specify that age is less than 18, and the action area might specify that credit is refused. More technical users can use the BusinessEvents rule language to create more complex rules.

One VRF can have multiple implementations. You can set a priority that determines the order of execution for multiple implementations of a VRF. Functions are also available for choosing an implementation to execute (and other actions specific to decision tables). If there is just one implementation, you can call the virtual rule function in the same way you call any other rule function.

Startup and Shutdown Rule Functions

Startup and shutdown rule functions are rule functions that are configured to execute during normal system startup and shutdown, respectively.

Startup and shutdown rule functions take no arguments and their Validity setting must be Action (meaning they can't be used in conditions or queries).



See Appendix C, Engine Startup and Shutdown Sequence in *TIBCO BusinessEvents Administration* for a useful reference that helps you understand what you can do in startup and shutdown actions.

Startup Rule Functions

Startup rule functions are optional and are used to initialize the system. For example they can provide initial values for scorecards. Startup rule functions can be used to perform more "expensive" operations so that the system is more efficient at runtime. For example, in a startup rule function you might load specified entities from the backing store to the cache.

Startup rule functions may trigger rule actions. However, note that BusinessEvents executes all startup rule functions before it begins the first RTC cycle, which completes when all rules eligible to execute have executed and no more actions remain.

Shutdown Rule Functions

Shutdown rule functions are optional and are used to perform various actions during a normal shutdown, for example, they can send events to external systems.

When Startup Rule Functions Execute

Startup rule function execute on startup of an active node.

In recovery situations, startup rule functions execute on failback to a failed node that has restarted.

However, if recovery is from a situation that does not involve node failure, then startup actions do not execute. For example, the network connection goes down. The agent becomes inactive and fails over to another node. The connection is restored. The agent becomes active again, but does not restart. Startup functions do not execute on the node that became active again.

If you want to execute startup rule functions on only one node in a deployment, use programming logic to do so.

Creating Entities With a Startup Action in a Multi-Engine Project

Startup (and shutdown) rule functions execute in all active agents. When multi-engine (multi-agent) functionality is used, ensure that multiple agents do not attempt to create the same entity. See [Designing for Concurrency on page 96](#) for more information.

ActiveMatrix BusinessWorks Containers

In ActiveMatrix BusinessWorks integration projects, if ActiveMatrix BusinessWorks is running as the container, do not specify any startup actions that result in starting or invoking an ActiveMatrix BusinessWorks process.

Note that after the ActiveMatrix BusinessWorks engine is initialized, processes that invoke BusinessEvents rule functions will fail if the BusinessEvents engine has not finished starting up. For example, an ActiveMatrix BusinessWorks process that listens to a JMS queue may attempt to invoke a BusinessEvents rule function before the BusinessEvents engine has started up.

Chapter 5 **Run-time Inferencing Behavior**

This chapter explains rule evaluation in the inferencing engine and related topics.

Topics

- [Runtime Architecture and Flow, page 48](#)
- [Understanding Conflict Resolution and Run to Completion Cycles, page 51](#)

Runtime Architecture and Flow

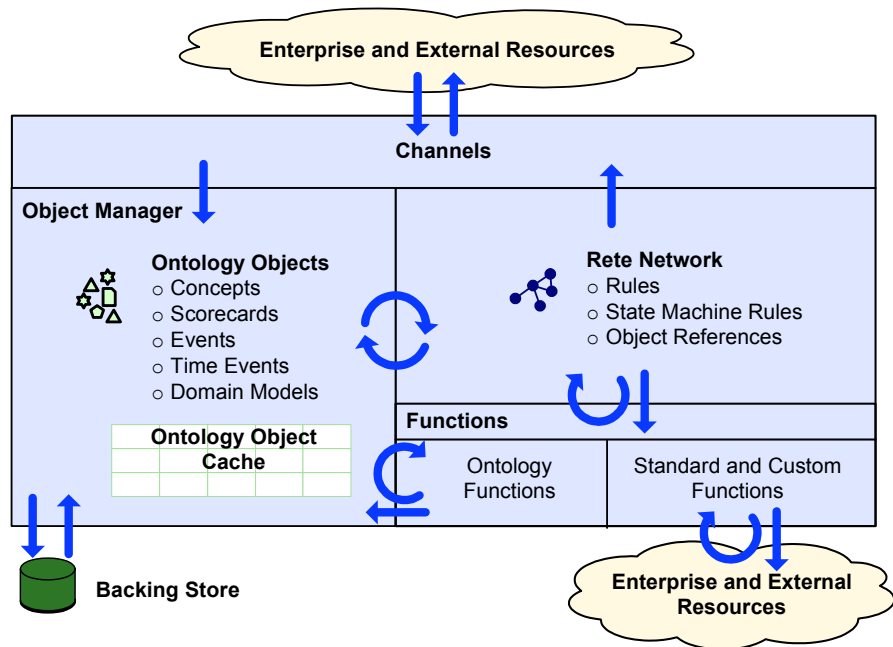
At runtime, one or more nodes (JVMs) running one or more BusinessEvents *inference agents* process the incoming events using a Rete network as the inferencing engine, and a set of rules that are triggered by conditions in incoming events. One or more event stream processing query agents can query incoming events.

BusinessEvents has two layers of functionality:

- **Rules Evaluation and Execution** based on the state and value of objects and incoming events. This functionality is achieved using one or more inference agents configured with the appropriate rules. Each inference agent executes rules using one or more Rete networks to optimize performance and provide rule inferencing capabilities.
- **Lifecycle Management of Objects and Events** including distribution, clustering, persistence and recoverability. Various options are available to achieve the levels of functionality appropriate for business needs, from in-memory only storage of objects, to advanced caching features and a backing store (database). [Figure 4](#) shows cache object management.

In addition, when TIBCO BusinessEvents Event Stream Processing software is used, a third layer is added: queries and pattern matching. A query agent enables visibility into the event stream and cache data. Pattern matching features enable actions to be taken on recognition of a pattern of events, or failure to complete a pattern of events.

Figure 4 TIBCO BusinessEvents Architecture



Rule Evaluation and Execution

Information from enterprise applications and other sources flows into BusinessEvents through channels as messages. Messages represent the events that BusinessEvents processes based on event definitions (event types). Events can be filtered (ignored), preprocessed into concepts or cached concepts, or asserted into the rule engine's working memory.

In an inference agent, all the rules whose conditions match information in the events (as well as concepts, if specified in the rule conditions) are assembled into a *rule agenda* and the first rule executes. If a rule successfully executes, its rule actions create and modify the objects in working memory. The rule agenda is derived from an internal runtime memory structure known as a *Rete network* (because it uses a derivative of the *Rete algorithm*).

BusinessEvents rule engine is a forward-chaining inferencing engine. Every time the facts (concepts, score cards, and events) in its working memory change — due to rule actions or the arrival of new events — the inferencing engine updates the rule agenda. As a result, new rules are available to execute while others are now unavailable. The selection of which rule to execute first from a choice of several is

called *conflict resolution*. The agenda process repeats until there is no more new information to process. This is known as RTC, or *run to completion*. See [Understanding Conflict Resolution and Run to Completion Cycles on page 51](#) for more details.

(Note that State Machine, is present only in TIBCO BusinessEvents Data Modeling add-on software).

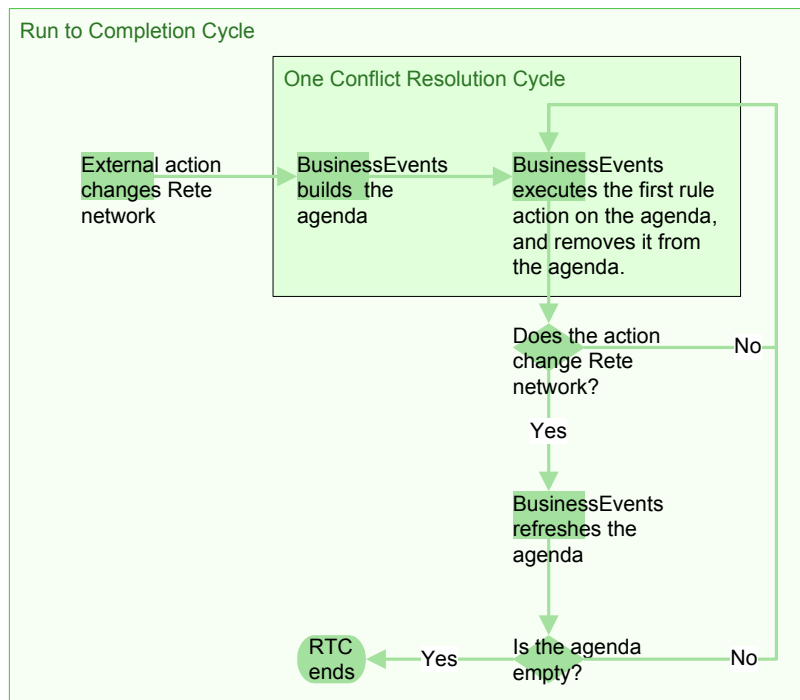
Understanding Conflict Resolution and Run to Completion Cycles

This section helps you to understand what triggers rules to execute, and why a rule may not execute, so that you can design rules more effectively.

Run to completion cycle

A *run to completion*, or RTC, cycle generally begins when an external action causes changes to the Rete network. It ends when there are no more rule actions to execute as a result of that initial change (and any subsequent changes caused by rule actions). This is also known as *forward chaining*, or *inferencing*. During one RTC changes can occur in the Rete network, but no new *external* actions can affect it.

Figure 5 Run to Completion Cycle



Conflict resolution cycle

One RTC is composed of one or more *conflict resolution cycles*. A conflict resolution cycle begins when BusinessEvents builds (or refreshes) a *rule action agenda*, a list of all rules that are eligible to fire. The agenda is used to determine which rule action to execute next. The agenda is built based on the following information:

- The scope and conditions of the rules in the project.
- The current contents of the Rete network.

One conflict resolution cycle ends when a rule action is executed (or the agenda is empty). If the rule action changes the contents of the Rete network, another conflict resolution cycle begins.

A More Detailed Look

The Rete network changes The first of the conflict resolution cycles is initiated by change in the Rete network, caused by an external action such as a message arriving at a destination.

All subsequent changes to the Rete network during one RTC occur because of rule actions.

BusinessEvents builds the agenda To build the rule action agenda, BusinessEvents examines all rules that are *newly true* because of the change to Rete network and compares them with rule dependencies. The agenda's entries are ordered according to rule priority, rule rank, and other criteria.

See [How the Rete Network is Built on page 53](#) and [How a Rule Becomes Newly True on page 54](#) for more details.

BusinessEvents executes the first rule on the agenda and removes it from the agenda As a result, one of the following occurs:

- The rule action does not change the Rete network and BusinessEvents executes the next rule entry in the agenda (if there is one).
- OR
- The rule action does change the Rete network and BusinessEvents refreshes the rule action agenda (see next section).



Events created during an RTC are not sent to destinations until the entire RTC is complete.

Similarly, objects are not written to cache until the entire RTC is complete.

Next conflict resolution cycle

BusinessEvents refreshes the rule action agenda If a rule action changes the contents of the Rete network, the agenda is refreshed, beginning a new conflict resolution cycle. When the agenda is refreshed, any of the following can occur:

- Rules that have become newly true are added to the agenda.
- Rules that have become false are dropped from the agenda.
- Rules that were newly true at the last conflict resolution cycle and are still true remain in the agenda. (In other words, rules are newly true for the duration of the run to completion cycle unless they become false.)

As a result, either the agenda is empty and the RTC ends, or the first rule in the refreshed agenda is executed, ending this conflict resolution cycle and beginning the next one.

An empty agenda marks the end of one RTC

An empty agenda ends the RTC At some point, no more actions remain to be executed. The conflict resolution has run to completion. The RTC is over. Now begins the post RTC phase. At various points during this phase the following actions happen (depending on how the project has been configured):

- Events are sent to destinations.
- Cache OM: Changes are saved to the cache and written to the backing store.
- Cache OM, cache-only cache mode: All cache-only objects are removed from the Rete network. See [Loading Cache-Only Objects into the Rete Network, page 93](#) for important information about working in cache only mode.
- Persistence OM: One transaction is completed and saves changes (enabling rollback in case of failures).
- Profiler: profiler data is updated.

How the Rete Network is Built

Before any data enters the system, BusinessEvents builds the Rete network, which embodies all the *rule dependencies*, using the rule conditions (if any). All the dependencies in a rule are called its *dependency set*.

A Rule's Dependency Set

A rule's dependency set is everything needed to determine the truth of all the conditions. For example, a rule has this condition:

```
c.name == "Bob";
```

Where *c* is a concept of type */Customer*. In this case, the dependency set of the rule contains only the name property of the concept type */Customer*.

As another example, suppose a rule has these conditions:

```
b.num<10;
hasAGoldMembership(c);
```

Where *b* is another concept type and *num* is one of its properties. The dependency set for this rule is *b.num* and *c*.

Testing the Truth of a Rule's Conditions Using the Dependency Set

During a conflict resolution cycle, BusinessEvents tests each rule's dependency set against the new set of facts. If the facts match the rule dependencies, the rule conditions are all true and the rule action is added to the rule action agenda. The structure of the Rete network enables very quick matching between facts and rule dependency sets.

If BusinessEvents cannot calculate dependencies on the properties of an entity from the rule condition, for example if you pass an entity to a function, BusinessEvents evaluates the rule every time the entity or its properties changes.

How a Rule Becomes Newly True

A rule is true if objects in the rule scope exist in the Rete network and if all of the rule conditions are met by objects in the Rete network. However when building the rule action agenda, BusinessEvents examines only rules that are *newly* true.

A rule is *newly true* if it has become true due to a change in the Rete network during the current RTC.

In the case of a rule with no conditions, assertion of an object in the scope (declaration) of the rule makes the rule newly true.

A rule that was false and becomes true because of the changes in the Rete network during the RTC is newly true.



Less obviously, a rule that was already true can also become newly true. For example, a rule may already be true because a condition that specifies a range is satisfied. It becomes newly true if the property value in the Rete network changes but is still within the range. For example, the condition `c.b < 10`; is true if the Rete network includes a `c.b` with value 9. It is newly true if an action at the end of a conflict resolution cycle changes the value from 9 to 8.

A rule remains newly true until it is executed or it is removed from the agenda, or the RTC ends.

A rule is removed from the agenda because a change in the Rete network during an RTC means that the facts no longer satisfy its dependency set, for example because a concept instance is deleted or a concept property changes value.

Order of Evaluation of Rule Conditions

The order in which conditions are evaluated is determined internally by BusinessEvents. Using a rule's dependency set, BusinessEvents evaluates the following kinds of rule conditions in the order shown, to perform the evaluation efficiently.

1. **Filters**, that is, conditions that only involve one scope element (object). Filters are the least expensive operations, in terms of processing cost. For example:

```
Customer.type = "gold";
Customer.numOrders > 50;
```


2. **Equivalent join conditions**, that is, conditions that compare two expressions using `==` where each expression involves one (different) object from the scope. Equivalent joins take more processing than filters, but less than non-equivalent joins. For example:

```
Customer.accountMgr == AccountManager.id;
```

3. **Non-equivalent join conditions**, that is, conditions involving two or more scope elements other than equivalent joins. These are done last because they are the most expensive in terms of processing cost. For example:

```
Customer.numOrders < AccountManager.threshold;  
MyFunctions.match(Customer, AccountManager);
```



To optimize performance, do as much filtering as possible, to reduce the number of times BusinessEvents evaluates a join condition.

Enforcing the Order of Condition Evaluation

To enforce the order of evaluation between two or more conditions, put them on the same line (that is, in one statement ending in a semicolon) joined by the logical operator `&&`.

Be aware of some differences in execution when you combine conditions. For example, consider the following separate conditions. A null pointer exception might be thrown if `concept.containedConcept` is null, if the second condition was checked before the first:

```
concept.containedConcept != null;  
concept.containedConcept.property == "test";
```

You can, however, combine the conditions as follows:

```
concept.containedConcept != null && concept.containedConcept.property == "test";
```

In this case, a null pointer exception is not thrown when `concept.containedConcept` is null because it is always checked first.

Chapter 6

Object Management Options

This chapter introduces object management and fault tolerance options in BusinessEvents so you can select the appropriate options for your needs.

Topics

- [*Object Management \(OM\) Overview, page 58*](#)
- [*Berkeley DB Object Manager, page 62*](#)
- [*Object Management and Fault Tolerance Scenarios, page 64*](#)

Object Management (OM) Overview

Object management refers to various ways that BusinessEvents can manage the ontology object instances created by BusinessEvents.

Use of the Cache manager enables rich functionality and is generally chosen for enterprise applications. The In Memory object manager can also play a useful secondary role in testing, and as an event router.



Note that you can't mix object managers in one BusinessEvents application.

The goals of object management are as follows:

- **Object Persistence** Enables objects to be available for reuse, either in memory caches or in databases. Objects can also be recalled into the Rete network, thus extending the possible functionality of your system.
- **Data Recovery** Ability to survive failures without loss of data.
- **Object Partitioning** The ability to partition the objects among multiple JVMs. and to handle notifications of object additions, deletions, and changes to all the agents, enabling them to remain synchronized
- **Object Clustering** The ability to maintain multiple copies of each object in different nodes (JVMs) such that if one node fails, another node can take over (backup-count).
- **Message Acknowledgment** See [Message Acknowledgment on page 21](#) for information on the way each object management option handles message acknowledgment.



The Cache Object Manager

In order to understand these options you should understand these terms:

Processing Unit: A processing unit deploys as a BusinessEvents engine. One engine runs in one JVM.

Agent: Each processing unit contains one or more agents of different types. The main types are inference agents, which perform the inferencing work, and cache agents, which manage the objects.

Using cache clustering technology, object data is kept in memory caches, with redundant storage of each object for reliability and high availability. Within a cache cluster, processing units deployed as cache agents manage the data and handle recovery. Cache data is shared by all agents in the cluster.

Recovery from total failure is available if you implement a persistent backing store. Recovery from failure of individual processing units (JVMs) is available without a backing store. Optional cache management features (modes) provide fine-grained controls for managing the memory footprint.

Fault tolerance is provided at the inference agent level. Agents belonging to the same agent class can act in a traditional fault tolerant manner, where active agents take over for inactive agents. Fault tolerance can also be provided implicitly, because all active agents in the same class share the workload. There may be no need to keep any agents inactive. It depends on your needs.

Cache-based object management is generally the best choice for CEP systems. It offers richer functionality, and is the method that receives most focus in these chapters.



Cache-Related Features Cache-based object management enables additional features, such as features for querying the cache (additional querying features are available with TIBCO BusinessEvents Event Stream Processing add-on software), and concurrency features. Rules can also take advantage of the availability of persisted data. Load balancing of messages from a queue is also available.

Cache OM offers finer-grained object management options, at the object level. See [Chapter 7, Distributed Cache OM, on page 69](#).

For implementation details, see *TIBCO BusinessEvents Administration*.

The In Memory Object Manager

The In Memory object manager does not persist object instances. They are maintained in local JVM memory only. Objects are managed by standard JVM features. This is the only section on In Memory manger, because of its simplicity.

In Memory OM does not provide data recovery in case of system failure. The working memory on each system is not synchronized. Object state is not maintained. At startup after a failure, object state is initialized to the application's starting state.

The In Memory option is a good choice for development and testing environments. In production environments, the In Memory option is best used for stateless operations and transient objects. An independently deployed In Memory application can act as an event router, directing events to agents in a cache cluster for processing.



For Fault Tolerance If you require fault tolerance with an in memory only system, then configure for Cache OM, but use the Memory Only mode for all objects. (In Memory OM itself does not support fault tolerance.) Because data is not persisted, it is lost during failover and failback. However, the engine process continues.

Another advantage of this approach is that the in memory processing units can participate in the larger cluster, instead of being a separately deployed application.

The Berkeley DB (Persistence) Object Manager



- Do not confuse Berkeley DB object manager with cache plus backing store object management. Both use a database, but are otherwise quite different.
- Berkeley DB object manager is deprecated in BusinessEvents 4.0.0.

Object data is periodically written to a data store on disk. Each agent has its own data store. This option enables recovery of objects from the persisted state, but does not support built-in fault tolerance mechanisms. Custom means can be used to provide fault tolerance.

Summary of Object Management Features

The following table illustrates which features are supported for each object management option.

OM Option	Persistence	Data Recovery	Partitioning	Clustering	Fault Tolerance
In Memory	No	No	No	No	No (Use Cache with Memory Only objects)
Berkeley DB	Yes	Yes (snapshot)	No	No	(Custom)

OM Option	Persistence	Data Recovery	Partitioning	Clustering	Fault Tolerance
Cache	Yes	Yes	Yes	Yes	Yes (at agent level)

Migrating to a Different Object Management Method

You can use In Memory object management in early phases of development. In later phases, you can implement Cache OM and take advantage of features it makes possible.

You can also migrate a production system from Persistence to Cache. See *TIBCO BusinessEvents Installation*, Chapter 6, Migrating Persistence Data to Backing Store.



Perform tests after changing object management method As with any change in configuration, be sure to perform thorough testing before going into production.

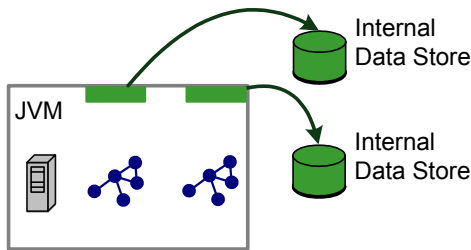
Berkeley DB Object Manager




Berkeley DB OM (formerly known as Persistence-Based OM) is deprecated in this release. Do not use this OM in new projects. Instead use Cache Manager with backing store. Follow procedures in *TIBCO BusinessEvents Installation* to migrate data. You must of course configure the other features of Cache Manager as well.


As illustrated in the figure on this page, the Berkeley DB object management option persists a snapshot of the working memory for each inference agent (rule session) in the deployed system. The data for each inference agent is persisted to a data store at specified intervals.


A small cache for each inference agent ensures that currently used objects are available in memory for improved performance. You can control the size of the cache (see *Caches Used for Persistence-Based Object Management in TIBCO BusinessEvents Administration*).



Legend

 BusinessEvents Server (Engine)

 Rete Network (Working Memory – Rule Session)

 Persistence Data Cache

The persistence data store uses Berkeley DB and is provided and managed by BusinessEvents.

Persistence-based object management provides data recovery in the case of a complete system failure. When the system comes up after a system failure, BusinessEvents restores the working memory (or memories) to the last checkpoint state. It also receives all of the previously unacknowledged messages.

Data in memory at time of failure and not yet written to disk is lost.

Use of the Berkeley DB option affects performance, due to the disk writes required. BusinessEvents provides parameters — checkpoint interval and property cache size — to help you tune performance. You can also determine how many objects to keep in the data cache, in order to manage JVM memory usage for the application for better performance.



Memory Usage Tips

- The persistence database can be used purely as virtual memory. You can disable recovery features if you don't need them.
- You can tune memory usage by setting the number of properties and number of events to keep in JVM memory. You can also set aside a percentage of JVM memory for use by the persistence layer.
- For services that need a lot of memory, consider running each rule session (inference agent) in a separate engine.

See Cluster Tab — Berkeley DB Manager Settings and Properties in *TIBCO BusinessEvents Administration* for details.

Fault Tolerance With Berkeley DB Manager

Fault tolerance features for Berkeley DB object management are not provided by BusinessEvents. You can, however, implement a custom fault tolerance solution using TIBCO Rendezvous and TIBCO Hawk or third-party fault-tolerance tools. For example, you could set up two servers that each point to the same persistence data store, and you could write rules in your fault-tolerance tool to detect failure and take appropriate steps (for example, removing any lock files) when failing over to the secondary server.



If you will provide a custom fault tolerance solution, do not enable any built-in BusinessEvents fault tolerance features. They are not used with your custom solution.

Object Management and Fault Tolerance Scenarios

The tables in this section help you understand how fault tolerance and object management options work in various deployment scenarios to maintain data integrity. The tables explain what is possible in each type of object management given the following conditions:

Processing Units (PUs) One or multiple PUs, where a PU is a BusinessEvents server running in one JVM.

Agents One or multiple inference agents running in a PU. Each inference agent is configured by an agent class in the CDD. An inference agent has one or more Rete networks. See [Designing for Concurrency on page 96](#) for related details.

When implementing a recovery strategy you must take care to maintain the integrity of stateful objects. Concepts and scorecards are stateful objects and must maintain state across inference agents. Not all options provide that option.

Cache OM with Memory Only Mode on All Objects and Fault Tolerance Scenarios



In Memory object management does not support fault tolerance. This table presents options available if you use Cache OM with Memory Only mode set on all objects, which provides fault tolerance for memory only objects.

Table 4 Cache OM with Memory Only Mode on All Objects and Fault Tolerance Scenarios

# PUs # Agents	With Fault Tolerance Configuration	No Fault Tolerance Configuration
1 PU 1 Agent	(N/A)	Data is isolated to a single PU (JVM). No recovery.
1 PU <i>n</i> Agents	(N/A)	No recovery.
<i>n</i> PUs 1 Agent	Data is isolated in each PU. Failover and failback are allowed. Object state is not preserved or transferred. Recommended only for stateless operations.	Data is isolated to each PU. No recovery.

Table 4 Cache OM with Memory Only Mode on All Objects and Fault Tolerance Scenarios (Cont'd)

# PUs # Agents	With Fault Tolerance Configuration	No Fault Tolerance Configuration
n PUs	Data is isolated in each multi-agent PU.	No recovery.
n Agents	Object state is not maintained during failover and failback. Recommended only for stateless operations.	

Berkeley DB Object Management and Fault Tolerance Scenarios



Fault Tolerance with Persistence-Based Object Management As explained in the table below, the BusinessEvents built-in fault tolerance feature is not supported for use with persistence-based object management. You can implement a custom solution, however.

Table 5 Persistence and Fault Tolerance Scenarios

# PUs # Agents	With Fault Tolerance Configuration	No Fault Tolerance Configuration
1 PU 1 Agent	(N/A)	Data is isolated in a single persistence database. On recovery, object state is recovered to the last checkpoint.
1 PU n Agents	(N/A)	In all deployment scenarios, each agent's data is isolated in a separate persistence database. On recovery, object state is recovered to the last checkpoint of the appropriate database.
n PUs 1 Agent	Not supported with BusinessEvents built-in fault tolerance. Automatic failover and failback is not possible due to presence of lock files. Use a custom solution.	
n PUs n Agents	Not supported with BusinessEvents built-in fault tolerance. Automatic failover and failback is not possible due to presence of lock files. Multiple write operations by agents on the primary PU could lead to data inconsistency. Use a custom solution.	

Cache Object Management and Fault Tolerance Scenarios

In all cases it is assumed that dedicated cache agents are also running. Fault tolerance of the engine process refers to inference agents only. See [Distributed Cache and Multi-Agent Architecture and Terms on page 76](#).

If you use multi-engine (multi-agent) features, fault tolerance is implicit. When all agents in an agent group are active, if any active agent fails, remaining agents in the group automatically handle the work load.

In all cases, in the event of total system failure, use of a backing store ensures recovery of data written to the backing store.

Table 6 Cache and Fault Tolerance Scenarios

# PUs # Agents	With Fault Tolerance Configuration	No Fault Tolerance Configuration
1 PU 1 Agent	(N/A)	(N/A)
1 PU <i>n</i> Agents	(N/A) Each agent in the same PU is a different agent, not part of the same agent group.	(N/A)

Table 6 Cache and Fault Tolerance Scenarios (Cont'd)

# PUs # Agents	With Fault Tolerance Configuration	No Fault Tolerance Configuration
n PUs 1 Agent	<p>Fault tolerance is at the agent level.</p> <p>Multi-agent mode: If one or more agents in a group fails, the load is distributed among remaining agents in that group. All agents can be active or some can be inactive. Configuration uses a <code>MaxActive</code> property and a <code>Priority</code> property.</p> <p>Single-engine mode (Deprecated feature): Priority setting determines which agent in an agent group is active, as well as the failover and failback order.</p> <p>Cluster data is shared between agents in all groups across all PUs, using the cache cluster.</p> <p>If the number of cache object backups is one, one cache agent (at a time) can fail with no data loss. If the number of backups is two, two servers can fail, and so on.</p> <p>Because caches exist in memory only, recovery is not available in the case of a total system failure. All data in each JVM memory is lost in a total system failure.</p> <p>In the event of total system failure, use of a backing store ensures recovery of data written to the backing store.</p>	<p>Multi-agent mode: N/A. Fault tolerance is implicit.</p> <p>Single-engine mode (deprecated feature): N/A</p>
n PUs n Agents	Same as n PUs 1 agent. Each of the agents in one PU is fault tolerant with the agents in the same agent group, which are deployed in other PUs.	Multi-agent mode: N/A. Fault tolerance is implicit.

Chapter 7

Distributed Cache OM

Cache object management (OM) is the standard choice for most BusinessEvents deployments. Distributed cache architecture has been chosen as the most appropriate for BusinessEvents.

This chapter provides an overview of Cache OM and concurrency features. Concurrency can be achieved using multi-agent features or concurrent Rete features or both.

Concurrent Rete does not require Cache OM, but multi-agent features do.

More detailed information is provided in the chapters following.

Topics

- [Cache Object Management Feature Overview, page 70](#)
- [Characteristics of Distributed Caching Schemes, page 73](#)
- [Distributed Cache and Multi-Agent Architecture and Terms, page 76](#)
- [Cache Cluster Discovery, page 80](#)
- [Load Balancing and Fault Tolerance of Inference Agents, page 82](#)
- [Cache OM with a Backing Store, page 84](#)
- [Cache Manager Options at the Entity Level, page 86](#)

Cache Object Management Feature Overview

Cache-based object management is generally the best choice for a CEP system, and a distributed cache is generally the most appropriate, especially when used with a backing store (database). All the provided caching schemes use a distributed cache and are configured for production as shipped.

For configuration of other caching schemes, and for advanced configuration of the provided schemes, consult the *TIBCO BusinessEvents Cache Configuration Guide* online reference.

Cache OM is a requirement for other features such as multi-agent and concurrent Rete features (as explained in [Chapter 9, Concurrency and Project Design, on page 95](#)).

Distributed Cache Characteristics

In a distributed cache, cached object data is partitioned between the PUs (JVMs) in the cache cluster for efficient use of memory. By default one backup of each item of data is maintained, on a different PU. You can configure more backups of each object to be kept on different PUs to provide more reliability as desired, or to disable maintenance of backups.

Distributed caching offers a good balance between memory management, performance, high availability and reliability. It also offers excellent system scaling as data needs grow. See [Characteristics of Distributed Caching Schemes on page 73](#) for more details.

Scaling the System

To scale the system's capacity to handle more data, add more cache agents, which are PUs specialized to handle cache data only (see [Cache Agents \(Storage Nodes\) on page 78](#)).

To scale the systems capacity to process more data, add more inference agents (see [Inference Agents on page 77](#)).

In addition, each entity can have a different cache mode, to help you balance memory usage and performance (see [Between Rete Network and Cache: Cache Modes on page 87](#)).

Reliability of Cache Object Management

When you use Cache object management without a backing store, objects are persisted in memory only, and reliability comes from maintaining backup copies of cached objects in memory caches.

To provide increased reliability in the case of a total system failure, add a backing store.

See [Characteristics of Distributed Caching Schemes on page 73](#) for more details.

Concurrency — Multi-Agent and Concurrent Rete Features

When you use cache object management (generally with a backing store) then you can also use multi-agent features.

You can also use concurrent Rete with any OM option.

Multi-Agent Concurrency

Multiple inference agents can run concurrently in either of two ways. In both cases the agents share the same ontology and same cache cluster:

- Multiple instances of the same inference agent, each running on different PUs, form an *agent group*. This provides load balancing of messages arriving from queues, as well as fault tolerance.
- Different agents in different PUs work concurrently to distribute the load on the JVM processes. This results in quicker conflict resolution and the ability to handle a heavy incoming message load. For example, Agent X connects to Agents Y and Z to create rule chaining across a set of PUs. Each agent uses different sets of rules, such as rules for fraud, upsell and cross-sell. All agents operate against the same cluster and share the same ontology. The output from one agent may trigger rules deployed in another agent, causing forward chaining of the work load

Concurrent Rete

Another way to achieve concurrency is to use the multi-threaded Rete feature. This feature can be used with or without cache. For example, suppose one application routes messages to multiple other agents in other applications. It might use In Memory object management, because there is no need to persist or reuse data, and also use concurrent Rete for high performance.

Concurrency and Locking

With agent or RTC concurrency, you must use locking: in both cases multiple RTCs are being processed at the same time, and data must be protected as in any concurrent system. See [Designing for Concurrency on page 96](#) for more details.

Where Object Management is Configured

Object management is configured using the Cluster Deployment Descriptor, an XML file that you edit in BusinessEvents Studio using a provided editor.

See Chapter 2, CDD Configuration Procedures and Chapter 3, Cluster Deployment Descriptor Reference in *TIBCO BusinessEvents Administration* for details.

Characteristics of Distributed Caching Schemes

The cache characteristics are defined by a caching scheme. The provided caching schemes are all distributed caching schemes, and the appropriate scheme is chosen internally based on configuration choices. This section explains in some more detail the advantages of a distributed scheme over a replicated scheme, in which all data is replicated in all JVMs.

In a distributed cache, cached object data is partitioned between the storage PUs in the cache cluster for efficient use of memory. This means that no two storage PUs are responsible for the same item of data. A distributed caching scheme has the following characteristics:

- Data is written to the cache and to one backup on a different JVM (or to more than one backup copy, depending on configuration). Therefore, memory usage and write performance are better than in a replicated cache scheme. There is a slight performance penalty because modifications to the cache are not considered complete until all backups have acknowledged receipt of the modification. The benefit is that data consistency is assured.



Each piece of data is managed by only one cluster node, so data access over the network is a "single-hop" operation. This type of access is extremely scalable, because it can use point-to-point communication and take advantage of a switched network.

- Read access is slightly slower than with replicated cache because data is not local. The cache is distributed between the nodes.
- Data is distributed evenly across the JVMs, so the responsibility for managing the data is automatically load-balanced across the cluster. The physical location of each cache is transparent to services (so, for example, API developers don't need to be concerned about cache location).
- You can add more cache agents as needed for easy scaling.
- The system can scale in a linear manner. No two servers (JVMs) are responsible for the same piece of cached data, so the size of the cache and the processing power associated with the management of the cache can grow linearly as the cluster grows.

Overall, the distributed cache system is the best option for systems with a large data footprint in memory.

Failover and Failback of Distributed Cache Data



It is not necessary to use fault tolerance for cache agents: the cluster transparently handles failover of data to other cache agents if one cache agent fails.

The object manager handles failover of the cache data on a failed cache agent and it handles failback when the agent recovers.

When a storage node (that is a node hosting a cache agent) fails, the object manager redistributes objects among the remaining storage nodes using backup copies, if the remaining number of cache nodes are sufficient to provide the number of backups, and if they have sufficient memory to handle the additional load.

However, because this is a memory-based system, note that if one node fails, and then another cache node fails before the data can be redistributed, data loss may occur. To avoid this issue, use a backing store.

If redistribution is successful, the complete cache of all objects, plus the specified number of backups, is restored. When the failed node starts again, the object management layer again redistributes cache data.

Specifically, when a cache agent JVM fails, the cache agent that maintains the backup of the failed JVM's cache data takes over primary responsibility for that data. If two backup copies are specified, then the cache agent responsible for the second backup copy is promoted to primary backup. Additional backup copies are made according to the configuration requirements. When a new cache agent comes up, data is again redistributed across the cluster to make use of this new cache agent.

Because they store data in memory, cache-based systems are reliable only to the extent that enough cache agents with sufficient memory are available to hold the objects. If one cache agent fails, objects are redistributed to the remaining cache agents, if they have enough memory. You can safely say that if backup count is one, then one cache agent can fail without risk of data loss. In the case of a total system failure, however, the cache is lost.

Limited and Unlimited Cache Size

The size of a cache can be unlimited or limited. Limited cache size is used when a backing store is available to hold the evicted cache entries. (see Specifying Limited Cache Size in *TIBCO BusinessEvents Administration*).

Performance is best when all the data is in cache. But if the amount of data exceeds the amount of memory available in the cache machines, you must limit the cache size and use a backing store to store additional data. Depending on the application needs, you can use the backing store as the main storage and retrieve objects from the backing store as needed.

Eviction Policy

With a limited cache, objects are evicted from the cache when the number of entries exceeds the limit. The default eviction policy is a hybrid policy:

Hybrid eviction policy chooses which entries to evict based the combination (weighted score) of how often and recently they were accessed, evicting those that are accessed least frequently and were not accessed for the longest period first.

The evicted objects are transparently loaded from the backing store when needed by agents.



Only use an unlimited cache if you deploy enough cache agents to handle the data. Otherwise out of memory errors may occur.

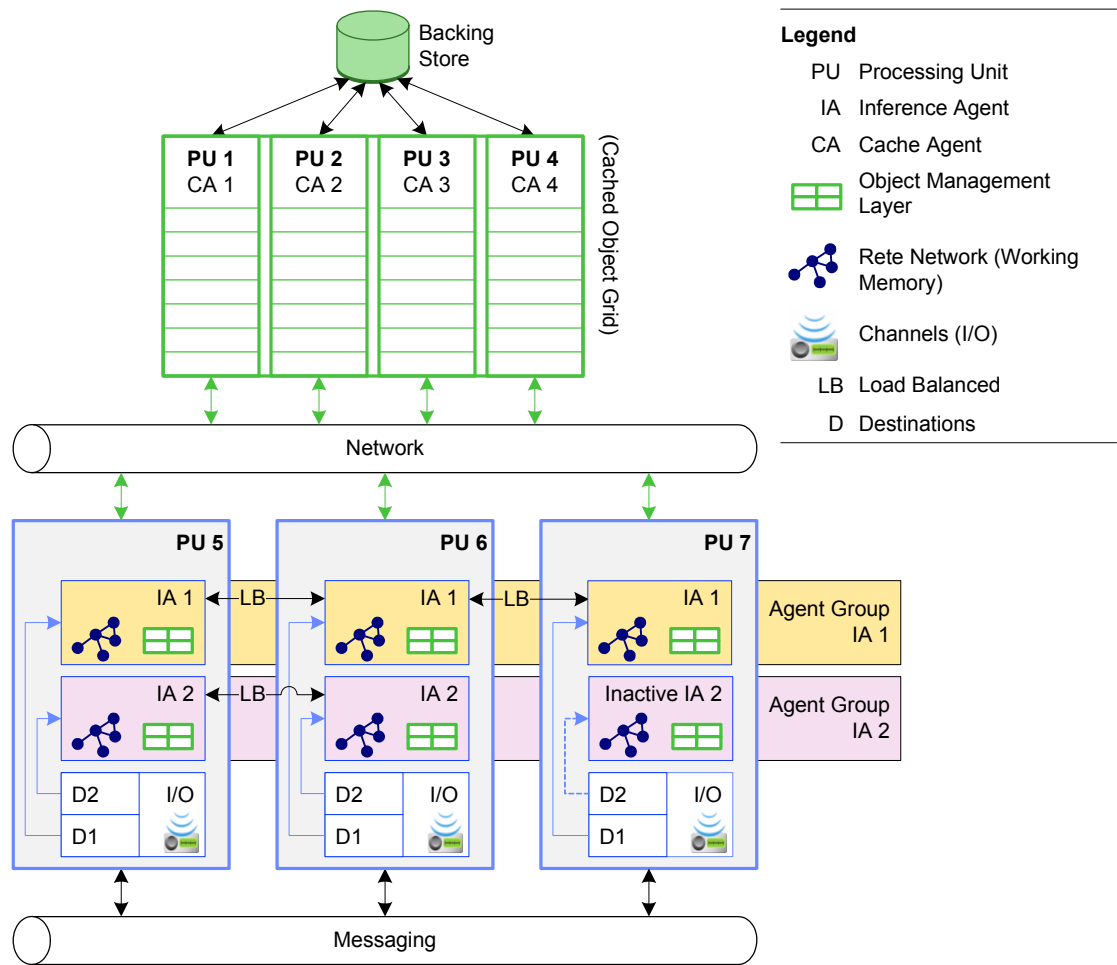
Only use a limited cache size if you have configured a backing store. Evicted objects are lost if there is no backing store.

For backing store configuration, see Chapter 13, JDBC Backing Store Configuration in *TIBCO BusinessEvents Administration*.

Distributed Cache and Multi-Agent Architecture and Terms

The drawing below illustrates a typical Cache OM architecture.

Figure 6 Cache Object Management and Fault Tolerance Architecture



Different BusinessEvents processing units and agents within the processing units have specialized roles in a Cache OM architecture. Sections below provide more detail.

The drawing illustrates one possible configuration, and assumes destinations that are JMS queues, for load balancing.

Agent group IA1 has three active agents, and agent group IA2 has two load balanced agents and one inactive agent for fault tolerance. Each agent group is listening on a different destination. See [Load Balancing and Fault Tolerance of Inference Agents on page 82](#) for details.

Cache Clusters

A cache cluster is a logical entity that provides the following services:

- **Cache Management:** Partitioning, replication, distribution and failure recovery (see [Reliability of Cache Object Management on page 71](#)).
- **Fault Tolerance (of data):** Notifications to inference agents so that the state of each agent's working memory remains synchronized with the others, so any agent in the cluster can take over in event of a JVM failure.

You define the cluster member machines, processing units, and agents in the Cluster Deployment Descriptor (CDD) which is an XML file, configured in the CDD editor in BusinessEvents Studio. See CDD Configuration Procedures in *TIBCO BusinessEvents Administration*.

Cache Cluster Processing Units (Nodes)

Each processing unit in a cache cluster runs in an instance of a Java virtual machine (JVM). It hosts one or more BusinessEvents agents. It is also known as a node in the *TIBCO BusinessEvents Cache Configuration Guide*.

Inference Agents

An *inference agent* executes rules according to the rule agenda created using the Rete network. In Cache OM systems, inference agents are connected to the cache cluster, enabling fault tolerance of engine processes and cache data, as well as load balancing (with queues).

At design time, you configure an inference agent *class* with a selection of rules from the project, and a selection of destinations, and, as needed, a selection of shutdown and startup functions. Instances of an agent class deploy as an *agent group*. Each agent in a group runs in a different processing unit, concurrently, to enable load balancing (when events come from a queue) and fault tolerance.

For more information see the following sections:

- [Load Balancing and Fault Tolerance of Inference Agents on page 82](#)
- [Designing for Concurrency on page 96](#)

Cache Agents (Storage Nodes)

The purpose of cache agents is to store and serve cache data for the cluster. Dedicated cache agent PUs are non-reasoning agents used as storage nodes only (one per PU). Cache agents are responsible for object management. They participate in distribution, partitioning and storage of the objects in the cluster.



Other Agent Nodes Functioning as Cache Agents It is possible, but not recommended, to enable inference and query agent nodes to act as cache agents in addition to their other functions. Using dedicated cache agent nodes for data storage is more efficient and more scalable for production scenarios. Enabling storage on a different kind of agent can be convenient during testing.

When a backing store is used, you can balance what objects to keep in the cache and what to keep in the backing store, until needed.

Memory and Heap Size Guideline for Cache Servers

The amount of memory you need for cache servers depends on factors such as how many objects you have, their object management configuration, and whether you are using limited or unlimited cache.

You must find an appropriate balance for your projects between too little memory, which leads to too much time spent in garbage collection, and too much memory, which leads to longer garbage collection cycles. The optimal heap size depends on factors such as how much data is kept in each cache server, how many cache servers are used, and whether the cache is limited or unlimited.

For example, if you use a JVM heap size of 1024 MB (1GB), in order to minimize the impact of garbage collection, about 75% of the heap can be used to store cache items, which means about 768 MB per heap. The other 25% is then available for garbage collection activities.

Different operating systems have different requirements so make adjustments as required.

Query Agents

Query agents are available only with TIBCO BusinessEvents Event Stream Processing add-on software. Query agents use an SQL-like query language. You can query data that is in the cache. You can also query data arriving in events, known as event stream processing or ESP.

A query agent is a non-reasoning agent. It has read-only access to the underlying objects in the cache cluster. A query agent can execute rule functions, but not rules. You can mix query agents and inference agents within one node as desired.

TIBCO BusinessEvents Event Stream Processing Query Developer's Guide explains how to work with the query language.

Dashboard Agents

Dashboard agents are available only with TIBCO BusinessEvents Views add-on software. They are similar to a query agent in that their role is to generate information based on queries. The information is made available to the BusinessEvents Views dashboard. See *TIBCO BusinessEvents Views Developer's Guide* for details.

Cache Cluster Discovery

This section explains how the members of the cache cluster can be defined. In many cases, the default multicast values in the provided engine property files work out of the box, with no configuration. Two methods are available: multicast, and well-known-addresses.

Configuration instructions are provided in *Configuring a Cache Manager Cluster — Cluster Tab* in *TIBCO BusinessEvents Administration*



When discussing cache clusters, BusinessEvents *processing units* are also referred to as *nodes*. The term node is used in the *TIBCO BusinessEvents Cache Configuration Guide* and in this section.

Cluster Member Discovery Using Multicast Discovery

With multicast cache cluster configuration, the cluster membership is established using the multicast IP address and port. When a BusinessEvents node subscribes to this multicast IP address it broadcasts information about its presence to the address.

Multicast discovery is used by default. The default values provided mean you may not have to configure any discovery-related properties. However, if you deploy multiple BusinessEvents projects in your environment, you must specify different multicast address and port settings for each project.

You can use multicast when instances of an application are deployed to hosts in the same subnet, or across subnets when broadcast is enabled between the subnets.

Adding, Removing, and Moving Nodes

The object management layer uses multicast to discover new nodes and add them to the cache cluster. Similarly when nodes are removed or moved to a different server, the multicast protocol ensures that members are kept current without any additional configuration.

For the above reason, multicast discovery requires less maintenance than the well-known-address method, which requires configuration updates for all such changes.

Network Traffic

When multicast discovery is used, network traffic is generated by the following activities:

Cluster heartbeat The most senior member in the cluster issues a periodic heartbeat using multicast. The rate is configurable and defaults to one per second.

Message delivery Messages intended for multiple cluster members are often sent using multicast, instead of unicasting the message one time to each member.

For peer-to-peer communications and data transfer, the object management layer then uses unicast as needed. Peer-to-peer connections are automatically established with other servers listening on the multicast address.

Cluster Member Discovery Using Well-Known-Addresses

Well-known-address configuration uses direct member-to-member (point-to-point) communication, including messages, asynchronous acknowledgements (ACKs), asynchronous negative acknowledgements (NACKs) and peer-to-peer heartbeats. The addresses are "well known" to the extent that they are known to each server in the cluster.

Use well-known-addresses instead of multicast to define the cluster participants when multicast is undesirable or unavailable, for example, if nodes are deployed to different subnets and broadcast between the subnets is not enabled.



Specifying one or more well-known addresses disables all multicast communication.

Adding, Removing, and Moving Nodes

You must ensure that the well known address information is updated on all machines to account for system changes, such as adding, removing, and moving nodes to different machines.

If a machine changes name or IP address, the cluster continues to work correctly until the next time you restart that machine.

Discovery When Host Machines Have Multiple Network Cards

If a host machine has multiple network cards, you must specify which IP address and port to bind to. To do this you use `localhost` and `localport` properties.

Do this whether you are using multicast or well-known-addresses to define the cache cluster.

Load Balancing and Fault Tolerance of Inference Agents

This section assumes multi-engine (now termed multi-agent) features are enabled. They are enabled by default and the single-engine mode is deprecated.



Cache agents and query agents do not need or use fault tolerance features. Query agents do not maintain stateful objects and don't require fault tolerance. Fault tolerance of cache agents is handled transparently by the object management layer. For fault tolerance of cache data, the only configuration task is to define the number of backups you want to keep, and to provide sufficient storage capacity. Use of a backing store is recommended for better reliability (see [Reliability of Cache Object Management on page 71](#)).

Load Balancing of Inference Agents in a Group

Load balancing enables horizontal and vertical scaling. The underlying cluster behaves like a database for all the agents connected to the cluster. Load balancing makes use of point-to-point messaging, such as JMS queues. With point-to-point communication, messages are automatically distributed among the members of an agent group. (You can also use different agents to listen to different queues.)

Every JMS input destination runs in its own JMS Session. This provides good throughput for processing, and less connections (see [Each JMS Input Destination Runs a Session on page 19](#)).

Certain aspects of the design have to be managed by the application. See [Designing for Concurrency on page 96](#) for related information.

Fault Tolerance Between Inference Agents in a Group

All inference agents in an agent group automatically behave in a fault tolerant manner. All load is distributed equally within all active agents in the same group. If any agents fail, the other agents automatically distribute the load between the remaining active agents in the group.

You can optionally start a certain number of agents in a group and keep the rest inactive. If an active agents fails, an inactive agent is automatically activated. For most situations, there is no need to maintain inactive agents.

In single-engine mode (deprecated), only one agent in a group is active at a time. A priority property determines the startup order and the failover and failback order.



Fault Tolerance Limitation

Entities that use Memory Only cache mode are not recoverable in failover or failback situations. (See [Working With Cache Modes on page 90.](#))

Behavior of Inactive Agents

Inactive agents maintain a passive Rete network. They do not listen to events from channels, do not update working memory, and do not do read or write operations on the cache.



Startup rule functions do not execute on failover When an inactive node becomes active, it does not execute startup rule functions.

Cache OM with a Backing Store

You can implement a backing store for use with Cache OM to provide data persistence.

During regular operation, cache data is written to the backing store. On system restart, data in the backing store is restored to the cache cluster.

Implementing a Backing Store

To implement a backing store, you need to provide a supported database product. Scripts are provided to set up the database for your project's ontology. If the ontology changes, scripts help you adapt the backing store accordingly (though some manual work may be required depending on the nature of the changes). Existing backing store data can be preserved.

See Chapter 13, JDBC Backing Store Configuration in *TIBCO BusinessEvents Administration* for more details.

A legacy Oracle-only backing store implementation is also maintained in this release but is deprecated.

Configuring Backing Store Options

Various options are available for configuring the backing store for your needs, and for using it to support your data persistence and access needs, as explained in this chapter. See [Cache Manager Options at the Entity Level on page 86](#) for more information about fine grained controls over data storage and retrieval.

Backing Store Write Options — Cache-aside and Write-behind

Writes to the backing store can be done in either of two ways: *Cache-aside*, in which the inference agent handles all writes simultaneously, and offers transaction control; or *write-behind*, in which the cache agent handles writes to cache then database.

Cache-aside is a later implementation to offer improved controls based on experiences with the earlier write-behind method.

For most systems, especially those with a lot of I/O, including updates and deletes within an RTC, cache-aside is generally recommended.

Cache-aside

With cache-aside database write strategy, inference agents manage writes to the cache, the local L1 cache, and the backing store, simultaneously, in the post RTC phase. (The cache agent reads from the backing store, but does not write to it.)

Cache-aside provides transaction control, making sure transactions, including deletes, are performed following an RTC. It allows batching of Rete transactions (RTCs) and provides thread and queue size controls.

Write-behind

With write-behind database write strategy, the cache management layer handles writes to cache and to the backing store. First it writes data to the cache and then to the backing store.

Writes are managed by the cache agents. For inserts and updates, one write-behind thread is used for each entity type. Deletes are performed by the distributed cache threads (configurable) and they are synchronously deleted from the database.

You can configure a write-delay property to define whether the write is synchronous or asynchronous.

Write operations from multiple writers to the cache are batched.

Write-behind batches the writes during the delay period which increases database call efficiency and minimizes network traffic.

Write-behind does not offer transaction controls, and can be slower than cache-aside.

If enough cache agents fail, the cache management layer won't be able to persist a write that was done previously, resulting in an inconsistent database. (This risk can be minimized by using a short write-behind delay or synchronous writes.)

Cache Manager Options at the Entity Level

With cache-based object management and a backing store, objects created by the event processing application, can be used or kept in three locations:

- The Rete network (JVM memory)
- The cache
- The backing store

You can manage where the object data is kept at the level of the entity type. The best choice depends on how often the object changes, and how often it is accessed. The various options balance the memory and performance characteristics of the system. Different applications have different priorities and it is up to you to choose the options that suit your needs.

Between Cache and Backing Store: Preloading Options and Limited Cache Size

When the system demands an object that exists in backing store but not in cache, the object is automatically loaded from the backing store into the cache, and then into the Rete network. This takes time, but reduces the need to store so much data in the cache, which uses up memory.

You can also specify what to preload into the cache from the backing store at startup.

You can remove items from cache that are not frequently needed, and use the backing store to store them.

See Understanding How Entity Objects are Managed in *TIBCO BusinessEvents Administration*.

Limiting Cache Size

When you use a backing store, you can limit the size of the cache by specifying the cache size. This is helpful for large volumes of data that the available memory cannot hold. When the number of objects in cache reaches the cache size limit, some of the objects are automatically evicted from cache (they are still in the backing store).

See Chapter 2, CDD Configuration Procedures in *TIBCO BusinessEvents Administration*.

Between Rete Network and Cache: Cache Modes

As explained above, you can store less-frequently-used objects in the backing store only, and they are retrieved into the cache as needed. In a similar way, you can keep memory objects in the cache or Rete network as follows:

- **Cache Plus Memory:** In the Rete network as well as in the cache, used for constants and concepts that change infrequently.
- **Cache Only:** Only in the cache, most commonly used mode.
- **Memory Only:** Only in the Rete network, used for objects whose persistence is not important.

These controls are known as cache modes, or simply modes, for short. They apply to individual entity types.

Cache modes allow you to define how to manage the instances of each object type. In a concurrent system, Cache Plus Memory mode is inappropriate for most objects, due to the difficulties involved in keeping all the concurrent processes synchronized. Use that mode only for constants, and concepts that change infrequently.

Small, frequently used, stateless entities can be kept in JVM memory only, for improved performance.

Most typically, Cache Only mode is used, and the objects are retrieved into memory (the Rete network) only when needed for an RTC. Locking is still required as in any concurrent system.

See [Chapter 8, Cache Modes and Project Design, on page 89](#) for more details.

Chapter 8

Cache Modes and Project Design

This chapter explains how use of different cache modes for ontology objects can affect your project design.

Topics

- [Working With Cache Modes, page 90](#)
- [Loading Cache-Only Objects into the Rete Network, page 93](#)

Working With Cache Modes

As explained in [Between Rete Network and Cache: Cache Modes on page 87](#), with Cache OM, you can keep memory objects in the cache or Rete network using the following cache modes:

- Cache Plus Memory: In the Rete network as well as in the cache
- Cache Only: Only in the cache
- Memory Only: Only in the Rete network, depending on your need.

This section describes the cache modes available in more detail, and explains how to use them appropriately.

The Rete network consumes a large amount of the available memory in the JVM. You can use cache modes to tune the performance of your application, and reduce its footprint in memory.



See [Loading Cache-Only Objects into the Rete Network, page 93](#) for important information about how use of the cache only mode affects your project design.

Cache Modes are Set on Individual Entities to Tune Performance

You set cache modes at the level of individual entity types in your project. This fine granularity allows you to tune performance and memory usage based on the size and usage of the concepts, scorecards, and events in your project ontology.

For example, you can use the In Memory Only cache mode to that frequently used stateless entities are kept in memory (and are not cached). Objects kept in memory are highly available to the application.

On the other hand, using Cache Only mode for large and infrequently used entities reduces the memory footprint. However, you must explicitly load them (in rules or rule functions) so they are available to the Rete network.



Don't Mix Memory Only with Cache Modes in Related Concepts

All concepts related by hierarchy, containment, or reference must use either:

- Memory Only mode
- Cache Only mode or Cache Plus Memory mode, or both (but not Memory Only).

See [Concept Relationships on page 34](#) for more details about the relationships between concepts.

Cache Plus Memory — For Constants and Less Changeable Objects

With the cache plus memory setting (Cache + Memory), the entity objects are serialized and are always available in cache. There is one object in the cache (in a logical sense), and any number of references (handles) to that object in each JVM. References to the entities are tracked in the working memory so they can be deserialized and retrieved from the cache when needed by the engine.

The agent's working memory is used to store the Rete network for the loaded rules. The inference agent does not store the actual object. It relies on the object management layer to load the objects on demand from the backing store. For example, consider a rule of this form:

Declaration (Scope): Customer

Condition: Customer.age < 20

If the cache mode is Cache plus Memory (Cache + Memory), then working memory stores handles to all customers, including those whose age is greater than 20. The customer objects for customers whose age is less than 20 are deserialized and retrieved from cache when the rule condition is evaluated, in order to process the rule.

Because a Rete network is so large, the references themselves can consume large amounts of memory. To reduce the memory footprint, it is recommended that you use the Cache Only mode for most entity types.

In addition, the Cache Only mode is more straightforward to use as there is less to synchronize in a multi-agent environment when the Rete network is flushed after each RTC.

Appropriate locking in complex projects that use concurrency features can be difficult with Cache Plus Memory.

In Memory Only — Useful for Stateless Entities

When you select In Memory Only mode for an entity type, instances of that entity are available only in the engine's local JVM memory only. These entities and their properties are not recoverable, or clustered or shared. For this reason, it is recommended that you use this mode for stateless entities only.

This mode is typically used for static reference data that can be created in the rule functions on startup.

In Memory Only mode can also be used for transient utility entities that created and deleted within a single processing, and are not needed across RTC cycles.

Entities configured in this mode do not persist objects to the cluster and correspondingly the objects are not recovered from the cluster.

This cache mode works the same as the In Memory object management option (but is set for individual objects).



Fault Tolerance Limitation

Entities that use In Memory Only cache mode are not recoverable in fault tolerance failover or failback situations.

Cache Only Mode

As with the Cache plus Memory mode, when you choose the Cache Only mode for selected entities, the entity objects are serialized and are always available in cache. The difference is that at the end of the RTC, the objects and their references are removed from the Rete network, thus freeing memory.

The Cache Only mode is stateless between RTCs. You must explicitly load the objects needed by rules in an RTC, and you must ensure proper locking is used (see [Using Locks to Ensure Data Integrity Within and Across Agents on page 101](#)).

Various functions are available for loading the entities into the Rete network. They are generally used in an event preprocessor. See [Loading Cache-Only Objects into the Rete Network, page 93](#) for more details.

Loading Cache-Only Objects into the Rete Network

When you use cache only mode for an entity type, objects of that type behave normally when they are created during an RTC (see [Understanding Conflict Resolution and Run to Completion Cycles on page 51](#) for more details). At the end of an RTC, however, they are removed from the Rete network and written to the cache.



Ensure you use correct locking before loading objects. See [Designing for Concurrency on page 96](#) for details.

Cache Load Functions

You must explicitly load cache-only objects into the Rete network when they will be needed during an RTC, using an appropriate cache load function in the Coherence function category:

```
C_CacheLoadConceptByExtId()
C_CacheLoadConceptsByExtId()
C_CacheLoadConceptById()
C_CacheLoadConceptsById()
C_CacheLoadEventByExtId()
C_CacheLoadEventById()
C_CacheLoadParent()
```

Contained
concepts

The functions that load concepts by ExtID or ID have a parameter to control whether contained concepts are also loaded. The `C_CacheLoadParent()` function, which loads a given concept's parents, has the option to return all parents or the immediate parent. (Parents are concepts related to the given concept by a containment relationship).



Referenced Concepts You must explicitly load all referenced concepts as needed. Only containment relationships can be handled automatically.

Use in
preprocessor

A general best practice is to use these functions in an event preprocessor. Event preprocessors are multithreaded so performance is better. Also, if you load the objects in the preprocessor, then the rules themselves do not have to take account of the need to load the objects during execution. For example, in the preprocessor, you could preload an order concept using an ExtID available in the event as follows:

```
Concepts.Order order =
Coherence.C_CacheLoadConceptByExtId(orderevent.Order_Id, false);
```

Loaded Objects are Not New and Don't Trigger Rules to Fire

The loaded objects do not behave like newly arrived entities. They are not asserted: their presence alone does not trigger rules. They are simply restored to the Rete network. They behave as if they had never been removed. For example, rules do fire if there is a join condition between the entity loaded from cache and another entity that is asserted or modified in the same RTC.

Also if you modify the object that you reloaded, it can trigger the rule.



Limited Use of `getByExtId()` Only use this function to retrieve cache-only objects that have already been loaded into the Rete network by a preprocessor. The `getByExtId()` function does not load the object into the Rete network.

Chapter 9

Concurrency and Project Design

This chapter explains some aspects of concurrency (engine concurrency and concurrent Rete) that affect project design.

Topics

- [Designing for Concurrency, page 96](#)
- [Multi-Agent Features and Constraints, page 97](#)
- [Using Locks to Ensure Data Integrity Within and Across Agents, page 101](#)

Designing for Concurrency

You can use multiple active inference agents to achieve load balancing, scaling, and performance. You can also enable concurrent RTC cycles within one agent, known as the *concurrent Rete* feature. Multi-agent and concurrent Rete features provide concurrent RTC functionality — across agents in the case of multiple agents, and within agents, in the case of concurrent Rete.

It is important to be aware of some design considerations when designing project that take advantage of these concurrency features.

The section [Multi-Agent Features and Constraints on page 97](#) explains more about concurrency and how it affects the way events and objects are processed in a multi-agent configuration, or with concurrent Rete.

The section [Using Locks to Ensure Data Integrity Within and Across Agents on page 101](#) explains how to manage access to objects in a concurrent configuration. As with any concurrent system, care must be taken to ensure that two agents or RTCs do not attempt to update the same instance at the same time, and to ensure that reads return a valid and up-to-date instance of an object.

Multi-Agent Features and Constraints

Multi-agent features can be used in two ways:

- Deployment of instances of the *same* agent, each in a different processing unit, for load balancing and fault tolerance. (A processing unit is one JVM, also known as a node.)
- Deployment of instances of *different* agents, to achieve rule-chaining for high distribution of the work load and high performance.

In both multi-agent cases, the agents are in the same cache cluster and work on the same ontology objects and, to provide performant systems.

Concepts are Shared Across Agents Asynchronously

All concepts are shared between agents in the cluster in an asynchronous manner.

For instance, an Agent X receives an event E, fires a rule R1 that creates a concept C1. An agent Z receives an event E2, fires a rule R2 that joins concept C1 and event E2.

Therefore, there is inherent latency between an object change in an agent, and reception of the notification by the other agents in the cluster.



Because of the asynchronous sharing of objects between agents, ensure that events have a long enough time to live setting so that they do not expire before all actions pertaining to the event are done.

It is recommended that you explicitly consume events when their work is done so that they don't cause any unwanted (unforeseen) actions to occur by their presence in the Rete network.

Scorecards are Local to the Agent

Scorecards are not shared between agents. (This is true in all OM types.) Each inference agent maintains its own set of scorecards and the values in each agent can differ. This enables scorecards to be used for local purposes and minimizes contention between the agents.

As an analogy consider a bank ATM scenario. Money can be drawn from the same account using different ATMs. However, each ATM maintains a "scorecard" indicating only how much money it dispenses.

An agent key property (`Agent . AgentClassName . key`) is available for tracking scorecards. It identifies an agent uniquely so that its scorecard can be restored from the cache.



Do not use scorecards as a mechanism to share data between multiple agents.

Events are Owned by the Agent that Receives Them

In a load balanced group of agents, events (messages) received by an agent are owned by that agent. Even when the event is retrieved from cache (for example for a join condition), the ownership is maintained. No other agent can work with that event, unless the owning agent fails.

Cache cluster services provide for reassignment of ownership of clustered events to other agents in the same group, in the event of node failure, so there is no single point of failure. (Of course, if the event's time-to-live period expires during this transition, the event expires and is not reassigned.)

Events from Queues

Events received from a queue are each taken up by one agent or another. You can set up load balancing of events arriving from a queue among events that share the destination. These are generally events in the same group (class) but don't have to be. For example, when an agent X receives an Event E1 from a queue, agent B in the same agent group does not see the event.

Events from Topics

Messages sent on a topic, however, are received by *all* agents that actively listen to the topic. Each agent generates its own event instance (with its own ID) when receiving the message. While it could be useful for multiple agents to receive events sent on a topic, this often leads to undesirable results. Care must be taken to ensure that just one agent receives topic-based messages.

Event Related Constraints

Repeating Time Events Not Supported

Time events configured to repeat at intervals are not supported in multi-agent configurations. Rule-based time events, however, are supported.

State Machine Timeouts

State machines can be configured to have state timeouts. The agents in the cluster collaborate to take ownership of management of the state machines, thereby providing automatic fault tolerance.

Multi-Agent Example Showing Event Handling

The following example shows how concepts are shared and events are not shared in a load balancing agent group.

Example Scenario Agent A has the following rules:

Rule	Scope	Condition	Action
R1	Event E1	None	Create concept C1
R2	Event E2, Concept C1	E2.x == C1.x;	Send event E3

Agent A listens to destinations on which events E1 and E2 arrive.

You start two instances of agent A called A1 and A2. Both are active, therefore they both listen to the destinations on which events E1 and E2 arrive.

At runtime, the arrival of two events illustrates the expected behavior:

1. Agent A1 receives an instance of Event E1:

1. Rule R1 executes and creates an instance of concept C1.
2. During the post RTC phase, the instance of C1 is written to cache.
3. Event E1 has a Time to Live of 30 seconds. It is acknowledged and then moved to the cache.
4. With Cache Plus Memory mode, Agent A2 receives a notification and loads the instance of concept C1 in its Rete network. With Cache Only mode, Agent A2 has to explicitly load the concept when it will be needed for an RTC.

Note that the event E1 is in the cache, but Agent A2 does not load the event in its Rete network. However, if the node (JVM) containing Agent A1 fails, then Agent A2 moves the pending events to its Rete network.

2. Agent A2 receives an instance of Event E2:

1. Rule R2 executes because agent A2 is aware of the instance of C1. (With Cache Only mode the instance of C1 is in the Rete network only if it has been explicitly loaded.)
2. In the post-RTC phase, Agent A2 sends out event E3.

The effect of cache modes on design is explained in other sections of this chapter.

Using Locks to Ensure Data Integrity Within and Across Agents

Multiple agents can read from and write to the same cache cluster and at times they can operate on the same set of objects. Multiple threads in one agent can also behave in a similar manner, to enable concurrent RTCs.

Locking is used to ensure the data you read is up-to-date, and to ensure that no other RTC is updating the same data concurrently.

Understanding Locking in BusinessEvents

Locking protects the thread of execution when multiple threads in an agent can cause conflicts by trying to write to the same concept at the same time during multi-threaded Rete operations (where multiple RTCs are running concurrently). The same type of issue can occur across inference agents operating concurrently. Locking is also necessary to ensure that stale data – data that has been modified in another RTC but not yet written to cache – is not read.

Locking is one of the necessary costs of tuning inference agents for higher performance through multi-threaded Rete or concurrency across agents.

Lock operations do not operate a lock on the object you want to protect itself. They set a common flag that represents a lock — it is up to the developer to ensure that all accesses and updates to a concept subject to locking are enforced, and that only necessary concepts (concepts that are written to as well as used in conditions) are locked

The goal of locking is to ensure consistency across concurrent RTCs. For example, if one RTC has a rule condition that includes a concept, and another RTC updates that concept, then the results could be undefined (does the condition use the new or old values of the object?). Or if two RTCs update the same object at the same time, then different properties of the object could be set by different threads leaving an overall object with incorrect property values.

The typical lock operation is: in the event preprocessor set the lock, using any unique string as a key. For example, you can use the object extId as the lock string.

If a preprocessor cannot acquire the lock (because another event's preprocessor has acquired it) then it waits until either the lock is released OR some timeout occurs.

Locking code needs to be prepared carefully. If two events try lock(A) then lock(B) and lock(B) then lock(A) respectively, then a situation can arise where both are waiting on each other's thread. Locking should be used sparingly.

With Cache Plus Memory OM, the need for locks is greater than with Cache Only, because each agent's Rete network must be synchronized for changes made in all other agents' Rete networks.



Because of issues around effective locking with Cache Plus Memory in multi-agent scenarios, Cache Only mode is generally recommended. Cache Plus Memory is useful for objects that change infrequently.

When to Use Locking

Depending on your application, locking may not be required in all cases. However it is almost always needed. For most applications, use locking in the following cases:

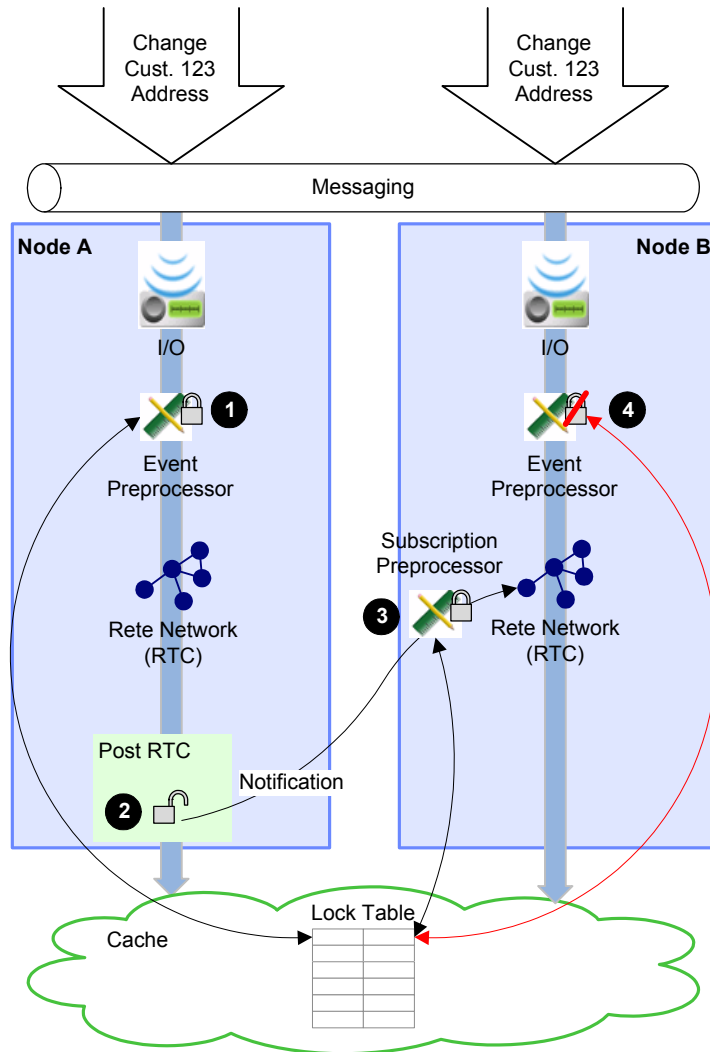
- **With Cache Only mode, for writes.** Locking is done using an event preprocessor.
- **With Cache Plus Memory mode, for writes, and for subscription RTCs.** These RTCs run internally to keep the Rete networks on each agent synchronized with the cache). Locking is done using a subscription preprocessor. It uses the same locking key (string) as in the event preprocessor, if one was used.
- **With any mode, for reads.** If you want to read the latest version of a concept in one agent at the same time that another agent might create or update the same concept, mediate the reads through the same lock that was used when creating or updating the concept. This is done using an event preprocessor and, for Cache Plus Memory, a subscription preprocessor.
- **With state modeler, for timeouts.** State modeler timeouts don't go through an event preprocessor, so locking is done a different way. This is explained in *TIBCO BusinessEvents Data Modeling Developer's Guide*.

Lock Processing Example Flow

The following example demonstrates common locking requirements. It uses these features:

- Cache Plus Memory mode (requires more locking than Cache Only mode)
- Cache-aside and backing store (backing store not shown).

Two agents receive messages that require changes to one Customer instance:



Event
Preprocessor

Note that event preprocessors are multi threaded (see [Event Preprocessors on page 17](#) for more details).

1. A message comes into a channel on Agent A: a change to a customer address. BusinessEvents dequeues the message from the queue, deserializes the message to an event, and calls the event preprocessor function. The preprocessor acquires a lock using the customer's extID as the key:
`Coherence.C_Lock(Customer@extId, -1, false);`

This function causes the thread to stop until it gets the lock. In this example, the thread gets the lock.

2. Only one thread handles the RTC. (Concurrent RTC is not used in this example.) Other event preprocessor threads go into a queue. During the RTC, a rule executes and modifies the customer address. After RTC completes, the post RTC phase begins: the address change is written to L1 cache, the cluster cache, and the backing store in parallel. Messages arising from the RTC are sent.
3. After the post RTC actions are completed, the lock is released.



If the write-behind option is used instead of cache-aside, the lock is released after changes are written to cache.

Subscription
Preprocessor
(Cache Plus
Memory Only)

Subscription preprocessor activities run on a different thread from event preprocessor activities. One subscription task handles all the changes to the objects from one regular RTC and executes as many preprocessors as there are modified objects.

4. Agent A sends a change notification to all other agents that have subscribed to cluster notifications for this object. Agent B receives the notification, and calls the subscription preprocessor function. It contains the same function shown in [step 1](#) above. It uses the same locking string. Agent B acquires the lock (that is the function returns `true`). The agent updates the Rete network with the changes (using a "subscription RTC"). It will release the lock when the subscription RTC is complete.
5. While the subscription update thread holds the lock, Agent B receives a message with a change to the same customer's address and attempts to acquire a lock, but it is blocked (that is, the function returns `false`).

When the subscription preprocessor releases the lock, then Agent B's event preprocessor can acquire it (depending on timeout behavior) and assert the event to the Rete network.

Depending on timing, either an event preprocessor or a subscription preprocessor could be holding the lock.

Locking Functions

The BusinessEvents lock function has the following format:

```
Coherence.C_Lock(String key, long timeout, boolean LocalOnly)
```

If you want to acquire the lock only within the agent, set *LocalOnly* to true. Set the *LocalOnly* parameter to false to acquire a cluster wide lock. For example if you are only concerned about the preprocessor threads in one agent, you can use a local lock.

The thread that calls the `C_Lock()` function is the only thread that gets blocked.

Unlock Function

All the locks acquired during event processing are released automatically after all post RTC actions, cache operations (and database writes in the case of cache-aside mode) are done.

The format of the unlock function is as follows:

```
Coherence.C_UnLock(String key, boolean LocalOnly)
```

You can use the corresponding `C_Unlock()` function for cases where the automatic unlocking does not meet your needs.



The `Coherence.C_Lock()` and `Coherence.C_UnLock()` functions are available in event and subscription preprocessors and in rules. However, in general it is not a good idea to use `C_Lock()` in rules as the order of processing of rules is not guaranteed.

You can call `Coherence.C_Unlock` in a rule only when concurrent Rete is used.

Tips on Using Locks

The example `LockExample` (in `BE_HOME/examples/standard`) demonstrates these points, showing use of locks to prevent race conditions.

- Choose an appropriate key for `C_Lock()`. Note that `C_Lock()` does not lock any entity or object as such. The purpose of `C_Lock()` is to ensure sequential processing of related set of objects, but yet ensure concurrent processing of unrelated objects. For example, you want to process messages related to one customer sequentially across the cluster, but want to process messages for different customers in parallel. In this case you could use the customer ID as the key used for `C_Lock()`. This ensures that all messages for a given customer ID are processed sequentially.
- Do not use unchecked and infinite waits (-1) on the lock. The recommended approach is to use the timeout argument, and then exit with an error.
- Always check the return value of `C_Lock()` and if false, either retry or handle it as an error. Don't let application logic proceed if it returns false. Doing so may result in lost updates or stale reads or other such data inconsistency issues.

- Try to minimize the `C_Lock()`s acquired in one thread. If you have to acquire multiple locks in one thread, ensure that the locks are acquired in the same order of keys, that is, sort the keys. See [Avoiding Deadlock when Multiple Keys Protect One Object](#) on page 106.
- Acquire locks before creating instances, to ensure that no other thread creates the same instance.
- Use `c_lock()` even for read-only operations. If you don't you may get "Inconsistent Database" messages, for example, if there are concurrent deletes elsewhere in other threads or agents.
- In general, Avoid using `C_Lock()` in a rule. Since rule order of execution is not guaranteed it may lead to deadlocks.

Avoiding Deadlock when Multiple Keys Protect One Object

In simplest cases you can use some unique feature of the object you want to protect as the locking key, for example, a customer ID. However different events may point to the same objects using different information. For example, from one channel, the object may be identified using customer ID, and from another, using account ID.

In such cases multiple keys are used to identify the same object. When you acquire a lock to protect such an object, you must first get the other key from your system, sort the keys and take a lock on both keys. Sorting can be implemented using a custom function.

If the ordering of keys is not guaranteed, it may lead to a deadlock in a multi-agent or concurrent Rete (multi-threaded) environment. For this reason, avoid use of `C_Lock()` in a loop, where the intention is to process multiple messages. There are other ways to achieve this, for example, using the `assertEvent_Async()` function.

Diagnosing and Resolving Lock Failures

Instead of throwing an exception after failing to acquire a lock after a few attempts, re-route the event to a special destination that only handles errors (an "error queue"), so you have control over which queue the message goes to.

Write a preprocessor on the "error queue" that does do one of the following for each message:

- Consumes it
- Reports it and then consumes it
- Repairs it and then resends it

For example:

```
System.debugOut("Attempting to lock..");
boolean result = false;
for(int i = 1; i <= 3; i = i + 1){
    result = Coherence.C_Lock("$lock0", 2000, false);

    if(result == false){
        System.debugOut("Lock acquisition '$lock0' failed. Attempts: " + i);
    }
    else{
        System.debugOut("Lock acquisition '$lock0' succeeded. Attempts: " + i);
        break;
    }
}

if(result == false){
    Event.consumeEvent(newevent);
    Event.routeTo(newevent, "/Channel/LockErrorDestination", null);
}
}
```

Logging Lock Statuses

If you suspect a deadlock situation is occurring, you can enable the following property to log lock statuses:

```
be.engine.lockManager.enableRecording=true
```

When this property is set to true, BusinessEvents prints the lock statuses in a log in the local Inference engine. Use of this logging feature can affect performance.

Chapter 10 **Deploying, Monitoring and Managing**

This brief chapter adds some details about deployment and runtime.

Topics

- [Deploy-time Configuration and Deployment, page 110](#)

Deploy-time Configuration and Deployment

The BusinessEvents design-time project is deployed as a BusinessEvents application, which can have multiple engines spanning multiple hosts.

Each engine equates to one processing unit, which runs in one JVM (Java Virtual Machine). One processing unit can host multiple agents, except in the case of a cache agent. There can be only one cache agent per processing unit. Each BusinessEvents agent is a runtime component of the overall application.

In order to deploy these units you create two resources: an EAR file and a cluster deployment descriptor, which is an XML file.

The Enterprise Archive Resource (EAR) is a standard component, used when deploying a BusinessEvents application. The EAR file the project ontology. When you are finished designing the project in BusinessEvents Studio, you simply choose a menu option to build the EAR.

See [Administration Components on page 7](#) for an overview of the cluster deployment descriptor, the site topology editor and file, and the BusinessEvents Monitoring and Management component.

Cluster Deployment Descriptor (CDD)

All methods of deployment require a cluster deployment descriptor. After you define the project ontology and other resources, you configure the agents and processing units that will use those resources at runtime. Different kinds of agents play different roles in a large application: inference agents perform rule evaluation, query agents perform queries, and cache agents are deployed as cache agent nodes when the Cache object management option is used. You can include multiple agents in an engine instance by including multiple BusinessEvents agents within one processing unit.

Configuration an agent involves the following (depending on the type of agent you are configuring):

- Selecting one or more sets of rules
- Selecting destinations
- Selecting event preprocessors for destinations, and thread settings to handle preprocessing more efficiently
- Selecting functions that perform startup and shutdown actions

Also in the CDD, you configure the object manager you have chosen for the deployment. See [Chapter 6, Object Management Options, page 57](#) and [Chapter 7, Distributed Cache OM, page 69](#) for more on object management.

All the properties that in prior releases were listed in the engine TRA files are now consolidated into the CDD file, and the same file is used by every engine at deploy-time: you simply specify which processing unit you want to deploy.

For more information about configuring the CDD see Chapter 2, CDD Configuration Procedures in *TIBCO BusinessEvents Administration*.

Deployment Methods

You can use these three deployment methods:

- At the command-line. You specify the CDD file to use and the processing unit within that CDD file.
- Using BusinessEvents Monitoring and Management. This is the preferred method. BEMM can monitor and manage the units you configure in the topology file. It can also manage units you start in other ways: all it needs is the PID of the JVM running the processing unit.
- Using TIBCO Administrator. TIBCO Administrator belongs to an earlier generation of TIBCO products. Deployment to an Administrator domain has certain limitations in this release: You must call the processing unit `default`, and you must call the CDD file `default.cdd`. However if you have been using this utility in your environment, you can continue to do so.

For more details, see Chapter 8, Deploying a TIBCO BusinessEvents Project in *TIBCO BusinessEvents Administration*.

Hot Deployment

Depending on the changes made to your BusinessEvents project, you may be able to replace an EAR file for a BusinessEvents project with an updated one, without stopping the BusinessEvents engine. This feature is referred to as hot deployment. For more information about the BusinessEvents hot deployment feature, including the project changes that are supported, see Chapter 9, Hot Deployment in *TIBCO BusinessEvents Administration*.

Monitoring and Management

The BusinessEvents Monitoring and Management (MM) component provides a dashboard for deployment and for monitoring the status of deployed BusinessEvents engines. Before you use this component you must configure it to suit your needs. Most configuration is done in the CDD file for the `emonitor` project, the BusinessEvents application that acts as the MM server at runtime. See *TIBCO BusinessEvents Administration* for details.

Index

A

- acknowledgement of messages (events) [21](#)
- actions
 - overview of rule actions [11](#)
- advanced caching options [87](#)
 - and concept relationships [90](#)
 - cache only [92](#)
 - cache plus memory [91](#)
 - in memory only [91](#)
 - understanding [90](#)
- advisory events [10, 22](#)
- all values history policy [32](#)
- architecture
 - components [6](#)
 - runtime [48](#)

B

- BE_HOME [xv](#)

C

- cache modes [90](#)
- cache only (advanced caching option) [92](#)
- cache plus memory (advanced caching option) [91](#)
- cache-based object management
 - advanced caching options [87](#)
 - advanced caching options, understanding [90](#)
 - and message acknowledgement [21](#)
 - fault tolerance scenarios [66](#)
 - overview [58](#)
 - reliability [71](#)
 - See also advanced caching options
 - understanding [70](#)

- changes only history policy [32](#)
- channels [9](#)
 - overview [11](#)
- cluster heartbeat [81](#)
- command-line startup
- complex event processing
 - defined [2](#)
 - requirements for [3](#)
- components [6](#)
- concept model [4](#)
- concept reference [36, 36](#)
- concept relationships [35, 36](#)
 - and advanced caching options [90](#)
 - inheritance [34](#)
- concepts
 - history parameter [30](#)
 - multiple checkbox for concept properties [30](#)
 - overview [10](#)
- conditions
 - overview [11](#)
- conflict resolution [50](#)
- conflict resolution cycles [51](#)
- consumeEvent(event) [21](#)
- contained concept [35, 35](#)
- customer support [xviii](#)

D

- deploy time configuration
 - overview [111](#)
- deployment
 - hot [111, 111](#)
 - overview [110, 111](#)
- destinations [9](#)

E

entities, setting cache mode options for [90](#)

ENV_HOME [xv](#)

event acknowledgement [21](#)

event model [3](#)

events

advisory [10, 22](#)

expiry action [25](#)

simple [10, 22, 22](#)

time [10, 22](#)

time events [23](#)

expiry action [25](#)

F

failover and failback

and distributed cache data [74](#)

fault tolerance

and cache-based object management [66](#)

and in memory object management [64](#)

and persistence-based object management [63, 65](#)

not available for in memory only advanced caching
option [83, 92](#)

functions

overview [11](#)

See also rule functions

temporal [11](#)

types [11](#)

G

garbage collection guidelines [78](#)

H

heap size [78](#)

heartbeat, cluster [81](#)

history parameter

for concept properties [30](#)

history policy [32](#)

ring buffer [31](#)

history policy, effect on rule evaluation [33](#)

hot deployment [111](#)

overview [111](#)

I

in memory object management

and message acknowledgement [21](#)

fault tolerance scenarios [64](#)

in memory only (advanced caching option) [91](#)

inheritance [34, 34](#)

instances

concepts [10](#)

J

JMS

channels [11](#)

K

key metrics. See scorecards

L

local channels [11](#)

locks [104](#)

diagnosing failures of [106](#)

logging statuses of [107](#)

tips on using [105](#)

M

- memory
 - and cache only advanced caching option [92](#)
 - heap size guideline [78](#)
 - tuning tips for persistence object management [63](#)
- memory-based object management. See in memory object management
- message acknowledgement [21](#)
- message acknowledgement with object management [21](#)
- message delivery messages [81](#)
- multi-engine features [71](#)

N

- nodes (JVMs) [64](#)

O

- object management
 - advanced caching options [87, 90](#)
 - and fault tolerance scenarios [64](#)
 - and message acknowledgement [21](#)
 - options introduced [5](#)
 - overview [58](#)
 - See also in memory object management, persistence-based object management and cache-based object management
- object management and fault tolerance scenarios
 - cache-based object management [66](#)
 - in memory object management [64](#)
 - persistence-based object management [65](#)
- ontology functions [11](#)

P

- persistence-based object management
 - and fault tolerance [63](#)
 - and message acknowledgement [21](#)
 - fault tolerance scenarios [65](#)
 - overview [60](#)
 - understanding [62](#)
- preprocessors, locking functions in [104](#)
- property arrays [30](#)

Q

- query language [12](#)

R

- reevaluating rules [33](#)
- Rendezvous. See TIBCO Rendezvous
- Rete algorithm [49](#)
- Rete network [49](#)
- ring buffer [31](#)
- RTC [51](#)
- rule agenda [49](#)
- rule based time events [23](#)
- rule engines
 - conflict resolution cycles [51](#)
- rule evaluation [49](#)
- rule functions [4](#)
 - purpose and usage of [44](#)
 - virtual [44](#)
- rule model [4](#)
- rules
 - actions [11](#)
 - conditions [11](#)
 - overview [11](#)
 - reevaluating [33](#)
 - rule resources [11](#)
- run to completion [50](#)

S

- scorecards
 - overview [10](#)
- sense and respond [2](#)
- simple events [10](#), [22](#), [22](#)
- situation awareness [2](#)
- standard functions [11](#)
- startup. See command-line startup. See also deployment.
- state machines [4](#)
 - See also state modeler
- state modeler
 - overview [12](#)
- stateful rule engine [5](#)
- support, contacting [xviii](#)

T

- technical support [xviii](#)
- temporal functions
 - overview [11](#)
- TIBCO BusinessEvents terminology [9](#)
- TIBCO Rendezvous
 - channels [11](#)
- TIBCO_HOME [xv](#)
- time events [10](#), [22](#), [23](#)
 - rule based [23](#)
 - time-interval based [23](#)
- track and trace [2](#)

V

- virtual rule functions [44](#)