COGILITY
SOFTWARE

Model Deployment & Execution in Cogility Studio

Cogility Studio version 6.0

Document version number 6.0

Cogility Software Corporation
111 N. Market St. #815
San Jose, CA 95113

Printed in the United States of America.

Contents

Chapter 8 Remote Deployment

Chapter 9 Cogility Action Pad

Chapter 12 JMS Message Utilities

Chapter 13 Cogility Web Service Exerciser

Chapter 14 Cogility WatchDog

Preface

Cogility Studio provides the tools to create a model for an enterprise information system, and deploy it as a J2EE application. The Cogility Studio documentation provides support for this endeavor.

Cogility Studio documentation

Cogility Studio comes with several volumes of documentation to help you.

- *Installing and Configuring Cogility Studio* describes the installation and configuration of your application server, database and Cogility Studio.
- *Getting Started with Cogility Studio* is a brief overview of Cogility Studio.
- *Modeling with Cogility Studio* tells you how to build a model-driven enterprise application using Cogility Modeler and associated tools.
- *Using Action Semantics with Cogility Studio* provides a reference to modeling action semantics for use with Cogility Studio.
- *Change Management in Cogility Studio* describes the change management system for models and model artifacts.
- *Model Deployment & Execution in Cogility Studio* is a guide to application monitoring, maintenance and migration, and describes the utilities that you can use to test and monitor your model deployed as a enterprise application.

Several white papers on various topics are also available to further your understanding of enterprise application integration, business process management, model driven architecture and other related topics. See the Cogility website:

http://www.cogility.com.

2

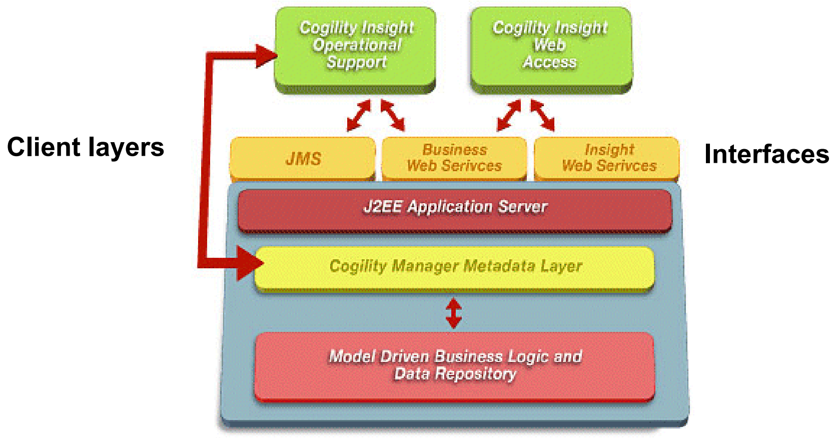
Cogility Manager

Cogility Manager is the run-time environment in which the deployed enterprise application runs. It enables a model created in Cogility Modeler to run as an executble application on a specific J2EE application server with a specific database for the run-time repository. That application runs on the application server and in the Cogility Manager execution environment.

During deployment, Cogility Modeler creates the Java beans, deployment descriptors, routines and other Java run-time utilities. You do not have to write any Java code to complete the deployment because deploying a model not only creates the Java application, but the run-time environment that manages interactions between the deployed application, the application server and the run-time repository.

Execution tools

For working with your model's enterprise application at run time, Cogility Studio includes several tools that work together, from within the J2EE app server as well as directly within Cogility Manager, against the run-time repository. The figure below describes how each of these tools fits within the execution environment.


Cogility Insight Operational Support
Cogility Insight Web Access
Client layers
JMS
Business Web Serivces
Insight Web Services
Interfaces
J2EE Application Server
Cogility Manager Metadata Layer
Model Driven Business Logic and Data Repository

The primary tools, Cogility Action Pad and Cogility Insight work at the client layer indicated by red arrows and run on the local machine. The client interfaces to talk to the application server, these include JMS provided by Cogility Message Traffic Viewer and web services provided by Cogility Web Service Exerciser. Cogility Action Pad and Cogility Insight also provide an interface for web services.

The Cogility Manager metadata layer is an interface to the database server machine. The J2EE Application Server layer is running on the J2EE application server machine, and it has its own copy of the Cogility Manager layer, which it uses to talk to the database.

The model-driven business logic and data repository layer is on the database server. Communication with this layer passes through the Cogility Manager layer. In a sense, the local client as well as the J2EE application server are clients of the database server machine.

Later chapters in this book discuss the execution tools in detail:

- Cogility Insight is web-based and uses a number of prepackaged web services to contact the application server and get metadata and execution data from the underlying run-time repository. For details on using Cogility Insight, see “Cogility Insight” on page 89.
- Cogility Web Service Exerciser can import limited metadata from the run-time repository and call model-defined inbound web services on the J2EE application server, or model-defined outbound (remote) web services from wherever they are hosted. It can also import WSDLs directly and run those remote web services. For more information about Cogility Web Service Exerciser, see “Cogility Web Service Exerciser” on page 127.
- Cogility Message Traffic Viewer can send and receive JMS messages on a specified JMS bus, but is not metadata aware. For details on using Cogility Message Traffic Viewer, see “JMS Message Utilities” on page 119.
- ActionPad is metadata aware and can send messages to a JMS bus, can invoke web services locally, can invoke web services (IWS and OWS) through the App Server interface and execute other logic as required directly against the execution database in a transacted manner. Using orange layer as well as yellow layer, possibly within the same transaction from adjacent instructions in an action pad. This is why in a sense the Action Pad is the public “face” of Cogility Manager, it is a tool that directly uses the Yellow layer. This tool allows you to see how Action Semantics become executable.

3 Configuration

You can configure Cogility Studio from several perspectives: from the modeling perspective with configuration settings that let you customize the user interfaces and set up the authoring repositories, from the project perspective where you assign libraries and other utilities to be used with your model, and from the run-time perspective where you customize the debugging and monitoring facilities of Cogility Manager.

Configuration files

Configuration settings are maintained in text files. The <DCHOME>\MB\Config directory holds the files for the Cogility Studio installation. See “Customizing installation configuration” on page 13. The <DCHOME>\Engagements\<project name>\Config folder holds the files for project-specific configurations. See “Project configuration” on page 14. Depending upon their application, the settings in these files may be referenced when you start Cogility Modeler, when you push a model, or during execution. The settings may be duplicated between installation locations and project locations. The configuration files in all locations are read when a configuration parameter is needed. The last-read configuration setting takes effect. The order of files read proceeds as follows:

- `<DCHOME>\MB\Config\defaultConfigurations.txt`
- `<DCHOME>\MB\Config\Files\*` (the sub folders of this directory contain the entire range of parameters)
- `<DCHOME>\Engagements\<project name>\Config\*` (the files containing the project configuration settings; these are read in random order)
- `<DCHOME>\MB\Config\customConfigurations.txt`

Customizing installation configuration

The `<DCHOME>\MB\Config` directory contains the files you use to customize an installation of Cogility Studio. Within that directory, the `\Files` folder contains several default configuration files. The most pertinent of the settings in these files are summarized in the `Configurations.txt` file. These are the installation-specific configuration files, as opposed to project-specific configuration files. See “Project configuration” on page 14.

Some of the installation-specific configuration settings pertain to Cogility Modeler and other tools and establish such specifications as the window size, font sizes, and so forth. Other settings pertain to the authoring repository (see “Authoring repository configuration” on page 21 of the guide, *Modeling with Cogility Studio*), custom Java actions and other pre-deployment considerations. Likewise, there are parameters that pertain to Cogility Manager and model execution. Both types of settings are described throughout the documentation and are listed in the index, with an entry for each parameter by name and a listing of all configuration parameters under Configuration Parameters.

All of the configuration parameters are documented with comments in the defaultConfigurations.txt file.

Note: Do not edit either the files in the `Config\Files` folder or the `defaultConfigurations.txt` file.

Instead, to customize your configuration, create a `customConfigurations.txt` file located in the `<DCHOME>\MB\Config` folder. Then copy the settings from the defaultConfigurations.txt file that you want to reset into your customConfigurations.txt file and edit the settings.

When you start Cogility Modeler or at run time, the `customConfigurations.txt` file is read last; its settings supercede the settings of the files in the `<DCHOME>\MB\Config\Files` folder. These settings also supercede any of the project-specific configuration settings that might be duplicated in your project configuration files. See "Project configuration" on page 14.

When you first run Cogility Modeler, the configuration parameters you have set are those with which your model is pushed and executed. Also, some of the configuration settings may affect your model's consistency as you develop it. If any of these configuration settings need to change while you are working with the model, you do not want to restart Cogility Modeler and reload your model. Instead, you can just reload the configuration settings (see "Reloading configuration parameters into Cogility Modeler" on page 28 of the guide, *Modeling with Cogility Studio*), including project configuration settings (described in "Project Configuration Editor" on page 49).

During run time, Cogility Manager lets you refresh the configuration parameters for your application while the application server is still running. Configuration parameters are cached upon startup of the application server, and can be reloaded upon restarting the application server. However, you do not have to restart in order to load new configuration parameters. Instead, you can send the application server a JMS message with the new configuration parameters. See "Reload configuration message" on page 36. You can also reload configuration parameters in Cogility Insight or in Cogility Action Pad. See "Cogility Functions" on page 93.

You may use these configuration parameters in your action semantics. See "Accessing configuration parameters in action semantics" on page 23.

In addition, configuration parameter values can be dumped upon the startup of any Cogility tool by setting the following configuration parameter to true:

```
com.cogility.dumpConfigurationParametersToConsole
```

Note: If uncommented configuration parameters exist in one or more files, it is noted in the Cogility tool which you are starting. This includes Modeler, all Operation Support tools, and the application server.

Project configuration

You may work with several different models, and each may utilize different source data or a different application and database configuration. This information may be collected in a specific project folder. You can configure your model to work with specific *project* data. Source code, Java libraries, and project-specific configuration information are examples of project data. Note that these are not model artifacts, yet they are used in building model artifacts or deploying the model. The Cogility Project Configuration Editor organizes project data for you. See "Project Configuration Editor" on page 49.

Project files are held in a folder named for the project and located under the `<DCHOME>\Engagements` directory. The subdirectories hold code libraries, configuration file libraries, action semantics script files and batch scripts for your project. Also, for each tool in Cogility Studio, a workspace file is maintained that tells the tool which project files to open when running that tool. These are discussed in the following sections. See also, "Projects" on page 49 for more information about this directory structure.

Project code libraries

The code libraries used in your project are located in the `3p-lib` and `lib` subdirectories of the project folder. These contain compiled JAR files for the Java code referenced in your action semantics. See "JAR files" on page 55 for information about working with JAR files in your project folder.

Project configuration libraries

Project configuration files are located in the Config subdirectory of the project folder. These include the project's metadata.xml file that describes the file structure for the entire project and the separate configuration files. See "Configuration files" on page 57 for more information about working with these files.

Project action semantics files

Action semantics scripts used in a batch process are copied to the ocl subdirectory of the project folder or bin subdirectory, depending on platform. See "Action semantics batch files" on page 53 for more information.

Project batch scripts

The batch scripts that run action semantics files for your project are located in the scripts subdirectory of the project folder. See "Action semantics batch files" on page 53 for more information.

Run-time configuration

During execution you can manipulate several system controls for debugging, reporting and the database connection pool, and the most common settings for these may be accessed through Cogility Insight. These are duplicates of some of the configuration settings detailed in the <DCHOME>\MB\Config\defaultConfigurations.txt file. Working with them in Cogility Insight allows you to override their settings on the application server's main memory without changing the settings in the actual configuration files. You can change a setting through Cogility Insight, observe the run-time behavior, then set the configuration back to its original state by refreshing configuration parameters from the file. See "Cogility Functions" on page 93 for more information about using Cogility Insight to do this. This section describes the most common configuration settings as they pertain to several areas of run-time data.

Debugging configuration options

The most commonly used debugging options are grouped below by area of interest: action semantics, state machine execution, JMS messaging, web service execution and run-time repository operations.

Action semantics debugging

The following configuration parameters provide visibility into the execution of action semantics.

com.ceira.ocl.println.debug

Description: If true, debugging messages from the com::ceira::DebugPrintln Java action will be printed to standard output. This is the main facility for printing out error messages during the execution of action semantics. It functions like a regular println statement, except it does not report to the console view of Cogility Insight, only to the standard output for the application server. The output does not identify itself (see the example below), so you should include a string that identifies the output as a debug statement.

Legal values: True or false.

Default: False.

Example: The following code is included in the UpdateCustomerAddress_IWS inbound web service:

```
java com::cogility::DebugPrintln('This is the output from a DebugPrintln
statement');
```

When the web service executes, the following statements are printed to standard output.

```
SystemOut     O [MESSAGE] [IWSExecHlp] Executing IWS: SimpleModel.UpdateCustomerAddress_IWS
SystemOut     O This is the output from a DebugPrintln statement
```

State machine debugging

State machines execute business processes and define transaction boundaries (see "Transactions" on page 181 of the guide, *Modeling with Cogility Studio* for more information). To follow these processes you can set any of the following parameters.

com.ceira.pm.enactment.debug.behavior

Description: If true, details of the business process executed by the state machine will be printed to standard output. Note that if the state machine includes composite states, the processes within those states will not be reported unless the configuration parameter, "com.ceira.pm.enactment.debug.composite" on page 16 is set to true as well.

Legal values: True or false.

Default: False.

Example: The following are just a few of the statements generated from the execution of a state machine:

```
SystemOut     O [MESSAGE] [ENACTMENT|BEHAVIOR]
[NAME="SimpleModel.CreateCustomer.CreateCustomer_SM.CreateCustomer_State"] Found 1 non-event
transitions, of which 1 passed their guard condition.
SystemOut     O [MESSAGE] [ENACTMENT|BEHAVIOR]
[NAME="SimpleModel.CreateCustomer.CreateCustomer_SM.CreateCustomer_State"] Taking non-Event
outbound transition
SystemOut     O [MESSAGE] [ENACTMENT|BEHAVIOR]
[NAME="SimpleModel.CreateCustomer.CreateCustomer_SM.CreateCustomer_State"] Normal exit
SystemOut     O [MESSAGE] [ENACTMENT|BEHAVIOR]
[NAME="SimpleModel.CreateCustomer.CreateCustomer_SM.CreateCustomer_State"] STATUS -> DONE
```

com.ceira.pm.enactment.debug.composite

Description: If true, details of the business process executed by the composite state of a state machine will be printed to standard output. Use this in conjunction with the configuration parameter, "com.ceira.pm.enactment.debug.behavior" on page 16 if you want to see messages from the entire state machine.

Legal values: True or false.

Default: False.

Example: The output generated is just like that for "com.ceira.pm.enactment.debug.behavior" on page 16. See the example for that parameter.

JMS message debugging

As JMS messages are published and received on the system, the following parameters reveal the processing at various levels.

com.ceira.pm.enactment.debug.message.gateway

Description: Prints out status information when the following occur:

- The JMS topic session is created
- A message is published, noting which E2M conversion was used and which topic it was published on
- A message is received, noting redelivery count (if any), the M2E conversions used, the topic on which the message was received and other message details
- A JMS MapMessage is created from an event
- (Before publishing) will list out values of the event attributes that will be mapped to the message
- noting if the message has a built in identifier of the state machine from which it was published
- A message is published from a state machine, noting when a message that triggers a state Do Activity is being published

Legal values: True or false.

Default: False.

Example: The following is one of several statements generated with this parameter set:

```
SystemOut     O [MESSAGE] [ENACTMENT|MSG_GW] [MessageOutputExecutor] Outbound Message Source
"[TOMObject:SimpleModel.CreationAck_Event:com.ceira.pm.objaccess.tom.instances.TOMInstance@2fbb115
6]" received.  Applied E2MConversion "SimpleModel.CreationAck_E2M", Resulting Message will be published
on Topic "topic://CreationAck_Topic?brokerVersion=1"
```

dctc.messaging.debug

Description: Prints messages to the Windows console that supports Cogility Message Traffic Viewer execution when a message is received by the Cogility Message Traffic Viewer.

Legal values: True or false.

Default: False.

Example: The following are some of the statements printed to standard output:

```
[MESSAGE] Looking up topic: CreationAck_Topic...
[MESSAGE] ...Done looking up topic
[MESSAGE] publish to a topic
[MESSAGE] ...Done publishing message
[MESSAGE] ************* Original topic name: CreationAck_Topic
[MESSAGE] ************* Parsed topic name: CreationAck_Topic
```

dctc.messaging.debug.conversion

Description: Not used

Legal values: True or false.

Default: False.

Example: Not used.

Web service debugging

During web service execution, the following parameters provide visibility into the process at various levels.

com.ceira.webservice.debug.ejb

Description: Indicates when web service EJBs are created and destroyed inside the application server. This is useful when you have several calls to the web service simultaneously (heavy loading) and need to see when additional EJBs are created to handle the load.

Legal values: True or false.

Default: False.

com.ceira.webservice.debug.encoder

Description: Indicates how values are parsed from SOAP XML strings and are converted to model web service argument attributes (message attributes).

Legal values: True or false.

Default: False.

Example: The following statements are printed to standard output:

```
[MESSAGE] [IWSExecHlp] Executing IWS: SimpleModel.UpdateCustomerAddress_IWS
[MESSAGE] [EncoderForSubEleParams] valueOfChild: initiating examination...
[MESSAGE] [EncoderForSubEleParams] valueOfChild: examining SOAPElement child of type
"com.sun.xml.messaging.saaj.soap.impl.TextImpl"
[MESSAGE] [EncoderForSubEleParams] valueOfChild: SOAPElement has one text-only child, returning its
value ("true")
```

com.ceira.webservice.debug.inbound

Description: Indicates when an inbound web service is called, and prints out the time, name and arguments. Indicates when parsed data from the SOAP message is copied to the web service argument, when the web service logic is being executed, when the pre-action is fetched, starts and ends. Indicates when the inbound web service execution starts and when execution completes, and when the database commits begin and end. Indicates errors, such as service not found. On successful completion, prints out the result of the call. Prints the completion time of the web service.

Legal values: True or false.

Default: False.

Example: The following statements are printed to standard output:

```
[MESSAGE] [IWSExecHlp] Copying WS Parameter List into input message...
[MESSAGE] [IWSExecHlp] ... copied WS Parameter List into input message (0ms)
[MESSAGE] [IWSExecHlp] ====================================================
[MESSAGE] [IWSExecHlp] Executing IWS: SimpleModel.UpdateCustomerAddress_IWS
[MESSAGE] [IWSExecHlp][Input] Type = SimpleModel.UpdateCustomerAddress_IWS.UCA_Arg
[MESSAGE] [IWSExecHlp][Input]     "state" -> "Connecticut" (java.lang.String)
[MESSAGE] [IWSExecHlp][Input]     "custID" -> "1901" [1969-12-31 16:00:01.901] (java.lang.Long)
[MESSAGE] [IWSExecHlp][Input]     "emailAddress" -> "support@cogility.com" (java.lang.String)
[MESSAGE] [IWSExecHlp][Input]     "zip" -> "40098" (java.lang.String)
[MESSAGE] [IWSExecHlp][Input]     "city" -> "New London" (java.lang.String)
[MESSAGE] [IWSExecHlp][Input]     "street" -> "123 Any Way" (java.lang.String)
[MESSAGE] [IWSExecHlp] =======================================================
[MESSAGE] [IWSExecHlp] Obtaining the pre-action...
[MESSAGE] [IWSExecHlp] ... obtained the pre-action (0ms)
[MESSAGE] [IWSExecHlp] Executing the pre-action of SimpleModel.UpdateCustomerAddress_IWS...
[MESSAGE] [IWSExecHlp] ... executed the pre-action (1022ms)
[MESSAGE] [IWSExecHlp] Result message contains (dumpAttributes())...
[MESSAGE] [TOMInstance][Dump] of "[TOMObject:SimpleModel.UpdateCustomerAddress_I
WS.UCA_Result:com.ceira.pm.objaccess.tom.instances.TOMInstance@1b853a5]"
[MESSAGE] [TOMInstance][Dump]     "description" -> "Success: The Customer address has been updated with
email address." (java.lang.String)
[MESSAGE] [IWSExecHlp] Result message contains (iterate and print)...
[MESSAGE] [IWSExecHlp]     "description" -> "Success: The Customer address has been updated with email
address."
[MESSAGE] [IWSExecHlp] =======================================================
[MESSAGE] [IWSExecHlp] Done Executing IWS: SimpleModel.UpdateCustomerAddress_IWS
[MESSAGE] [IWSExecHlp][Output] Type = SimpleModel.UpdateCustomerAddress_IWS.UCA_Result
[MESSAGE] [IWSExecHlp][Output]     "description" -> "Success: The Customer address has been updated with
email address." (java.lang.String)
[MESSAGE] [IWSExecHlp] =======================================================
[MESSAGE] [IWSExecHlp] Committing data transaction...
[MESSAGE] [IWSExecHlp] ... committed data transaction (0ms)
```

com.ceira.webservice.debug.invoke

Description: Lists the information about the endpoint of the web service being invoked. Provides details of service handlers that were invoked during the inbound as well as the outbound process (when replying to an invocation). Prints out details of the attribute list of the argument used to call the web service.

Legal values: True or false.

Default: False.

Example: The following statements are printed to standard output:

```
[MESSAGE] [IWSExecHlp] Executing IWS: SimpleModel.UpdateCustomerAddress_IWS
[MESSAGE] [WsJaxmInvokeHelper] Web Service Access Parameters:
[MESSAGE] [WsJaxmInvokeHelper]     endpoint    = "http://www.webservicex.net/ValidateEmail.asmx"
[MESSAGE] [WsJaxmInvokeHelper]     service     = "ValidateEmail"
[MESSAGE] [WsJaxmInvokeHelper]     port        = "ValidateEmailSoap"
[MESSAGE] [WsJaxmInvokeHelper]     namespace   = "http://www.webservicex.net"
[MESSAGE] [WsJaxmInvokeHelper]     soapAction  = "http://www.webservicex.net/IsValidEmail"
[MESSAGE] [WsJaxmInvokeHelper]     argEncode   = "WSParameterEncodingEnum(parameters as sub-
elements=1)"
[MESSAGE] [WsJaxmInvokeHelper]     resDecode   = "WSParameterEncodingEnum(parameters as sub-
elements=1)"
[MESSAGE] [WsJaxmInvokeHelper]     litEncode   = true
[MESSAGE] [WsJaxmInvokeHelper]     encTLName   = "IsValidEmail"
[MESSAGE] [WsJaxmInvokeHelper]     in NS ovr   = "null"
[MESSAGE] [WsJaxmInvokeHelper]     out NS ovr  = "null"
[MESSAGE] [WsJaxmInvokeHelper]     WebSvc name = "SimpleModel.OWS.ValidateEmail.IsValidEmail"
[MESSAGE] [WsJaxmInvokeHelper]     isEnactment = "true"
[MESSAGE] [WsJaxmInvokeHelper] ================================================
[MESSAGE] [WsJaxmInvokeHelper] Parameter List Dump (1 parameters):
[MESSAGE] [WsJaxmInvokeHelper] ================================================
[MESSAGE] [WsJaxmInvokeHelper]     "0:Email" -> "support@cogility.com" (java.lang.String)
[MESSAGE] [WsJaxmInvokeHelper] =================================================
[MESSAGE] [WsJaxmInvokeHelper] - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[MESSAGE] [WsJaxmInvokeHelper] SOAP connection factory class.:
com.sun.xml.messaging.saaj.client.p2p.HttpSOAPConnectionFactory
[MESSAGE] [WsJaxmInvokeHelper] SOAP connection class.........:
com.sun.xml.messaging.saaj.client.p2p.HttpSOAPConnection
[MESSAGE] [WsJaxmInvokeHelper] SOAP message factory class....:
com.sun.xml.messaging.saaj.soap.ver1_1.SOAPMessageFactory1_1Impl
[MESSAGE] [WsJaxmInvokeHelper] SOAP message class............:
com.sun.xml.messaging.saaj.soap.ver1_1.Message1_1Impl
[MESSAGE] [WsJaxmInvokeHelper] SOAP envelope class...........:
com.sun.xml.messaging.saaj.soap.ver1_1.Envelope1_1Impl
[MESSAGE] [WsJaxmInvokeHelper] Document builder factory class:
com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderFactoryImpl
[MESSAGE] [WsJaxmInvokeHelper] SOAP factory class: class
com.sun.xml.messaging.saaj.soap.ver1_1.SOAPFactory1_1Impl
[MESSAGE] [WsJaxmInvokeHelper] - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[MESSAGE] [WsJaxmInvokeHelper] About to send SOAP Message (OWS):
[MESSAGE] [WsJaxmInvokeHelper] <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
soap/envelope/" xmlns:s0="http://www.webservicex.net" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><SOAP-ENV:Header/><SOAP-
ENV:Body><s0:IsValidEmail><s0:Email>support@cogility.com</s0:Email><s0:IsValidEmail></SOAP-
ENV:Body></SOAP-ENV:Envelope>
[MESSAGE] [WsJaxmInvokeHelper] - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[MESSAGE] [WsJaxmInvokeHelper] SOAP response was:
[MESSAGE] [WsJaxmInvokeHelper] <?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><IsValidEmailResponse
xmlns="http://www.webservicex.net"><IsValidEmailResult>true</IsValidEmailResult></
IsValidEmailResponse></soap:Body></soap:Envelope>
[MESSAGE] [WsJaxmInvokeHelper] Returned values:
[MESSAGE] [WsJaxmInvokeHelper] =================================================
[MESSAGE] [WsJaxmInvokeHelper] Parameter List Dump (1 parameters):
[MESSAGE] [WsJaxmInvokeHelper] =================================================
[MESSAGE] [WsJaxmInvokeHelper]     "0:IsValidEmailResult" -> "true" (java.lang.String)
[MESSAGE] [WsJaxmInvokeHelper] =================================================
```

com.ceira.webservice.debug.jmsoutput

Description: Indicates if JMS setup can be performed successfully during a web service call.

Legal values: True or false.

Default: False.

Example: The following statements are printed to standard output:

```
[MESSAGE] [JMSOutputExecutorForWS] TopicCFLookup = "DemoConnectionFactory"
[MESSAGE] [IWSExecHlp] Executing IWS: SimpleModel.UpdateCustomerSLA_IWS
```

com.ceira.webservice.debug.paramlist

Description: Indicates if the parameter list created from the SOAP message has a parameter that does not have a matching attribute in the model's message object to which it is mapped.

Legal values: True or false.

Default: False.

Run-time configuration options

com.cogility.manager.auto.delete.state.rt

Description: Indicates whether to auto-delete all the state runtime instances whenever a State machine completes. This is done because the StateRT instance table can get pretty full, and the individual state track records are not useful anyway.

Legal values: True or false.

Default: False.

com.cogility.ocl.locateActionIgnoresNull

Description: Indicates if the locate or locateAll action ignores attributes with null values. The default is true, which means an attribute with null value is not used as a search criteria when locating business objects. Setting this parameter to false allows the explicit matching of null values in the search criteria.

Legal values: True or false.

Default: True.

com.cogility.pear.useDuplicatedFixedSchema

Description: Defaults to false. It may be set to true after running the FixedSchemaDuplicator utility so the application server can temporarily use a set of alternative fixed schema while the original fixed schema is being updated. This property must be quoted out or set to false for normal operations.

Legal values: Ttrue or false.

Default: False

com.cogility.pear.maxInstancesReturned

Description: limits the number of instances returned by a retrieve operation. This limit applies to the max number of instances returned for a single class (subclass instances are counted separately). For example, A has subclass B. A has 30000 instances, B has 15000. With the max set to 20000, a retrieveAllWithSubclasses operation on A would result 35000 objects: 20000 instances of A and 15000 of B. To turn this limit off, set to 0. The default value is 20000.

Legal values: Any positive integer.

Default: 20000.

PEAR connection pooling

Cogility Studio stores model metadata as well as business object data in a relational database, referred to as the runtime repository. See “Run-time respiratory” on page 321 of the guide, *Modeling with Cogility Studio*. At runtime, multiple parallel threads are spawned in the J2EE application server. These threads execute logic defined in the model and access the database. In order to optimize database resources, Cogility Studio maintains a pool of database connections. The following configuration parameters pertain to the settings that Cogility Studio uses for connection pooling – in most cases, users do not need to change these default values.

com.ceira.pear.connectionPool.initialSize

Description: The initial number of database connections available to Cogility Studio. This sets the lower limit of the connection pool.

Legal values: Any positive integer.

Default: 10

com.ceira.pear.connectionPool.maxSize

Description: The maximum number of database connections available to Cogility Studio.

Legal values: Any positive integer.

Default: 20

com.ceira.pear.connectionPool.increment

Description: The number of new database connections to add to the pool if all available connections are in use and the maximum has not been reached.

Legal values: Any positive integer.

Default: 5

com.ceira.pear.connectionPool.maxWait

Description: If the database connection pool has maximum number of allowed connections, and all connections are in use, a request to obtain a connection will wait up to this amount of time in milliseconds to see if a connection becomes available before giving up.

Legal values: Any positive integer.

Default: 20000

Sequenceable attribute settings

Business objects defined in the model may have sequenceable attributes. The value of these attributes is not explicitly modified by the model. Instead, the database assigns a sequentially increasing number automatically for each instance of such an attribute. See "Sequenceable attributes" on page 94 of the guide, *Modeling with Cogility Studio*.

The following settings control the values of these sequenceable attributes. There are two sets of configuration parameters, depending on which database is used.

Global sequenceable attribute settings

For Oracle and DB2 running on Windows, Cogility Studio supports global sequenceable attributes (all objects on the database share the same sequence).

com.ceira.pear.dbSequence.initialValue

Description: This is the initial value of the global sequence.

Legal values: Any positive integer.

Default: 1001

com.ceira.pear.dbSequence.increment

Description: This is the value by which global sequenceable attributes will be incremented.

Legal values: Any positive integer.

Default: 1

Business object-specific sequences

For SQL Server, MySQL and DB2 running on AS400, Cogility Studio supports business object-specific sequences (each object on the database has its own sequence). Sequenceable attribute configuration parameters may be defined for each class in the model that has a sequenceable attribute. If these values are not defined, the global settings will take effect (see "Global sequenceable attribute settings" on page 22). For example, in a model called MySchema, a class called Order with a sequential attribute called orderNumber would have configuration parameters as follows:

```
# the initial value of the sequence for this particular class.
dbSequence.MySchema.MIM.Order.initialValue = 2005

# the increment value of the aforementioned sequence
dbSequence.MySchema.MIM.Order.increment = 1
```

Configuration parameter testing

Cogility has a host of configurable parameters that can be manipulated and experimented with to control the behavior of the executable action semantics. All of these values can be defined and manipulated in a test environment within Cogility Action Pad, to test specific bits of business logic under circumstances identical to how it will run inside the application server, but in a much more interactive manner.

Accessing configuration parameters in action semantics

Your action semantics may need to access configuration parameter settings for your Cogility Studio installation. To do this, you declare a variable of type com::dctc::configurations::DcConfigurationFacility and use its methods to retrieve configuration parameter values. For more information on configuration parameters, see "Customizing installation configuration" on page 13.

For example, in the following lines, the variable `dc` is declared to hold the DcConfigurationFacility object. Using the getStringProperty method, the configuration parameter value is retrieved and assigned to the `str` variable.

```
dc : com::dctc::configurations::DcConfigurationFacility;
str: java::lang::String;
str := dc.getStringProperty('com.cogility.strValue');
```

The DcConfigurationFacility object includes the following methods. For the first set of three methods, you pass the name of the configuration parameter as a java::lang::String:

- getBooleanProperty(java::lang::String) - retrieves a Boolean property.
- getIntegerProperty(java::lang::String) - retrieves an integer property.
- getStringProperty(java::lang::String) - retrieves a String property.

For the following three methods, the first argument is the name of the configuration parameter passed as a java::lang::String; the second argument is the default value to use if the configuration parameter does not exist. If the configuration parameter does exist, but no value is specified, the empty string is returned.

- getBooleanProperty(java::lang::String, java::lang::Boolean) - retrieves a Boolean property, sets the provided value if the configuration parameter does not exist.
- getIntegerProperty(java::lang::String, java::lang::Integer) - retrieves an integer property, sets the provided value if the configuration parameter does not exist.
- getStringProperty(java::lang::String, java::lang::String) - retrieves a String property, sets the provided value if the configuration parameter does not exist.

Custom web applications

Cogility Studio may be configured to deploy your custom web applications alongside built-in web applications like Cogility Insight. The applications must be pre-packaged as J2EE WAR files, and Cogility will keep them "as-is" with the exception of modifying their manifest files to contain Cogility classpath (since that is the primary reason for the applications to be part of Cogility Studio in the first place). The information for these applications must be provided in configuration parameters. The following examples show how they are specified:

```
com.cogility.deployment.aux.webapps.count=2

com.cogility.deployment.aux.webapps.context.1=WebAppOne

com.cogility.deployment.aux.webapps.context.2=WebAppTwo

com.cogility.deployment.aux.webapps.warfilepath.1=C:/MyApp/appOne.war

com.cogility.deployment.aux.webapps.warfilepath.2=C:/MyApp/appTwo.war
```

A typical URL to access a custom web application should start with either ‘http://hostname:port/contextname’ or ‘https://hostname:port/contextname’ depending on whether or not SSL is involved. In the example configuration parameters above, if deployed to a SSL-enabled WebLogic application server, you could access WebAppOne by using the URL ‘https://localhost:7002/WebAppOne/home.jsp’.

4

Iterative Development

As you work with your system model following initial deployment, you will need to update the run-time environment or test pieces of it apart from the production domain on which it is deployed. Cogility Studio provides several facilities for iterative model development and deployment that get around the problem of redeploying the entire model.

Push modes

The discussion of "Deployment" on page 28 of the guide, *Modeling with Cogility Studio*, describes the deployment process and the objects created on both the application server and the database. This process has several variations that accomodate different types of revision to a running enterprise application.

In each of the push modes, the model schema is assumed to be additive. That is, any revisions to the information model or to messages and events is assumed to build on top of the extant schema. With any of these push modes, it is impossible to push into execution a schema that has any of the original schema definitions removed. However, you can update the schema with a non-additive schema. See "Non-additive schema updates" on page 30.

Some changes to the model's action semantics or changes to the configuration parameters prior to pushing the model may require that the application server be restarted. A dialog following the push notifies you of this.

A configuration parameter hides these alternative push mode selections from the main menu of Cogility Modeler. It defaults to true to show the menu selections, but you can set it to false to hide them. The configuration parameter reads as follows:

```
com.cogility.enableExtraPushMenuItems
```

Full model push

A push of an entire model into execution is described in "Pushing the model into execution" on page 326 of the guide, *Modeling with Cogility Studio*. Use this mode when you first push the model into execution.

A full push of the model the first time accomplishes the following:

- Parse the information model and generate the database schema to match it in the execution database (dynamic schema)
- Create the fixed schema in the specified database, which stores the executable artifacts derived from the model
- Parse the entire system model and create the definitions of all the information model artifacts in the execution database

- Parse the system model and create definitions for all the events and messages in the execution database
- Create definitions for all of the following artifact types:
    - Conversions in the execution database
    - Transformations in the execution database
    - Web services in the execution database
    - Information model behaviors and operations in the execution database
    - JMS-related artifacts in the application server environment
    - Web services in the application server environment
    - Cogility Insight-related artifacts

Incremental push

Subsequent to the first full push of a model, pushing the model incrementally requires fewer system resources as Cogility Modeler compares the new model artifacts against the extant model artifacts and pushes only the required changes.

A full push of the model after the first push accomplishes the following:

- Parse the information model and generate the additional database schema to match this in the execution database (dynamic schema)
- Parse the system model and validate the definitions of all the information model artifacts in the execution database
- Parse the system model and validate definitions for all the events and messages in the execution database
- Recreate definitions for all the conversions in the execution database
- Recreate definitions for all transformations in the execution database
- Create definitions for all new and modified web services in the execution database and remove unused web services
- Create definitions for all new and modified information model behaviors and operations in the execution database and remove unused ones, marking the latest behavior versions as current
- Create definitions for all new JMS-related artifacts in the application server environment and remove unused ones
- Create definitions for all new web services in the application server environment and remove unused ones
- Create Cogility Insight-related artifacts, if required

To accomplish an incremental push of the model, push the entire model into execution. See “Pushing the model into execution” on page 326 of the guide, *Modeling with Cogility Studio*.

Schema only push

A system model schema only push parses the information model and generates the additional database schema (dynamic schema) to match it in the execution database, as long as the schema being pushed is consistent with the schema that exists in the database. This is useful if a single artifact push is required that depends upon a change that has been made to the overall information model. See “Single artifact push” on page 28.

Pushing the business object schema only

To push the model schema only

1. In Cogility Modeler's tree view, select the model container.
2. Turn over the model.

   Turning over a model creates a new version of the model in the change management system. Before you can push a model into execution, you must turn it over. For more information, see "Turn-over version" on page 23 of the guide, *Change Management in Cogility Studio*.
3. From the **Selection** menu, choose **Actions > Push Business Object Schema Only**.
4. In the dialog, select the source of deployment parameters and click **Use Deployment**.

   See "Configuration Parameters" on page 28 of the guide, *Modeling with Cogility Studio*.

   During push, the push action view lists the actions performed as objects are created in the execution environment.
5. If the push is successful, a notice to that affect appears. Click **OK**.

   The Cogility Modeler console window indicates that the model successfully deployed to the application server.
6. If the push is unsuccessful, you see an error message.

   The error messages are also displayed in the push action view at the bottom of the Cogility Modeler window.

Application server only push

An application server only push allows the entire model to be deployed to the application server without creating any database artifacts. This only updates the J2EE application server, the database is not involved. This is desirable when your run-time environment contains multiple application servers, each of which would target the same run-time database. See "Pushing application server artifacts only" on page 27

You may also choose to force an update of the application server artifacts during a regular push by checking the Force Push of Application Server Artifacts checkbox. If this box is left unchecked, the Cogility.ear file will be repushed only if required. A new Cogility.ear file is created and deployed when changes are made to either WS deployments or JMS topics. See "Forcing application server artifacts push" on page 27.

Forcing application server artifacts push

To force a push of application server artifacts:

1. Push the model into execution with the **Force Push of Application Server Artifacts** push option checked.

   See "Pushing the model into execution" on page 326 of the guide, *Modeling with Cogility Studio*.

Pushing application server artifacts only

To push the application server artifacts only

1. In Cogility Modeler's tree view, select the model container.
2. If the application server is not running, start it.

   See "Application server" on page 323 of the guide, *Modeling with Cogility Studio*.
3. Turn over the model.

Turning over a model creates a new version of the model in the change management system. Before you can push a model into execution, you must turn it over. For more information, see "Turn-over version" on page 23 of the guide, *Change Management in Cogility Studio*.

4. From the **Selection** menu, choose **Actions > Push to AppServer**.
5. In the dialog, select the source of deployment parameters and click **Use Deployment**.

   See "Configuration Parameters" on page 28 of the guide, *Modeling with Cogility Studio*.

   During push, the push action view lists the actions performed as objects are created in the execution environment.
6. If the push is successful, a notice to that affect appears. Click **OK**.

   The Cogility Modeler console window indicates that the model successfully deployed to the application server.
7. If the push is unsuccessful, you see an error message.

   The error messages are also displayed in the push action view at the bottom of the Cogility Modeler window.

Database only push

A database-only push allows the entire model to be deployed to the database without creating any application server artifacts.

Pushing to the database only

To push to the database only

1. In Cogility Modeler's tree view, select the model container.
2. Turn over the model.

   Turning over a model creates a new version of the model in the change management system. Before you can push a model into execution, you must turn it over. For more information, see "Turn-over version" on page 23 of the guide, *Change Management in Cogility Studio*.
3. From the **Selection** menu, choose **Actions > Push to Database**.
4. In the dialog, select the source of deployment parameters and click **Use Deployment**.

   See "Configuration Parameters" on page 28 of the guide, *Modeling with Cogility Studio*.

   During push, the push action view lists the actions performed as objects are created in the execution environment.
5. If the push is successful, a notice to that affect appears. Click **OK**.

   The Cogility Modeler console window indicates that the model successfully deployed to the application server.
6. If the push is unsuccessful, you see an error message.

   The error messages are also displayed in the push action view at the bottom of the Cogility Modeler window.

Single artifact push

A system model single artifact push allows for an iterative development cycle of modify, push, execute and repeat. It covers the following model artifacts:

- A single inbound web service
- All behaviors (state machines) defined on a single MIM or DIM class
- All class operations and method bodies defined on a single MIM or DIM

Note that to make a single new method body active, you must push the entire operation hierarchy: the class where it is defined, and all classes where it is implemented, following the class hierarchy order from superclass to subclass.

- All operations and method bodies defined on a message or event
- All custom queries

A single artifact push updates only the database, the application server is not updated.

If the artifact is mutable, it is always deployed. If immutable, it is deployed only if it is different from the version in the database. Immutable artifacts are those that have been turned over or versioned. See "Turn-over version" on page 23 of the guide, *Change Management in Cogility Studio*. See "User modifications" on page 9 of the guide, *Change Management in Cogility Studio* for more information about mutable versus immutable objects.

Note: Do not deploy mutable artifacts without a version name into a production environment!

Pushing a single artifact

To push a single inbound web service:

1. In Cogility Modeler's tree view, select the artifact.
2. From the **Selection** menu, choose **Actions >** and one of the following:
   - **Push Single IWS**
   - **Push Single Class Behaviors**
   - **Push Single Class Operations**
3. In the dialog, select the source of deployment parameters and click **Use Deployment**.

   See "Configuration Parameters" on page 28 of the guide, *Modeling with Cogility Studio*.

   During push, the push action view lists the actions performed as objects are created in the execution environment.
4. If the push is successful, a notice to that affect appears. Click **OK**.

   The Cogility Modeler console window indicates that the model successfully deployed to the application server.
5. If the push is unsuccessful, you see an error message.

   The error messages are also displayed in the push action view at the bottom of the Cogility Modeler window.

   Pushing all custom queries

To push all custom queries:

1. In Cogility Modeler's tree view, select the model container.
2. From the **Selection** menu, choose **Actions > Push All Custom Queries**.
3. In the dialog, select the source of deployment parameters and click **Use Deployment**.

   See "Configuration Parameters" on page 28 of the guide, *Modeling with Cogility Studio*.

   During push, the push action view lists the actions performed as objects are created in the execution environment.
4. If the push is successful, a notice to that affect appears. Click **OK**.

   The Cogility Modeler console window indicates that the model successfully deployed to the application server.
5. If the push is unsuccessful, you see an error message.

The error messages are also displayed in the push action view at the bottom of the Cogility Modeler window.

Non-additive schema updates

Once a model schema is established on the run-time repository, you cannot push onto that schema a revised schema that lacks any extant elements. Such a schema is called non-additive. Before you can push a non-additive schema, you must edit the extant schema, removing or renaming those elements no longer in use. A model schema has both persistent elements such as classes and transient elements such as events and messages. Each type of element requires a different schema modification technique.

In all cases, you must locate and address the repercussions of the schema change within the model yourself. These techniques assume a hands-on knowledge of the model and require you to follow all changes you make throughout the model schema. They are not an automatic update of the system model as is a push of the model or model artifacts.

Force action semantics recompilation

Push is optimized to only recompile action semantics that have been updated. If the model's action semantics have not changed, they will not be recompiled. However there are times when you might want to force Cogility to recompile the action semantics. These scenarios include changed project-specific java code or third-party jar files, or a migration to a newer release of Cogility Studio. You specify a forced recompilation by checking the box for Recompiled Action Semantics in the push dialog. See "Pushing the model into execution" on page 326 of the guide, *Modeling with Cogility Studio*.

Persistent schema update

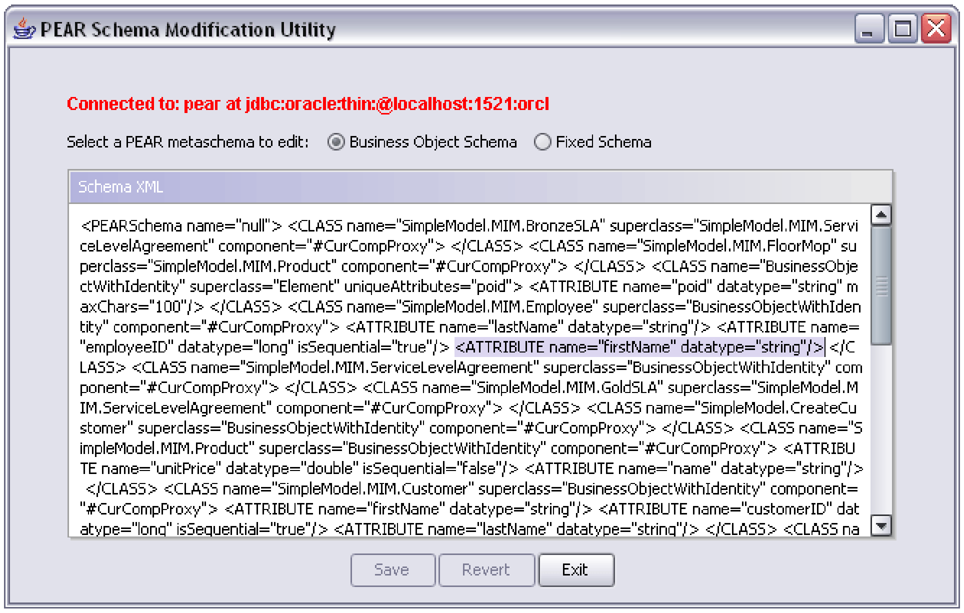
For updating the persistent information model schema with a non-additive schema, you may follow the technique described here. The persistent schema may apply to execution data already created at run-time, and this limits the extent to which you can change the persistent model schema. Only the following are supported for non-additive schema modification:

- Removing an attribute - see "Removing an attribute" on page 31
- Renaming an attribute - see "Renaming an attribute" on page 32
- Removing an association - see "Removing an association" on page 33
- Replacing an association with an association class without renaming (deleting the association and recreating it as an association class) - see "Replacing an association with an association class" on page 33

To complete the modification, you must modify both the persistent schema and the extant execution data. To modify the execution data, you use database tools such as DB Visualizer, moving the data out of old tables and into new ones as needed. To modify the persistent model schema you use the Pear Schema Modification Utility. For more information about the PEAR schema, see "Run-time respiratory" on page 321 of the guide, *Modeling with Cogility Studio*.

The Pear Schema Modification Utility presents an XML description of your model's persistent schema that you can edit and save. The utility edits the pear schema at the location where the model was last deployed. Be sure you have deployed to a location where you can modify the persistent schema before you run the utility. For instance, before you modify the schema in a production environment, you'll want to first test the modification in a test environment. You should first deploy the model and test the modification in that environment before you modify the schema in the production environment. The file, <DCHOME>\MB\Config\Files\pear-params.txt describes the location of the run-time repository where the model was last pushed. You can check that file before running the utility to be sure you are working with the proper repository.

The Pear Schema Modification Utility provides the option to modify either the business object schema (for the business objects in your information model, also called the dynamic schema) or the fixed schema (for the Cogility model artifacts). Do not modify the fixed schema. This option is for internal development only.



Connected to: pear at jdbc:oracle:thin:@localhost:1521:orcl

Select a PEAR metaschema to edit: [✓ Business Object Schema] Fixed Schema

```
<PEARSchema name="null"> <CLASS name="SimpleModel.MIM.BronzeSLA" superclass="SimpleModel.MIM.Servi
ceLevelAgreement" component="#CurCompProxy"> </CLASS> <CLASS name="SimpleModel.MIM.FloorMop" su
perclass="SimpleModel.MIM.Product" component="#CurCompProxy"> </CLASS> <CLASS name="BusinessObje
ctWithIdentity" superclass="Element" uniqueAttributes="poid"> <ATTRIBUTE name="poid" datatype="string" m
axChars="100"/> </CLASS> <CLASS name="SimpleModel.MIM.Employee" superclass="BusinessObjectWithIden
tity" component="#CurCompProxy"> <ATTRIBUTE name="lastName" datatype="string"/> <ATTRIBUTE name=
"employeeID" datatype="long" isSequential="true"/> <ATTRIBUTE name="firstName" datatype="string"/> </C
LASS> <CLASS name="SimpleModel.MIM.ServiceLevelAgreement" superclass="BusinessObjectWithIdentity" com
ponent="#CurCompProxy"> </CLASS> <CLASS name="SimpleModel.MIM.GoldSLA" superclass="SimpleModel.M
IM.ServiceLevelAgreement" component="#CurCompProxy"> </CLASS> <CLASS name="SimpleModel.CreateCu
stomer" superclass="BusinessObjectWithIdentity" component="#CurCompProxy"> </CLASS> <CLASS name="S
impleModel.MIM.Product" superclass="BusinessObjectWithIdentity" component="#CurCompProxy"> <ATTRIBU
TE name="unitPrice" datatype="double" isSequential="false"/> <ATTRIBUTE name="name" datatype="string"/>
 </CLASS> <CLASS name="SimpleModel.MIM.Customer" superclass="BusinessObjectWithIdentity" component=
"#CurCompProxy"> <ATTRIBUTE name="firstName" datatype="string"/> <ATTRIBUTE name="customerID" dat
atype="long" isSequential="true"/> <ATTRIBUTE name="lastName" datatype="string"/> </CLASS> <CLASS na
```

Figure 4-1: Pear Schema Modification Utility

Removing an attribute

To remove an attribute from the persistent schema:

1. Read the file, <DCHOME>\MB\Config\Files\pear-params.txt for the location of the PEAR schema to which you want to push a schema with the attribute removed.

   You should first test the modification in non-production location. Be sure that location is defined in pear-params.txt.

2. Execute the script,
   <DCHOME>\MB\scripts\Dbtools\Run_Pear_Schema_Modification_Utility.bat.

   This runs the Pear Schema Modification Utility.

   a. In the **Pear Schema Modification Utility** window, right-click and select **Find/Replace**.

   b. In the **Find/Replace** dialog, in the **Find String** field, enter the name of the attribute you wish to remove.

   c. Click **Find From Start**.

      Locate the attribute in its proper class. Note that the attribute name may be used for attributes in more than one class.

   d. Select the attribute, including its tag identifier and brackets.

      For example, `<ATTRIBUTE name="firstName" datatype="string"/>`.

   e. Press the **Delete** key.

   f. Close the **Find/Replace** window.

   g. In the **Pear Schema Modification Utility** window, click **Save** and **Exit**.

3. In Cogility Modeler, delete the attribute from the model.

   See “Deleting model artifacts” on page 27 of the guide, *Modeling with Cogility Studio*.
4. In Cogility Modeler, push the model into execution.

   Be sure that you select the deployment that corresponds to the location where you updated the PEAR schema in step 2, above. See “Pushing the model into execution” on page 326 of the guide, *Modeling with Cogility Studio*.

   The run-time repository still has a column for the attribute you deleted, and you can remove the data from that column, if necessary, using a database tool such as DB Visualizer. In these steps only the model’s schema has been modified so that the attribute may no longer be used in model development.
5. Once you are satisfied that your model is working properly in execution, repeat these steps for the production schema.

Renaming an attribute

To rename an attribute in the persistent schema:

1. Edit the file, <DCHOME>\MB\Config\Files\pear-params.txt for the location of the PEAR schema to which you want to push a schema with the attribute renamed.

   You should first test the modification in non-production location. Be sure that location is defined in pear-params.txt.
2. Execute the script,
   <DCHOME>\MB\scripts\Dbtools\Run_Pear_Schema_Modification_Utility.bat.

   This runs the Pear Schema Modification Utility.
   a. In the **Pear Schema Modification Utility** window, right-click and select **Find/Replace**.
   b. In the **Find/Replace** dialog, in the **Find String** field, enter the name of the attribute you wish to remove.
   c. Click **Find From Start**.

      Locate the attribute in its proper class. Note that the attribute name may be used for attributes in more than one class.
   d. Select the attribute, including its tag identifier and brackets.

      For example, `<ATTRIBUTE name="firstName" datatype="string"/>`.
   e. Press the **Delete** key.

      Note that you are deleting the attribute, not renaming it here. You will rename it in the model using Cogility Modeler.
   f. Close the **Find/Replace** window.
   g. In the **Pear Schema Modification Utility** window, click **Save** and **Exit**.
3. In Cogility Modeler, rename the attribute in the model.

   See “Editing model artifacts” on page 27 of the guide, *Modeling with Cogility Studio*.
4. In Cogility Modeler, push the model into execution.

   Be sure that you select the deployment that corresponds to the location where you updated the PEAR schema in step 2, above. See “Pushing the model into execution” on page 326 of the guide, *Modeling with Cogility Studio*.

   Essentially, you are now pushing an additive schema into execution. Neither the model nor the PEAR repository has a definition for the old attribute, and you are simply pushing the new attribute.
5. In the run-time repository (execution database), copy the data from the old column (with the old attribute name) the new column (with the new attribute name).

The run-time repository still has a column for the attribute you deleted, and you must copy the data from that column to the new column using a database tool such as DB Visualizer.

6. Once you are satisfied that your model is working properly in execution, repeat these steps for the production schema.

Removing an association

To remove an attribute from the persistent schema:

1. Edit the file, <DCHOME>\MB\Config\Files\pear-params.txt for the location of the PEAR schema to which you want to push a schema with the association removed.

   You should first test the modification in non-production location. Be sure that location is defined in pear-params.txt.

2. Execute the script, <DCHOME>\MB\scripts\Dbtools\Run_Pear_Schema_Modification_Utility.bat.

   This runs the Pear Schema Modification Utility.

   a. In the **Pear Schema Modification Utility** window, right-click and select **Find/Replace**.
   b. In the **Find/Replace** dialog, in the **Find String** field, enter the name of the association you wish to remove.
   c. Click **Find From Start**.
   d. Select the association, including its tag identifier and brackets.

      For example, the following XML describes an association:

```
      <ASSOCIATION name="A_address_customer" component="#CurCompProxy">
      <ROLE name="customer" class="SimpleModel.MIM.Customer"
maxCardinality="1"/>
      <ROLE name="address" class="SimpleModel.MIM.Address"/>
      </ASSOCIATION>
```

   e. Press the **Delete** key.
   f. Close the **Find/Replace** window.
   g. In the **Pear Schema Modification Utility** window, click **Save** and **Exit**.

3. In Cogility Modeler, delete the association from the model.

   See "Deleting model artifacts" on page 27 of the guide, *Modeling with Cogility Studio*.

4. In Cogility Modeler, push the model into execution.

   Be sure that you select the deployment that corresponds to the location where you updated the PEAR schema in step 2, above. See "Pushing the model into execution" on page 326 of the guide, *Modeling with Cogility Studio*.

   Physical truncation of the association junction tables and table dropping, if desired, can be achieved using tools such as DB Visualizer. In these steps only the model's schema has been modified so that the association may no longer be used in model development.

5. Once you are satisfied that your model is working properly in execution, repeat these steps for the production schema.

Replacing an association with an association class

Note that you can simply delete the association and replacing it with an association class of the same name in the model using Cogility Modeler. You do not need to modify the PEAR schema directly using the Pear Schema Modification Utility. Cogility Modeler will look at attributes of the association class as additive schema and allow the model to be pushed and the database to be updated with new column definitions. Obviously, the new association class attribute(s) will not have any values in the database. Existing associations in the database will be maintained.

To replace an association with an association class:

1. In Cogility Modeler, delete the association in the model.

   See "Deleting model artifacts" on page 27 of the guide, *Modeling with Cogility Studio*.
2. In Cogility Modeler, create the association class in the model; name it with the same name as the old association.

   See "Association classes" on page 116 of the guide, *Modeling with Cogility Studio*.
3. In Cogility Modeler, push the model into execution.

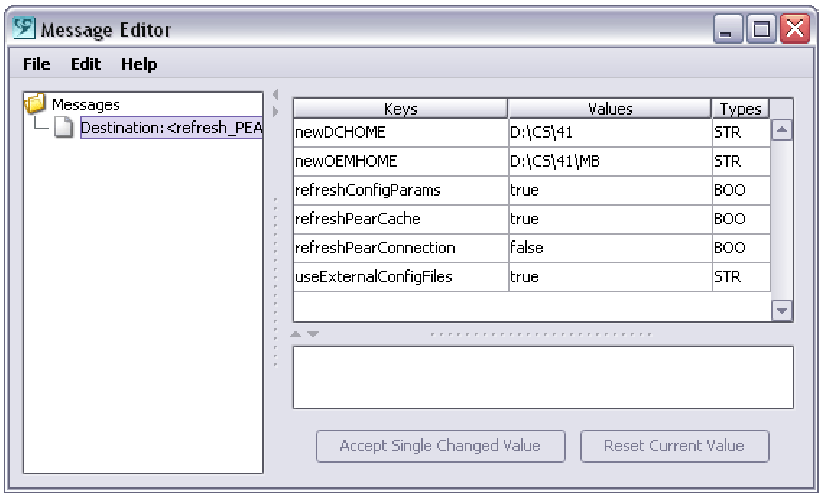
   Be sure that you select the deployment that corresponds to a test location.

   Essentially, you are now pushing an additive schema into execution. The PEAR repository does not have a definition for the new association class attribute(s), and you are simply pushing the new attribute(s). The PEAR repository still has a column for the association, and now has related columns for the association class attribute(s).
4. Once you are satisfied that your model is working properly in execution, repeat these steps for the production schema.

Dynamic execution model update

Dynamic execution model update is the process of updating the in-memory logic and configurations within the running application server. When the application server is started, the various configuration parameters (Cogility Studio as well as engagement-specific parameters) are read into memory. These parameters affect the execution behavior of the model-driven logic. Complex caching schemes write execution behavior into the application server memory as model-driven logic is executed against the application server. Cogility Studio includes several facilities for updating configuration parameters and refreshing these caches.

The dynamic execution model update is achieved by a non-model-driven message driven bean that is part of the J2EE application deployed with the executeable model by Cogility. You use Cogility Message Traffic Viewer to send the bean a JMS message that initiates the update. The message is located as <DCHOME>\MB\Config\RefreshPearMessage.msg. This message, shown in the figure below, includes several keys with default values for the various updates. The Boolean values tell the server whether or not to commit the update described in the key, the String values define new locations for their respective keys.



Keys	Values	Types
newDCHOME	D:\CS\41	STR
newOEMHOME	D:\CS\41\MB	STR
refreshConfigParams	true	BOO
refreshPearCache	true	BOO
refreshPearConnection	false	BOO
useExternalConfigFiles	true	STR

Figure 4-2: RefreshPearMessage.msg file

Notice that all key values except that of refreshPearCache are either undefined or false by default. The value of refreshPearCache is set to true by default. This does not have any bearing on the other settings, and it may remain set to true (or false) while another key's value is also true or otherwise defined.

Refresh PEAR connections

The Cogility Manager execution environment maintains a pool of connections to the execution database. These are used and reused by any executing model action semantics that require a connection. Due to network issues, the connections in the connection pool may become stale and need to be replaced by new connections. When the application server is instructed to refresh connections, it will clear out its database connection pool and generate new connections to the database.

Refreshing PEAR connections

To refresh PEAR connections with the application server running:

1. Run Cogility Message Traffic Viewer.

   See "Running Cogility Message Traffic Viewer" on page 119.
2. Click **Load**, navigate to <DCHOME>\MB\Config, select **RefreshPearMessage.msg** and click **Open**.
3. With the message selected, click **Inspect** (this starts Cogility Message Editor).
4. In Cogility Message Editor's tree view, select **Subject:<Refresh_pear>**.
5. In the key field, select **refreshPearConnection**; in the input field, enter **true** and click **Accept Single Changed Value**.
6. Close Cogility Message Editor.
7. In Cogility Message Traffic Viewer, click **Re-publish To Destination** to publish the message.

Refresh PEAR cache

Cogility Manager caches the executable model action semantics in the application server's memory. These include such elements as

- Web service operation logic
- Methods defined on classes, messages and events
- State machines
- Other action semantics defined in the model

If the definition of these has been updated in the run-time repository by means of a single artifact push (see "Single artifact push" on page 28), the application server can be instructed to flush its cache and re-read the required logic definition from the run-time repository.

Flushing the PEAR cache

You can also flush the PEAR cache using Cogility Insight. See "Manage" on page 95.

To flush the PEAR cache:

1. Run Cogility Message Traffic Viewer.

See “Running Cogility Message Traffic Viewer” on page 119.

2. Click **Load**, navigate to <DCHOME>\MB\Config, select **RefreshPearMessage.msg** and click **Open**.
3. With the message selected, click **Inspect** (this starts Cogility Message Editor).
4. In Cogility Message Editor's tree view, select **Subject:<Refresh_pear>**.
5. In the key field, select **refreshPearCache**; in the input field, enter **true** and click **Accept Single Changed Value**.

   Notice that the value is set to true by default. This does not have any bearing on the other settings, and it may remain set to true while another key’s value is also true or otherwise defined.
6. Close Cogility Message Editor.
7. In Cogility Message Traffic Viewer, click **Re-publish To Destination** to publish the message.

Configuration

Ordinarily, configuration parameter values are read from the embedded configuration files deployed with the model application. You can reload those values during run-time either by sending a message to the application server as described in “Reload configuration message” on page 36 or with a menu selection in either Cogility Action Pad (see “Model configuration” on page 77) or Cogility Insight (see “Cogility Functions” on page 93).

Reload configuration message

Cogility Studio defines configuration parameters that control the behavior of Cogility Manager underneath the J2EE application server. In addition, the system model itself uses configuration parameters that control the behavior of the model in execution. The value of these parameters is loaded into memory when the application server is started. You can send a message to the application server to change the value of these parameters and load them again without shutting down the application server.

The message allows you to tell the model application to read the configuration settings from the external source files where you set them prior to deployment. You can also reset the <DCHOME> environment variable if needed.

You can also reload configuration parameters while you are working with your model in Cogility Modeler. See “Reloading configuration parameters into Cogility Modeler” on page 28 of the guide, *Modeling with Cogility Studio*.

Reloading configuration parameters to the application server

To refresh the configuration parameters when the application server is running:

1. Edit your customConfigurations.txt or project configuration files, as needed and set any debugging parameters to false.

   See “Customizing installation configuration” on page 13.

1. Run Cogility Message Traffic Viewer.

   See “Running Cogility Message Traffic Viewer” on page 119.
2. Click **Load**, navigate to <DCHOME>\MB\Config, select **RefreshPearMessage.msg** and click **Open**.
3. With the message selected, click **Inspect** (this starts Cogility Message Editor).
4. In Cogility Message Editor's tree view, select **Subject:<Refresh_pear>**.

5. In the key field, select **refreshConfigParams**; in the input field, enter **true** and click **Accept Single Changed Value**.
6. If you need to update the application server for the location of the external source files for the previous step, in the **newDCHOME** field, enter the path to the new location of <DCHOME> and click **Accept Single Changed Value**.

   <DCHOME> is the location where you installed Cogility Studio.
7. If you changed the value for <DCHOME> in step 6, above, you must also change the value for the **newOEMHOME** field; enter the same path to <DCHOME> with **\MB** appended.

   See the example at "RefreshPearMessage.msg file" on page 35.
8. Close Cogility Message Editor.
9. In Cogility Message Traffic Viewer, click **Re-publish To Destination** to publish the message.

Executable model verification

When you need to verify the deployed model before you either push a revised model into execution or run the deployed model, you can compare the the extant model with the revised model and view a history of model pushes.

Deployed model comparison report

As the model changes during design, it is useful to determine how the executable behavior in the model is different from the deployed version in the database. The deployed model comparison report looks at certain executable artifacts and notes deviations from the model version, if any. Artifacts covered include:

- State machines
- Method bodies
- Inbound web services
- Transformation maps
- Opaque transformation maps
- Parallel transformation chains
- Serial transformation chains
- Legacy information model (LIM)
- LIM execute statement
- LIM select statement
- LIM SQL block
- Web service handler
- Web service handler chain

See the guide, *Modeling with Cogility Studio* for more information about these artifacts.

Generating the deployed model comparison report

To generate the deployed model comparison report:

1. In Cogility Modeler's tree view, select the model container.

2. From the **Selection** menu, choose **Actions > Generate Deployed Model Comparison Report**.
3. In the dialog, select the deployment container and click **Use Deployment**.

   You must specify to which deployment you are comparing the model. See "Configuration Parameters" on page 28 of the guide, *Modeling with Cogility Studio*.
4. In the consistency view, click the **Push Action** tab.

   The deployed model comparison report describes the model objects and their changes.

Deployment history

As the model is deployed and re-deployed, the execution database will keep track of model deployment history. The different types of pushes that deploy metadata to the database will be tracked, including full/incremental push and single artifact push. For every Full push history artifact, the system tracks all the different EJBs that were deployed with the push

See "Deployment History and Artifacts" on page 113.

5 Audit and Debug

To assist with database clean-up and auditing, Cogility Studio includes the facilities described in this section.

Web service auditing

You can configure Cogility Studio to report audit trail information during web service execution. This lets you monitor calls to both inbound and outbound web services for specific data. Auditing works for only top-level web service calls, so the data from an inbound web service called from within another inbound web service is not logged. Nor is the data audited from inbound web services called from Cogility Insight, because this process uses the same mechanism as a nested inbound web service.

To enable web service auditing, you set the following configuration parameters to true in your customConfigurations.txt file:

- `com.ceira.webservice.audit.ows=true` for outbound web service auditing
- `com.ceira.webservice.audit.iws=true` for inbound web service auditing
- `com.ceira.webservice.audit.fws=true` for fixed endpoint inbound web service auditing

The audit data is stored in the run-time repository (PEAR) in tables; you retrieve the data using SQL queries on these tables:

- PEAR_OWSAUDITTRAIL for outbound web service data
- PEAR_IWSAUDITTRAIL for inbound web service data
- PEAR_FWSAUDITTRAIL for fixed endpoint inbound web service data

The tables include the attributes described in the following sections.

Outbound web service audit data

The following table describes the data reported from an outbound web service call.

Attribute name	Description	Type	Size
A_WEBSERVICE_NAME_	Fully qualified model name	Varchar2	255
A_ENDPOINT_	Endpoint URL	Varchar2	1023
A_REQUESTMESSAGE_	SOAP request message	BLOB	3999
A_RESPONSEMESSAGE_	SOAP response message	BLOB	3999
A_STARTTIME_	Java timestamp of execution start	Number	22

Attribute name	Description	Type	Size
A_ENDTIME_	Java timestamp of execution end	Number	22
A_EXTRADATA1_	User defined data (see below)	String	255
A_EXTRADATA2_	User defined data (see below)	String	255
A_EXTRADATA3_	User defined data (see below)	String	255

Inbound web service audit data

The following table describes the data reported from an inbound web service call.

Attribute name	Description	Type	Size
A_WEBSERVICE_NAME_	Fully qualified model name	Varchar2	255
A_DEPLOYMENT_NAME_	Fully qualified model name This is different for fixed endpoint inbound web services, see Fixed endpoint webservice audit data, below	Varchar2	255
A_REQUESTMESSAGE_	SOAP request message This is different for fixed endpoint inbound web services, see Fixed endpoint webservice audit data, below	BLOB	3999
A_RESPONSEMESSAGE_	SOAP response message This is different for fixed endpoint inbound web services, see Fixed endpoint webservice audit data, below	BLOB	3999
A_STARTTIME_	Java timestamp of execution start	Number	22
A_ENDTIME_	Java timestamp of execution end	Number	22
A_EXTRADATA1_	User defined data (see below)	String	255
A_EXTRADATA2_	User defined data (see below)	String	255
A_EXTRADATA3_	User defined data (see below)	String	255

User defined data

You specify which webservice is being audited by specifiying the webservice in a customConfigurations parameter. The example below specifies that a GetInfoByCust webservice is to be audited.

To use the EXTRADATA columns, you designate the attributes that map to each EXTRADATA field in your customConfigurations.txt file. For example, the following configuration settings specify that the `extraData1` and `extraData2` fields map to the `publicLoanId` and `intranetUserId` request attributes, respectively. Also, the `extraData3` field is mapped to the `resultCode` response attribute. The values for these attributes are retrieved with xpath expressions and are reported to the PEAR_IWSAUDITTRAIL table.

```
com.ceira.webservice.audit.isw.named.GetInfoByCust=auditsOne

com.ceira.webservice.audit.iws.auditsOne.num.inattr=1

com.ceira.webservice.audit.iws.auditsOne.inattr.1.name=extraData1
com.ceira.webservice.audit.iws.auditsOne.inattr.1.maxlen=255
com.ceira.webservice.audit.iws.auditsOne.inattr.1.type=String
com.ceira.webservice.audit.iws.auditsOne.inattr.1.xpath=//publicLoanId/
child::()

com.ceira.webservice.audit.iws.auditsOne.inattr.2.name=extraData2
com.ceira.webservice.audit.iws.auditsOne.inattr.2.maxlen=255
com.ceira.webservice.audit.iws.auditsOne.inattr.2.type=String
com.ceira.webservice.audit.iws.auditsOne.inattr.2.xpath=//intranetUserId/
child::()

com.ceira.webservice.audit.iws.auditsOne.num.outattr=1

com.ceira.webservice.audit.iws.auditsOne.outattr.1.name=extraData3
com.ceira.webservice.audit.iws.auditsOne.outattr.1.maxlen=255
com.ceira.webservice.audit.iws.auditsOne.outattr.1.type=String
com.ceira.webservice.audit.iws.auditsOne.outattr.1.xpath=//resultCode/
child::()
```

For outbound web services, the configuration settings are the same, except the `iws` path fragment is replaced with `ows`, as in the following example:

```
com.ceira.webservice.audit.osw.named.SetInfoByCust=auditsTwo

com.ceira.webservice.audit.ows.auditsTwo.num.inattr=1

com.ceira.webservice.audit.ows.auditsTwo.inattr.1.name=extraData1
com.ceira.webservice.audit.ows.auditsTwo.inattr.1.maxlen=255
com.ceira.webservice.audit.ows.auditsTwo.inattr.1.type=String
com.ceira.webservice.audit.ows.auditsTwo.inattr.1.xpath=//taxpayerId/
child::()
```

Fixed endpoint webservice audit data

Previous versions of Cogility Studio used a fixed endpoint inbound web service deployment. That is, all inbound web services were called at the same endpoint, so the request attribute had to have a parameter that designated that endpoint (String_1), then an additional parameter to hold the request attributes (String_2). Therefore, the meaning of the audit data for the DEPLOYMENT_NAME, REQUESTMESSAGE, and RESPONSEMESSAGE attributes is different for fixed endpoint inbound web services, as described in the following table.

Attribute name	Description	Type	Size
A_WEBSERVICE_NAME_	Fully qualified model name	Varchar2	255
A_DEPLOYMENT_NAME_	Fixed endpoint for the inbound web service.	Varchar2	255
A_REQUESTMESSAGE_	String_2 input parameters XML holding the request attributes	BLOB	3999
A_RESPONSEMESSAGE_	Returned output parameters XML	BLOB	3999

Attribute name	Description	Type	Size
A_STARTTIME_	Java timestamp of execution start	Number	22
A_ENDTIME_	Java timestamp of execution end	Number	22
A_EXTRADATA1_	User defined data (see below)	String	255
A_EXTRADATA2_	User defined data (see below)	String	255
A_EXTRADATA3_	User defined data (see below)	String	255

Also, the configuration settings need to specify the parameters within the request message that designate the request attributes. To specify the publicLoanId, internetUserId, and resultCode (from the previous example under "User defined data" on page 40), the configuration settings are as follows:

```
com.ceira.webservice.audit.fsw.named.GetAddress=auditsThree

com.ceira.webservice.audit.fws.auditsThree.num.inattr=2

com.ceira.webservice.audit.fws.auditsThree.inattr.1.name=extraData1
com.ceira.webservice.audit.fws.auditsThree.inattr.1.maxlen=255
com.ceira.webservice.audit.fws.auditsThree.inattr.1.type=String
com.ceira.webservice.audit.fws.auditsThree.inattr.1.xpath=//params/
@publicLoanId

com.ceira.webservice.audit.fws.auditsThree.inattr.2.name=extraData2
com.ceira.webservice.audit.fws.auditsThree.inattr.2.maxlen=255
com.ceira.webservice.audit.fws.auditsThree.inattr.2.type=String
com.ceira.webservice.audit.fws.auditsThree.inattr.2.xpath=//params/
@intranetUserId

com.ceira.webservice.audit.fws.auditsThree.num.outattr=1

com.ceira.webservice.audit.fws.auditsThree.outattr.1.name=extraData3
com.ceira.webservice.audit.fws.auditsThree.outattr.1.maxlen=255
com.ceira.webservice.audit.fws.auditsThree.outattr.1.type=String
com.ceira.webservice.audit.fws.auditsThree.outattr.1.xpath=//params/
@resultCode
```

Completed state machines

State machine object instances are created during execution. After a state machine is complete, the data becomes just an audit trail. Unless it is needed for debugging, this data can be removed from the database. Otherwise, over time, an accumulation of completed state machines can degrade database performance. A batch file with a deletion routine is provided with Cogility Studio; you run the batch file after providing its configuration settings in your `customConfigurations.txt` file.

There are two utilities for deleting completed state machine instances from the database. These are located in the `<DCHOME>\scripts\Poal` directory and named as follows:

- `DeleteCompletedStateMachines.bat` - an interactive routine that runs in standard output and prompts the user to continue.
- `BackgroundDeleteCompletedStateMachines.bat` - a background process that reports to a log file.

Both will delete the state machine instances that conform to the values of the configuration settings as follows:

- `delete.SM.olderThanDays=5`
- `delete.SM.olderThanHours=0`
- `delete.SM.olderThanMinutes=0`
- `delete.SM.transactionBatchSize=100`

The first three define how old the completed state machine must be. The last defines the number of state machines which will be deleted per transaction.

Deleting completed state machines

To remove completed state machines:

1. Edit your customConfigurations.txt as described in "Customizing installation configuration" on page 13. Insert the following settings.

   ```
   delete.SM.olderThanDays=5
   delete.SM.olderThanHours=0
   delete.SM.olderThanMinutes=0
   ```

   Set any of these values to determine the age of those state machines you want to remove.

   ```
   delete.SM.transactionBatchSize=100
   ```

   Set this value to delete state machines in batches of the given size.
2. Run either of the following batch files.

   These routines will read the configuration parameters you set in step 1, above and delete the state machine instances accordingly.

   a. This routine first reports the number of state machine instances to remove and prompts you to continue:

      <DCHOME>\scripts\Poal\DeleteCompletedStateMachines.bat.

      The deleted state machine instances are reported to standard output.

   b. This routine does not report to standard output, nor does it prompt you to continue:

      <DCHOME>\scripts\Poal\BackgroundDeleteCompletedStateMachines.bat

      A log file with details of the state machine instances deleted is generated and saved at <DCHOME>\scripts\Poal\DeletedStateMachines.log.

6 Scaling Your Installation

In a shared-load, multi-application server environment, you can migrate between versions of Cogility Studio or between model versions without shutting down all of the application servers at once. In this scenario the execution environment is updated in a stepwise fashion, addressing one application server at a time, allowing business operations to continue. The process includes duplicating the application's fixed schema and configuring the environment to utilize it until the update completes and is verified.

Migrating and upgrading multiple application servers

This scenario assumes an environment running three application servers of the same type (A, B and C) with the same version of Cogility Studio installed at the location of each server. It also assumes that traffic is intelligently routed to each server. For the instructions below, <`DCHOME`> is the location where you installed Cogility Studio.

To migrate and upgrade:

1. Determine if the run-time database has a duplicate fixed schema.

   If it does, the tables in the database will be prefixed with `PXXX`. You must remove the duplicate fixed schema using the removal utility for the version of Cogility Studio under which the duplicate schema was created.

   a. Edit the file, <DCHOME>\MB\Config\customConfigurations.txt to include the following entry:

   ```
   com.cogility.pear.useDuplicatedFixedSchema=false
   ```

   b. Navigate to <DCHOME>\MB\scripts\Launcher

   c. Run the script, **Run_FixedSchemaDupRemover.bat**.

2. On the machine running application server A, do the following:

   a. Stop traffic to server A.

   b. Stop Cogility WatchDog on server A (if applicable).

   c. Stop application server A.

3. Run the following script:

   <DCHOME>\MB\scripts\Launcher\runFixedSchemaDuplicator.bat

   This script creates a duplicate fixed schema on the run-time repository.

4. On the machine running application server B, do the following:

   a. Stop traffic to server B.

   b. Stop Cogility WatchDog on server B (if applicable).

   c. Stop application server B.

   d. Edit the file, <DCHOME>\MB\Config\customConfigurations.txt to include the following entry:

      ```
      com.cogility.pear.useDuplicatedFixedSchema=true
      ```

   e. Restart application server B.
   f. Restart Cogility WatchDog on server B (if applicable).
   g. Resume traffic to server B.
   h. Start Cogility Action Pad to verify that server B is now using the duplicate fixed schema.

      If server B is using the duplicate fixed schema, the following will appear in the Cogility Action Pad console upon start-up:

      ```
      [MESSAGE][PEAR]================================================
      [MESSAGE][PEAR]==== WARNING WARNING WARNING WARNING WARNING ====
      [MESSAGE][PEAR]====                  ====
      [MESSAGE][PEAR]==== Duplicated Fixed Schema is in use ====
      [MESSAGE][PEAR]====                  ====
      [MESSAGE][PEAR]==== WARNING WARNING WARNING WARNING WARNING ====
      [MESSAGE][PEAR]================================================
      ```

5. Repeat step 4, previous, for application server C (and each additional application server).
6. On the machine running application server A, do the following:
   a. Install the new release of Cogility Studio and follow the migration instructions provided in a separate document (if necessary).
   b. Run Cogility Modeler, import, load, update (if necessary) and turn over the model.
   c. Start application server A.
   d. In Cogility Modeler, from the **Actions** menu, select **Push**, check the boxes to **Force Recompile of Action Semantics** and **Force Push of App Server Artifacts,** select the deployment for application server A and click **Use Deployment**.

      You may have to update the deployment for application server A prior to executing the push. See also, “Force action semantics recompilation” on page 30.
   e. Start Cogility WatchDog on server A (if applicable).
   f. Resume traffic to server A.
7. On the machine running application server B, do the following:
   a. Stop traffic to server B.
   b. Stop Cogility WatchDog on server B (if applicable).
   c. Stop application server B.
   d. Install the new release of Cogility Studio and follow the migration instructions provided in a separate document (if necessary).
   e. Run Cogility Modeler, import, load, update (if necessary) and turn over the model.
   f. Start application server B.
   g. In Cogility Modeler, from the **Actions** menu, select **Push to AppServer**, select the deployment for application server B and click **Use Deployment**.
   h. Start Cogility WatchDog on server B (if applicable).
   i. Resume traffic to server B.
8. Repeat step 7, previous, for application server C (and each additional application server).
9. Once you are satisfied that the new release is functioning properly, remove the duplicate schema with the following script:

   <DCHOME>\MB\scripts\Launcher\Run_FixedSchemaDupRemover.bat

Note that the script must be run from the environment where the duplicate schema was created; that is, the previous version of Cogility Studio.

This completes the deployment on all machines.

7

Project Configuration Editor

You may work with several different models, and each may utilize different source data or a different application and database configuration. This information may be collected in a specific project folder. This chapter describes how to configure your model to work with specific *project* data. Source code, Java libraries, and project-specific configuration information are examples of project data. Note that these are not model artifacts, yet they are used in building model artifacts or deploying the model.

The Cogility Project Configuration Editor lets you work with your source data and assign it to a specific project. When you want to use the source data in your model artifacts, it is available in the project classpath. See "Classpath" on page 54.

You can also set up configuration files specifically for the project. When you open the model associated with the active project, the project configuration files' settings supercede the installation's configuration files' settings.

Note that the Cogility Project Configuration Editor runs independently of the other applications in Cogility Studio. It is not necessary to have a model opened in Cogility Modeler in order to create a project. You may associate more than one model with a particular project.

Running Cogility Project Configuration Editor

To run the Cogility Project Configuration Editor:

1. From the **Start** menu, select **All Programs** > **Cogility Studio > Project Configuration Editor**.

Projects

When you create a project, a folder labeled with the project's name is created in the `DCHOME/Engagements` folder. The project folder serves as a central location for all project-specific artifacts including source code, third party JAR files, configuration files and scripts. If project-related code or configuration files are maintained in a different location, perhaps controlled by a CM system, they may be copied into the project folder using Cogility Project Configuration Editor. It is necessary to add these artifacts to the project in order for Cogility Modeler and the deployed application to utilize them. Code or JAR files get added to the class path for Cogility Modeler, the application server, and other Cogility applications. Configuration files are cached when any of the aforementioned applications is started.

When the project is created, the following subdirectories and files are located in the project directory:

- **3p-lib** - for any third party libraries required by the project-specific source code. See "Importing JAR files" on page 55.

- **bin -** This is the location where the editor writes action semantic batch scripts. This is also true regarding the scripts (used for windows installs) folder. The generation of action semantic batch scripts is performed by selecting the appropriate right-button menu in the left-hand pane of the editor.

  There are two menu picks for generating batch scripts.

  Included in the bin folder are two files for creating a project related jar from the project source:

  `build.xml`

  `compile_project.sh`
- **Config** - for project configuration files. These communicate project data to Cogility Modeler and Cogility Manager. If they are changed or deleted from the project directory and need to be replaced, you can do this through the Project field menu. You can copy the updated files into the project Config/Files location, remove files from this location or refresh existing files from their original location. Any .txt file in the project/Config/Files folder will be read by Cogility Modeler or Cogility Manager at startup and the values cached. See "Rewriting configuration artifacts" on page 52.
  - **metadata.xml** - this file describes your project's source files and directory structure. It is a full description of all of the information that Cogility Project Configuration Editor knows. It includes not only the names of the various imported entities, but also the location from which each was originally imported.
  - **Files** - a directory for your project's configuration files. These are configuration files that pertain to the project specifically, not the Cogility Studio installation. The Cogility Studio configuration files are described in "Configuration files" on page 13.
- **lib** - for compiled JAR files not found in the 3p-lib directory. See "Importing source trees" on page 56.
- **model** - for exported project models. There is no specific functionality build in to the tool to support this folder.
- **ocl** - for project related .ocl files. When generating action semantic batch scripts, you are prompted to specify which .ocl file to execute. This file is copied to this location and the batch script references the .ocl file from this location.
- **src** - for your project's source files. In the event that your project files may have been changed manually, you can find out by scanning your project directory for changes. See "Scanning for project files for changes" on page 53. When the project's source files have changed significantly, you can re-import the source files you imported into your project. See "Refreshing all imports" on page 53. Doing so first deletes the entire directory structure you have established in your project then rewrites it, so use this operation with discretion.
  - **SetupEngagementClassPath** (.sh or .bat) - this utility describes the classpath for your project. The .sh version is for Unix, the .bat for Windows. It adds the imported JAR files and source trees into the classpath. This utility is required by all of the Cogility scripts (those for Cogility Modeler and Cogility Manager) that need access to the engagement-specific code.
  - **Projects** - a directory for your project's source trees, for example a source tree named `ProjectA.Source.com.cogility`. See "Importing source trees" on page 56.
- **scripts** - Project-related action semantics scripts for Windows environments. These include supporting files to demonstrate how to execute action semantics using a .bat file. When using this tool to generate action semantic batch scripts, the resultant scripts will be created in this location.

  There are two menu picks for generating batch scripts.

  Included in the scripts folder are two files for creating a project related jar from the project source:

  `build.xml`

```
compile_project.bat
```

The compile_project script is used to compile project source code, creating a single project .jar file. If the script completes without error, the .jar file is written to the DCHOME\Engagements\projectName\3p-lib folder. The existence of this file in this location will not automatically insert it into the

Execution of this script is performed outside of the PCE, using a command window.

The second file build.xml, is an ANT script. Apache Ant is installed by Cogility in DCHOME/Common/ThirdParty/apache-ant

In order to execute the ANT script however, a JDK must be installed and a system environment variable named JAVA_HOME must be defined.

Other requirements include that the source in question must be defined in the Active Project and the file-structure of the src folder must conform to the following:

```
DCHOME/Engagements/projectName/src/srcProjectName/Source
```

Multiple srcProjectName folders can exist. However, source from each is bundled into a single jar file.

Creating a new project

To create a new project:

1. In the **Project** field, right-click and select **Create New Project**.
2. In the input dialog's name field, enter a project name and click **OK**.

Note: Project names may not include spaces. Cogility Project Configuration Editor automatically replaces spaces in project names with underscores before saving the project folder name. You have the option to cancel the project creation for the name.

The project name appears in the Project field. The output field reports the subdirectories and files created.

Removing projects

To remove a project:

1. If the project is the active project, make a different project the active project.

   See "Active project" on page 52. If you remove the active project, configuration information for the models associated with that project will remain in memory until the next time you run Cogility Modeler. If you start Cogility Modeler with the active project folder deleted, no project-specific configuration information will be loaded.

   In the Project field, select the project, right-click, and select Remove Project From List. This action will remove the project from the list but does not delete it from the file system.

Restoring projects

Only projects that were removed using the Project Configuration Editor can be restored using the following procedure.

To restore a project:

1. Right-click in the Project field.
2. Select **Restore 'Removed' Project(s) To List...**
3. Use the **Select Project(s) to Restore** dialog to select the project you wish to restore.
4. Click **Restore**.

Renaming projects

To rename a project:

1. If the project is the active project, make a different project the active project.
2. In the Project field, select the project you wish to rename, right-click and select **Rename project….**
3. In the input dialog, enter the new name of the project and click **OK**.

Note: If the project contains files that reference other files by their path, you will need to edit the files manually to reflect the new path.

Active project

The active project is the project for which models in Cogility Modeler are configured. Its sources and libraries are added to the classpath and its configurations are cached, thus making them accessible to model artifacts. When you run Cogility Modeler, the last active project in Cogility Project Configuration Editor serves as the project for the model in the current con. Before you create a model in Cogility Modeler, be sure the project with which you want to associate the model is the active project in Cogility Project Configuration Editor.

The active project affects both modeling and run time. When you make a project the active project, both the DCHOME/Config/Engagement.bat and setenv.sh are updated with the name of the active project.

Activating a project

To make a project the active project:

1. In the **Project** field, select the project, right-click and select **Make project name the Active Project**.

   The project name appears in the Project field in bold blue font and the active project setup file is updated with the new project.

Rewriting configuration artifacts

To rewrite configuration artifacts:

1. In the **Project** field, select the project whose metadata.xml and SetupEngagementClasspath files you want to update, right-click and select **Rewrite *project name* Config Artifacts**.

   Note that you can do this with any project listed, not just the active project.

Scanning for project files for changes

To scan the project for changes:

1. In the **Project** field, select the project, right-click and select **Scan *project name* for Changes**.

   In the output field, Cogility Project Configuration Editor reports the changes discovered in the source directory: new files, changed files and deleted files.

   If any changes to the project directory have occurred, they are reported in the output field.

Reporting project file import locations

To report the project files original import location:

1. In the **Project** field, select the project, right-click and select **Report all *project name* original import locations**.

   In the output field, Cogility Project Configuration Editor reports the source directories of the files imported into the project.

Refreshing all imports

Performing this operation first deletes the entire directory structure you have established in your project then rewrites it with the files updated at their source locations. Before you refresh the imports, be sure the location is accessible with **Report all original import locations**, see "Reporting project file import locations" on page 53.

To refresh all imports:

1. In the **Project** field, select the project, right-click and select **Refresh all *project name* imports from original locations**.
2. If you are sure you want to refresh the project files, click **yes** in the dialog.

   A refresh summary is reported in the output field.

Action semantics batch files

Action semantics script files may be processed as .bat files in Windows or .sh files in Unix. You may have a script that you want to run as a repeated process automatically in the background, or you want to make your action semantics scripts otherwise available to the operating system. To do so, you create an action semantics script batch file using the procedures below.

- "Creating action semantics script batch files" on page 54 describes the procedure for setting up a batch file for an action semantics script that does not publish a JMS message.
- "Creating script batch files that publish JMS messages" on page 54 describes the procedure for setting up a batch file for an action semantics script that publishes a JMS message. To do so, the script must know the properties for the application server for the project under which you are creating this batch file. Cogility Project Configuration Editor creates a batch file that retrieves these properties based on the path to configuration properties for the project.

In creating the batch file for the action semantics script, Cogility Project Configuration Editor copies the action semantics script from its original location to the folder, DCHOME/Engagements/<project name>/ocl. Note, this is a copy of the original file, not an import, and it will not be recopied when you refresh imports (see "Refreshing all imports" on page 53).

Next, Cogility Project Configuration Editor creates a batch file that runs the script in the folder DCHOME/Engagements/<project name>/scripts. The file will be named the same as the original script, except it will have a .bat or .sh extension.

You can copy the batch file into any directory on your system and run it.

A log file is created for each execution of the script. Log files are written to the DCHOME/ Engagements/projectName/log folder. The name of the log file is displayed in the console window in which the batch script is executed.

Before you create the batch file, you should verify that the script compiles and executes using Cogility Action Pad. See "Cogility Action Pad" on page 69. Also, the model to which the script refers must be pushed into execution. See "Pushing the model into execution" on page 326 of the guide, *Modeling with Cogility Studio*.

You can run the script programatically from your model's action semantics. See the guide, *Using Action Semantics with Cogility Studio*.

Creating action semantics script batch files

To create an action semantics script batch file:

1. In the Project view, select the project where you want to create the batch file.
2. Right-click and select **Generate OCL batch scripts for <project name>**.
3. Navigate to the location of the action semantics script file and click **Open**.

Creating script batch files that publish JMS messages

To create a script batch file that publishes JMS messages:

1. In the Project view, select the project where you want to create the batch file.
2. Right-click and select **Generate OCL batch scripts for <project name> and <application server>**.
3. Navigate to the location of the action semantics script file and click **Open**.

Defining the preferred JDBC Driver

Cogility Studio supports both Oracle 9i and 10g configurations and installs both drivers in the DCHOME/MB/3p-lib folder. However, when defining the classpath for Cogility applications, OracleJDBC.jar is reference. To define which driver to use, the editor makes a copy of either the 9i or 10g driver file and renames it to OracleJDBC.jar. Oracle 10g is the default driver.

To define the preferred Oracle JDBC driver:

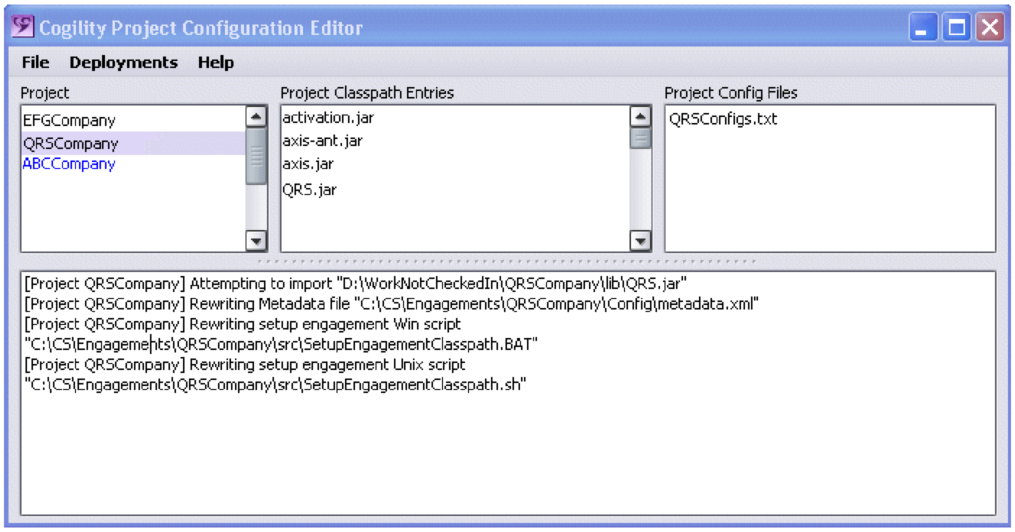
1. In the Project view, right-click and select **Define preferred Oracle JDBC…**
2. In the input dialog, select the preferred driver from the drop-down list and click **OK**.

Classpath

The project classpath includes entries for project specific source files that you want added to your project. You can import both JAR files and source trees, and delete these entities if need be. You can refresh imported files from their sources, and you can change the classpath root for extant source trees.

JAR files

Imported JAR files are imported into the 3p-lib subdirectory. Thereafter they are available to the model and Cogility scripts that access the model's source code. Jar files are shown in black, directories are shown in blue; jar files or directories that do not exist are shown in red.



```
[Project QRSCompany] Attempting to import "D:\WorkNotCheckedIn\QRSCompany\lib\QRS.jar"
[Project QRSCompany] Rewriting Metadata file "C:\CS\Engagements\QRSCompany\Config\metadata.xml"
[Project QRSCompany] Rewriting setup engagement Win script
"C:\CS\Engagements\QRSCompany\src\SetupEngagementClasspath.BAT"
[Project QRSCompany] Rewriting setup engagement Unix script
"C:\CS\Engagements\QRSCompany\src\SetupEngagementClasspath.sh"
```

Figure 7-3: Imported JAR file

Importing the file accomplishes the following:

- The file is copied to the 3p-lib folder.
- The SetupEngagementClasspath is updated to include the path.
- The metadata.xml file is updated to include the location from where the .jar file was copied.

Importing JAR files

To import a JAR file:

1. In the **Project** field, select the project.

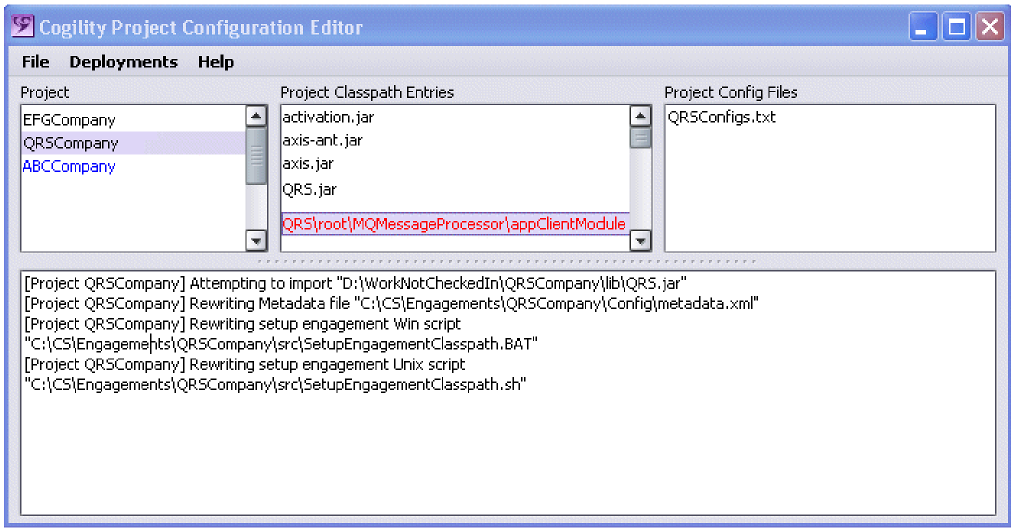
   Note that you can import into any project listed in the Project field, not just the active project.
2. In the **Project Classpath Entries** field, right-click and select **Import new .JAR**.
3. Navigate to the location of the JAR files, select them and click **Open**.

   To select multiple files, hold down the Ctrl key while selecting

Source trees

Imported source trees are imported into the src subdirectory. Subsequently they may be manually compiled into JAR files and located in the lib directory. The SetupEngagementClasspath utility looks for each imported source tree, and if absent, adds the correspondingly named JAR to the classpath out of the lib directory.

You can only import a source tree once. A second attempt to import the source tree results in an error.



```
[Project QRSCompany] Attempting to import "D:\WorkNotCheckedIn\QRSCompany\lib\QRS.jar"
[Project QRSCompany] Rewriting Metadata file "C:\CS\Engagements\QRSCompany\Config\metadata.xml"
[Project QRSCompany] Rewriting setup engagement Win script
"C:\CS\Engagements\QRSCompany\src\SetupEngagementClasspath.BAT"
[Project QRSCompany] Rewriting setup engagement Unix script
"C:\CS\Engagements\QRSCompany\src\SetupEngagementClasspath.sh"
```

Figure 7-4: Imported source tree

The source tree is listed in the Project Classpath Entries field in red.The metadata.xml file is updated with the location from where the source was imported, and the SetupEngagementClasspath file is updated.

Importing source trees

To import a source tree:

1. In the **Project** field, select the project.
2. In the **Project Classpath Entries** field, right-click and select **Import new source tree**.
3. Navigate to the top of the source tree and click **Open**.

Deleting entities

Deleting files in this manner removes the files from the project folder, but not from the location from which they were imported.

To delete a JAR file or source tree:

1. In the **Project** field, select the project.
2. In the **Project Classpath Entries** field, select the JAR files or source trees, right-click and select **Delete code project** (for source trees) or **Delete code archive** (for JAR files).

   To select multiple files, hold down the Ctrl key while selecting
3. If you are sure you want to delete the entity, click **Yes** in the dialog.

   The metadata.xml and SetupEngagementClasspath files are updated appropriately.

Refreshing code

To re-import the JAR file or source tree from the source directory:

1. In the **Project** field, select the project.
2. In the **Project Classpath Entries** field, select the JAR files or source trees, right-click and select **Refresh code project** (for source trees) or **Refresh code archive** (for JAR files).

To select multiple files, hold down the Ctrl key while selecting.

3. If you are sure you want to refresh the entity, click **Yes** in the dialog.

Changing the classpath root

If you want to change the way a source tree is represented in your project, you can change the classpath root.

To change the classpath root:

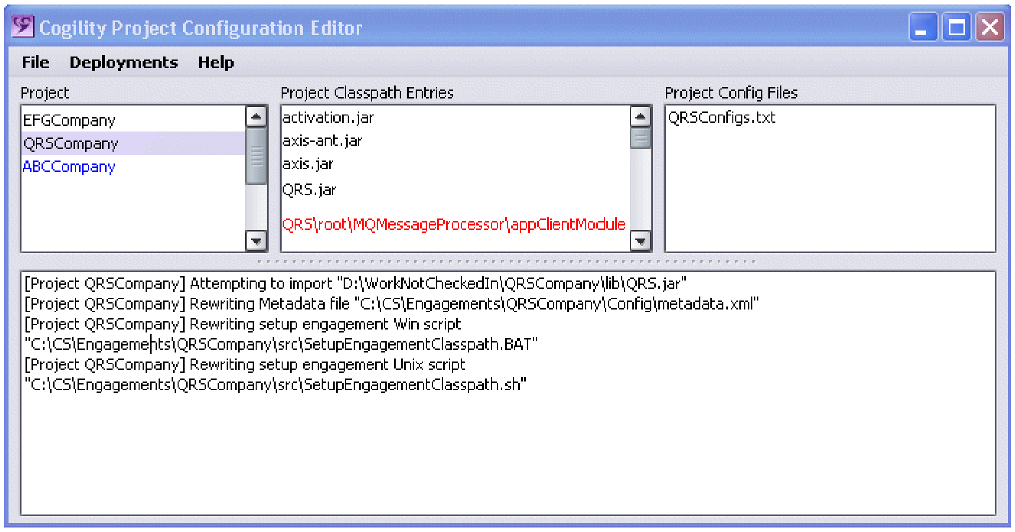
1. In the **Project** field, select the project.
2. In the **Project Classpath Entries** field, select the source tree, right-click and select **Change classpath root**.
3. Navigate to the location where you want to establish the classpath root and click **Open**.

Configuration files

Project-specific configuration settings are separate from installation-specific configuration settings. The installation-specific configurations are described in the files of the `<DCHOME>\MB\Config\Files` folder and in the `customConfigurations.txt` file. These files are always read last, so their settings supercede all others. Your project specific configuration files are read prior to the installation configuration files and their settings are intended to be completely separate from the installation specific settings. For more information about installation specific custom configurations, see "Customizing installation configuration" on page 13.

You can import, delete, and refresh project configuration files. Any project-specific configurations that apply to your model may be listed in custom configurations files. To make them available to your model, you import them into your project. Configuration files contain settings as described in "Configuration" on page 13 of the guide, *Model Deployment & Execution in Cogility Studio*.

The configuration file is listed in the Project Config Files field in black .


Cogility Project Configuration Editor
File
Deployments
Help
Project
Project Classpath Entries
Project Config Files

Project	Project Classpath Entries	Project Config Files
EFGCompany	activation.jar	QRSConfigs.txt
QRSCompany	axis-ant.jar	
ABCCompany	axis.jar	
	QRS.jar	
	QRS\root\MQMessageProcessor\appClientModule	

```
[Project QRSCompany] Attempting to import "D:\WorkNotCheckedIn\QRSCompany\lib\QRS.jar"
[Project QRSCompany] Rewriting Metadata file "C:\CS\Engagements\QRSCompany\Config\metadata.xml"
[Project QRSCompany] Rewriting setup engagement Win script
"C:\CS\Engagements\QRSCompany\src\SetupEngagementClasspath.BAT"
[Project QRSCompany] Rewriting setup engagement Unix script
"C:\CS\Engagements\QRSCompany\src\SetupEngagementClasspath.sh"
```

Figure 7-5: Project configuration files

Importing configuration files

To import project configurations files:

1. In the **Project** field, select the project.
2. In the **Project Config Files** field, right-click and select **Import New Config File(s)**.
3. Navigate to the location of the configuration files, select them and click **Open.**

   To select multiple files, hold down the Ctrl key while selecting.

Deleting configuration files

To delete a configuration file from your project:

1. In the **Project** field, select the project.
2. In the **Project Config Files** field, select the file(s), right-click and select **Delete imported Config File**.

   To select multiple files, hold down the Ctrl key while selecting.
3. If you are sure you want to delete the file, click **Yes** in the dialog.

Refreshing configuration files

To re-import the configuration file from the source directory:

1. In the **Project** field, select the project.
2. In the **Project Config Files** field, select the file(s), right-click and select **Refresh imported Config File**.

   To select multiple files, hold down the Ctrl key while selecting.
3. If you are sure you want to re-import the file, click **Yes** in the dialog.

8 Remote Deployment

Cogility Studio supports deployment of the model application to an application server located on a second machine. Both the location from which the model is deployed (the source machine) and the location to which the model is deployed (the target machine) must have the application server installed. The application server on the target machine must be running during remote deployment. The same Cogility Studio release must be installed on both the source and target machines. The following sections describe various aspects of remote deployment.

Deploying to a remote server

To perform a remote deployment:

1. On both the source and target machines, define the Application Server specific variables. See "Application Server Variables" on page 59.
2. On the source machine, be sure the hosts file includes an entry for the target machine. See "Hosts file" on page 61
3. On the target machine, create a project for the deployed model and make it the active project. See "Project for the deployed model" on page 60
4. Copy the configuration files from the source machine to the target machine. See "Configuration files" on page 61
5. On the source machine, in Cogility Modeler, edit the application server deployment artifact. See "Remote deployment server artifact" on page 62
6. On the source machine, in Cogility Modeler, edit the database deployment artifact. See "Database artifact" on page 66.
7. On the source machine, in Cogility Modeler, deploy the model to the target machine. See "Pushing the model into execution" on page 326 of the guide, *Modeling with Cogility Studio*.

Application Server Variables

Cogility Studio refers to several environment variables that describe the location of the application servers involved in the deployment. The script to set application server specific variables needs to be updated on both the local and remote machines after you install Cogility Studio. The script is located under `%OEMHOME%\scripts\subroutines\` on Windows or `$OEMHOME/bin/subroutines/` on LINUX/UNIX.

If you are deploying to WebSphere, locate the script and update the `WAS_HOME` in the script, for example:

On Window update the `set_was_env.bat` file:

```
WAS_HOME=D:/IBM/WebSphere/AppServer
```

On LINUX>UNIX update the `set_was_env.sh` file:

```
WAS_HOME=/opt/IBM/WebSphere/AppServer
```

If you are deploying to Oracle Application Server, locate the script and update the OAS_HOME in the script, for example:

On Windows update the `set_oas_env.bat` file:

```
OAS_HOME=D:/OAS
```

On LINUX/UNIX update the `set_oas_env.sh` file:

```
(OAS_HOME=/opt/OAS)
```

If you are deploying to Weblogic Application Server, locate the script and update the WL_HOME and WL_DOAMIN in the script, for example:

On Windows update the `set_wl_env.bat` file:

```
WL_HOME=D:/bea/weblogic9
WL_DOMAIN=CogilityDomain
```

On LINUX/UNIX update the `set_wl_env.sh` file:

```
WL_HOME=/opt/bea/weblogic9
WL_DOMAIN=CogilityDomain)
```

Project for the deployed model

The deployed model must be associated with the active project on the target machine. Cogility Studio refers to the active project for the location of the source files, libraries and project configuration files for the application running on the target machine. You must run Cogility Project Configuration Editor, create the project and make it the active project. This need be done only once, before you deploy the application. For more information, see "Project Configuration Editor" on page 49 of the guide, *Model Deployment & Execution in Cogility Studio*.

Creating and activating a project

To create and activate a project:

1. On the target machine, from the **Start** menu, select **All Programs** > **Cogility Studio > Project Configuration Editor**.
2. In the **Project** field, right-click and select **Create New Project**.
3. In the input dialog's name field, enter a project name and click **OK**.

   Note: Project names may not include spaces. Cogility Project Configuration Editor automatically replaces spaces in project names with underscores before saving the project folder name. You have the option to cancel the project creation for the name.

   The project name appears in the Project field. The output field reports the subdirectories and files created.
4. In the **Project** field, select the project, right-click and select **Make <project name> the Active Project**.

The project name appears in the Project field in red italics and the active project setup file is updated with the new project.

Hosts file

Before deploying to the target machine, if you specified a hostname in the Target Host field in Modeler, the hostname must be able to be resolved by DNS. If it is recognized by DNS, it is not necessary to add an entry in the machine's hosts file; otherwise, an entry is needed.

The hosts file is located as follows:

- **Windows** - \Windows\System32\Drivers\etc\hosts
- **Linux or Unix** - /etc/hosts

Configuration files

On the Target AppServer Machine

Before any deployment to the target machine, there are two types of files that must be copied from the source machine to the target application server machine. These are the custom configurations file and any project files. Usually, you need to copy these files only once, before deployment. However, if the configuration settings change subsequently, the files for those changes must be copied again.

Custom configurations

Your model's custom configurations reside in the `<DCHOME>\MB\Config\customConfigurations.txt` file. This file must be copied to the `<DCHOME>\MB\Config` location on the target application server machine.

Project configurations

All project configuration files in the `<DCHOME>\Engagements\<project name>\Config\Files` directory on the source machine must be copied to the `<DCHOME>\Engagements\<project name>\Config\Files` directory on the target application server machine. The `<project name>` on the target application server installation is the name you provided for the project you created in "Creating and activating a project" on page 60.

Multiple application server environments

If you are deploying to multiple application servers that access the same database, each deployment's custom configurations should include a setting for that installation's Cogility Watchdog.

Cogility Watchdog is a J2EE application that runs on the application server and manages the time events. To prevent collisions between multiple Watchdogs, each Watchdog must be configured with its own ID. The Watchdog ID is persisted with each running time event record. The following configuration parameter should be added to your `customConfigurations.txt` file under the `<DCHOME>\Config` directory:

```
com.cogility.wachdogId=
```

Each instance of Cogility Studio should contain a different ID. If the parameter is not specified, the default is zero (0).

For each application server instance, there should be a separate Cogility directory installed on that same machine for that particular application server instance. In an environment that employs multiple application server instances (hence, multiple Cogility installation directories), each Cogility installation directory should assign a unique integer for WatchDog and the referenced appserver. These two config params can be defined in `customConfigurations.txt`:

```
com.cogility.watchdogId=<some integer>
```

```
com.cogility.appserverId=<some integer>
```

The default value for `<some integer>` is 0 if those configuration parameters are not specified.

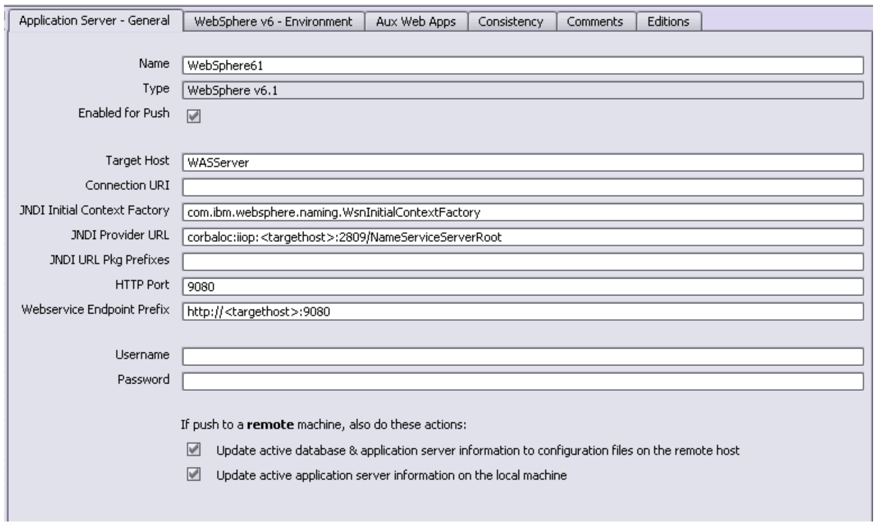
For more information, see *Model Deployment & Execution in Cogility Studio*.

Remote deployment server artifact

Defining the WebSphere 6.1 application server artifact

These instructions describe defining the WebSphere 6.1 application server for a remote deployment. If you are defining a local deployment, follow the instructions in "“Defining a WebSphere 6.1 application server artifact” on page 313 of the guide, *Modeling with Cogility Studio*.

To define the WebSphere 6.1 application server artifact under the General tab:



Name	WebSphere61
Type	WebSphere v6.1
Enabled for Push	✓
Target Host	WASServer
Connection URI	
JNDI Initial Context Factory	com.ibm.websphere.naming.WsnInitialContextFactory
JNDI Provider URL	corbaloc:iiop:<targethost>:2809/NameServiceServerRoot
JNDI URL Pkg Prefixes	
HTTP Port	9080
Webservice Endpoint Prefix	http://<targethost>:9080
Username	
Password	

If push to a **remote** machine, also do these actions:

✓ Update active database & application server information to configuration files on the remote host

✓ Update active application server information on the local machine

1. On the source machine, in Cogility Modeler, select the deployment object.

   You should define a deployment specifically for remote deployment. See “Configuration Parameters” on page 28 of the guide, *Modeling with Cogility Studio*.
2. In the **Target Host** field, enter the host name or IP address of the target machine.

   The value for this field will be substituted for `<target host>` in the subsequent fields. If you specify a host name, that host name must able to be resolved by the DNS on your network. See “Hosts file” on page 61.

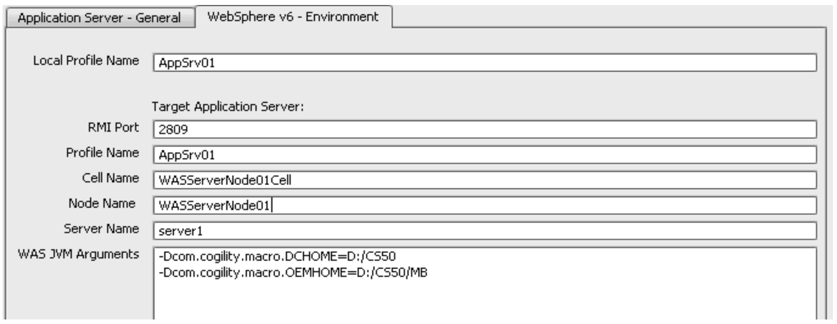
   In the remaining fields on this tab, if your target WebSphere is configured differently than the default configurations, you will need to update the settings appropriately.
3. Check the two checkboxes. These options are only applicable for remote deployment.
   - Checkbox for "Update Active database & application server…"

This checkbox should be checked so that database and application server configuration files, `pear-param.txt` and `Deploy_Active_AppServer.txt`, are automatically copied to the target machine after deployment.

- Checkbox for "Update active application server…"

  Check this box if you want to update the `Deployed_Active_AppServer.txt` on the source machine.

To define the WebSphere 6.1 application server artifact under WebSphere v6-Environment tab:



Local Profile Name	AppSrv01
	Target Application Server:
RMI Port	2809
Profile Name	AppSrv01
Cell Name	WASServerNode01Cell
Node Name	WASServerNode01
Server Name	server1
WAS JVM Arguments	-Dcom.cogility.macro.DCHOME=D:/CS50 -Dcom.cogility.macro.OEMHOME=D:/CS50/MB

1. In the Local Profile Name field, enter the name of the WAS profile used to install WebSphere on the source machine.

   The default profile was used if you installed WebSphere according to the instructions in "IBM WebSphere" on page 28 of the guide, *Installing and Configuring Cogility Studio.*

   The remaining fields on this tab refer to settings on the target machine.
2. In the RMI Port field, enter the RMI port used by WebSphere on the target machine.

   The default is 2809 if you installed WebSphere according to the instructions in "IBM WebSphere" on page 28 of the guide, *Installing and Configuring Cogility Studio*.
3. In the Profile Name field, enter the name of the WAS profile used to install WebSphere on the target machine.
4. In the Cell Name field, enter the value from the target machine's `WAS_CELL` value.
5. In the Node Name field, enter the value from the target machine's `WAS_NODE` value.
6. In the Server Name field, enter the name of the WAS server name.

   The value is server1 if you followed the instructions in "IBM WebSphere" on page 28 of the guide, *Installing and Configuring Cogility Studio*.
7. In the WAS Jvm Arguments field, enter the following:

   a. In the first line, following `DCHOME=`, enter the location where Cogility Studio is installed on the target machine.

      For example, `OEMHOME=D:/CS50`.

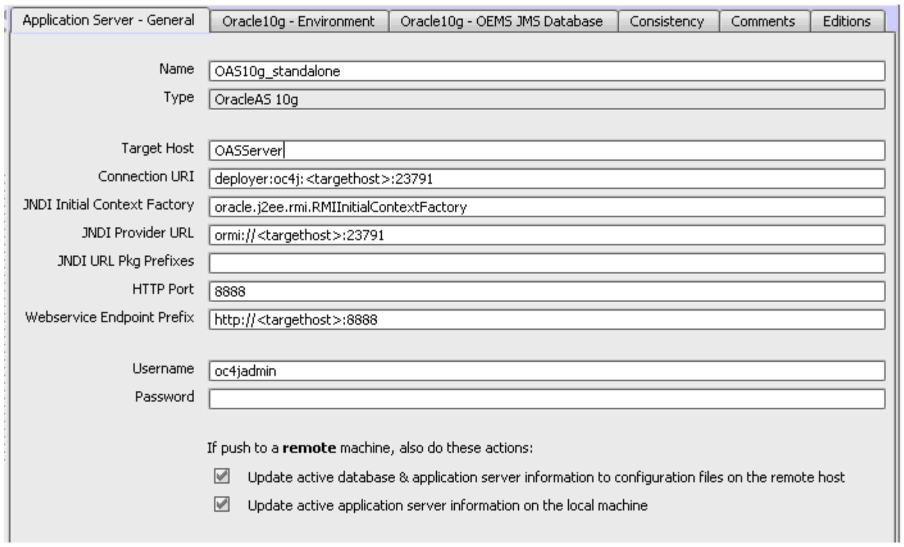
   b. In the second line, following `OEMHOME=`, enter the location where Cogility Studio is installed on the target machine and append /MB to the entry.

      For example, `OEMHOME=D:/CS50/MB`.

Defining the Oracle application server artifact

These instructions describe defining the Oracle application server for a remote deployment.

To define the Oracle10g application server artifact under Application Server-General tab:



Name	OAS10g_standalone
Type	OracleAS 10g
Target Host	OASServer
Connection URI	deployer:oc4j:<targethost>:23791
JNDI Initial Context Factory	oracle.j2ee.rmi.RMIInitialContextFactory
JNDI Provider URL	ormi://<targethost>:23791
JNDI URL Pkg Prefixes	
HTTP Port	8888
Webservice Endpoint Prefix	http://<targethost>:8888
Username	oc4jadmin
Password	

If push to a **remote** machine, also do these actions:

✓ Update active database & application server information to configuration files on the remote host

✓ Update active application server information on the local machine

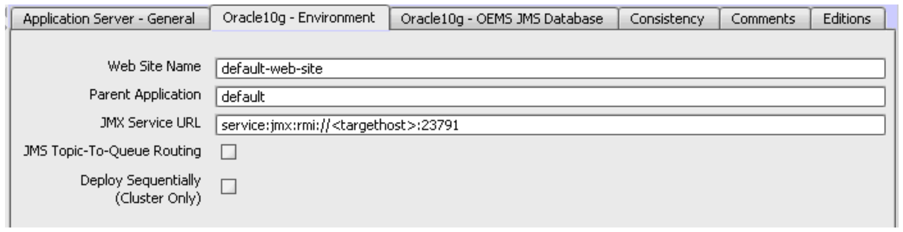
1. In the Target Host field, enter the host name or IP address of the target machine.The value for this field will be substituted for <target host> in the subsequent fields.

   If you specify a host name, that host name must able to be resolved by the DNS on your network. See "Hosts file" on page 61.
2. In the remaining fields on this tab, if your target OAS is configured differently than the default configurations, you will need to update them appropriately
3. Check the two checkboxes. These are only applicable for remote deployment.
   - Checkbox for "Update Active database & application server…"

     This checkbox should be checked so that database and application server configuration files, namely `pear-param.txt` and `Deploy_Active_AppServer.txt`, are automatically copied to the target machine after deployment.
   - Checkbox for "Update active application server…"

     Check this box if you want to update the `Deployed_Active_AppServer.txt` on the source machine.

To define the Oracle application server artifact under Oracle10g-Environment tab:



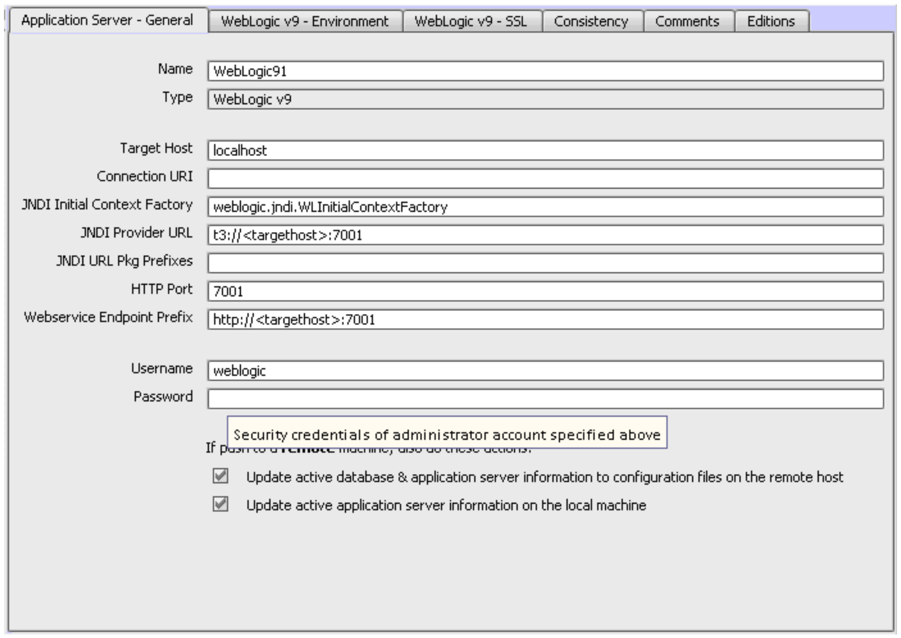
Web Site Name	default-web-site
Parent Application	default
JMX Service URL	service:jmx:rmi://<targethost>:23791
JMS Topic-To-Queue Routing	☐
Deploy Sequentially (Cluster Only)	☐

In the JMX Service URL field, enter the RMI port configured on the target host; the default port is 23791.

Defining the WebLogic application server artifact

These instructions describe defining the WebLogic application server for a remote deployment.

To define the WebLogic application server artifact under Application Server-General tab:



Name	WebLogic91
Type	WebLogic v9
Target Host	localhost
Connection URI	
JNDI Initial Context Factory	weblogic.jndi.WLInitialContextFactory
JNDI Provider URL	t3://<targethost>:7001
JNDI URL Pkg Prefixes	
HTTP Port	7001
Webservice Endpoint Prefix	http://<targethost>:7001
Username	weblogic
Password	

Security credentials of administrator account specified above

If push to a **remote** machine, also do these actions:

- ✓ Update active database & application server information to configuration files on the remote host
- ✓ Update active application server information on the local machine

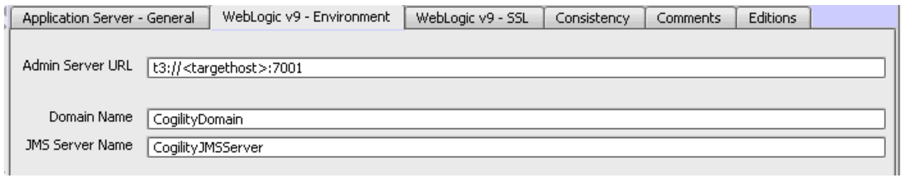
1. In the Target Host field, enter the host name or IP address of the target machine.The value for this field will be substituted for <`target host`> in the subsequent fields.

   If you specify a host name, that host name must able to be resolved by the DNS on your network. See "Hosts file" on page 61.
2. In the remaining fields on this tab, if your target OAS is configured differently than the default configurations, you will need to update them appropriately
3. Check the two checkboxes. These are only applicable for remote deployment.
   - Checkbox for "Update Active database & application server…"

     This checkbox should be checked so that database and application server configuration files, namely `pear-param.txt` and `Deploy_Active_AppServer.txt`, are automatically copied to the target machine after deployment.
   - Checkbox for "Update active application server…"

     Check this box if you want to update the `Deployed_Active_AppServer.txt` on the source machine.

To define the WebLogic application server artifact under Application Server-WebLogic v9 - Environment tab:



Admin Server URL	t3://<targethost>:7001
Domain Name	CogilityDomain
JMS Server Name	CogilityJMSServer

On The Weblogic environment tab, enter the values configured on the target host; the default values are prepopulated.

Database artifact

A database artifact defines a connection to the run-time repository. The run-time repository is a database that holds the deployed system model's schema and the run-time objects that are created during the model application's execution. The connection to the run-time repository consists of a username and password and the connection string that specifies the database's location.

Cogility Studio supports the IBM DB2, MySQL and Oracle 9i and 10g databases. The default deployment model includes one database artifact for the Oracle 9i database that is configured for a default installation. If your deployment requires a different database, you can change this artifact or create a new artifact for the database type.

You should create a database artifact for each discrete database installation. Several deployments may use the same database. But there may be different installations of the same database type, and each must have a distinct database artifact. For example you may have Oracle 9i installed both locally and on a remote machine. When you deploy the model application to the local machine, you use the database artifact for that installation. The artifact's connect string reads, `jdbc:oracle:thin:@localhost:1521:orcl` where remotehost is the name of the machine where the database resides.. If you are deploying the model application remotely, you must use the deployment including the database for that remote machine. Its connect string might read, `jdbc:oracle:thin:@remotehost:1521:orcl`. Because each installation has a different connect string, specifically a different host name, each installation must have its own database artifact.

Defining the database artifact

You can define a database artifact either through a deployment or by editing the artifact in the tree view. Changing the artifact in either location will cause the changes to be propagated to all other deployments where the artifact is used.

Note that if you change the default database artifact for a database other than Oracle 9i, the icon for the artifact will change to match the database type as specified in the connect string.

To define a database artifact:

1. In the tree view, select the database artifact or the deployment containing the database artifact.
2. In the **Database** tab, in the **Name** field, enter a name for the artifact.
3. In the **User ID** field, enter a user ID.
4. In the **Password** field, enter a password.

   You created the User ID and Password values during installation. See *Installing and Configuring Cogility Studio*. The default value for both the user ID and password is **pear**.
5. In the **Connect String** field, enter the database connect string.

   This is the connection string to the database used for the run-time repository. The default connection string to the Oracle database is provided with the default server name (localhost) and the default database name or SID (orcl). See your database server documentation for more information.

Note: The default connect string specifies "localhost" as the location for the database. Regardless of the location of the database, it must be specified by either the IP address or the machine name. If specified by the machine name, that name must be able to be resolved by the DNS on your network, as an entry in the hosts file, for example.

Testing the database connection

Before you can push the model, the model must have a valid database connection string.

To test the database connection:

1. In Cogility Modeler's tree view, select the model container and right-click to display the menus.
2. Under **Actions**, select **Check Database Connection**.

Remote execution

During run time, you can use Cogility tools installed on a remote (or client) machine to work with the model on the target machine. The client machine must have the same Cogility Studio installation as the target machine, and it, too, must have the application server installed; i.e. Websphere, Oracle Application Server, or WebLogic. The Cogility tools on the client machine use JAR files specific to the application server running Cogility Manager. The application server needs not be running on the client machine, but it must be the same application server type as the application server running on the target machine. When you run Cogility tools on the client machine, they must be configured to run with the same application server type that you deployed.

The way to ensure that the Cogility tools run with the proper JAR files is to copy the deployment files, Deploy_Active_App_Server.txt and pear-params.txt from the target machine, where the application server is running, to the client machine.

Also, the hosts file on the client machine must have the proper definitions. See "Hosts file" on page 61.

When running Cogility Insight on a client machine, the URL to which Cogility Insight connects must be that for the target machine's application server. When Cogility Insight runs, it looks for the application server on the local host. To connect with the application server on the target machine, substitute the target machine's host name for `localhost` in the URL:

```
http://localhost:9080/CogilityInsight/
```

9 Cogility Action Pad

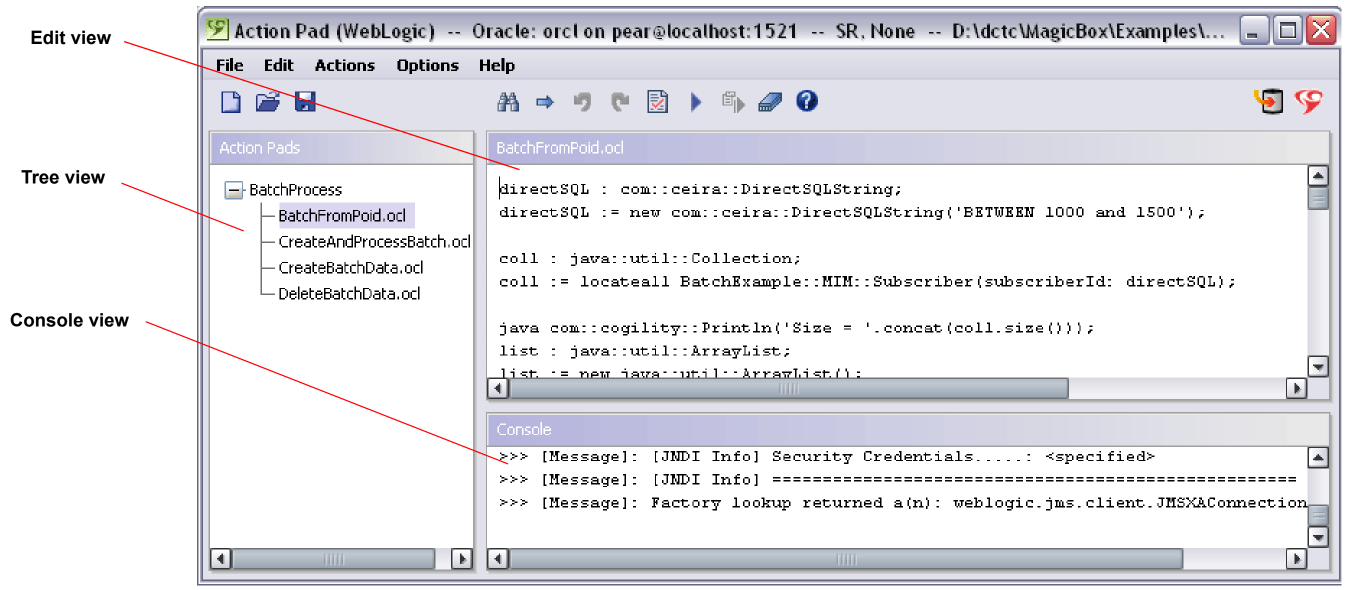
Cogility Action Pad provides a window on the executable model that has been deployed from Cogility Modeler. It allows you to look at the different aspects of the information model and the interface model. However, the Cogility Action Pad is not simply a static environment, it also is an executable environment. It lets you run action semantics against the deployed model, even without using the J2EE application server because Cogility Action Pad runs in its own process thread.

You can also run your action semantics scripts in a .bat file through the Windows console or .sh file in a Unix shell. See “Action semantics batch files” on page 53.

With Cogility Action Pad you can edit, compile and execute one or more action pads (scripts) within a single framework. Cogility Action Pad can check any Cogility action semantics specification for syntactical accuracy (see “Syntax verification” on page 76) and directly execute it against a J2EE application server and associated JDBC database. No matter which J2EE application server or JDBC database you are using, you do not have to change the interface; you simply specify action semantics in terms of your deployed model’s nomenclature exclusive of the execution environment. Cogility Action Pad validates, compiles and executes the action semantics in an action pad against the schema and data in the database where you have deployed your model. You must push the model with that schema into the database to work with it in your action pads.

When you run Cogility Action Pad, it auto-connects to the run-time repository where the model was last pushed and caches the model metadata. From that point on, it is completely aware of the various model artifacts that constitute the executable model. Thus, it can provide help on specifying UML action semantics that need to reference various model elements.

When Cogility Action Pad opens, a list of action pads appears in its **tree view**. The contents of a selected action pad appear in the **edit view**. Messages to standard output as well as error and exception reporting appear in the **console view**.



```
directSQL : com::ceira::DirectSQLString;
directSQL := new com::ceira::DirectSQLString('BETWEEN 1000 and 1500');

coll : java::util::Collection;
coll := locateall BatchExample::MIM::Subscriber(subscriberId: directSQL);

java com::cogility::Println('Size = '.concat(coll.size()));
list : java::util::ArrayList;
list := new java::util::ArrayList();
```

```
>>> [Message]: [JNDI Info] Security Credentials.....: <specified>
>>> [Message]: [JNDI Info] ==================================================
>>> [Message]: Factory lookup returned a(n): weblogic.jms.client.JMSXAConnection
```

Figure 9-6: Cogility Action Pad

The action pads that are opened with Cogility Action Pad are those that were last left open upon exit. Cogility Action Pad maintains the list of the files in a workspace file. See “Help” on page 79.

Running Cogility Action Pad

You must first run Cogility Modeler and push the model before compiling your the action semantics in an action pad. You may also have to start the application server to execute the action pad. You may elect not to start the application server before you run Cogility Action Pad. However, if you create or load action pads that require the application server, you must restart Cogility Action Pad before these will run properly.

To run Cogility Action Pad:

1. Run Cogility Modeler (optional).

   See “Logging into the current context” on page 19 of the guide, *Modeling with Cogility Studio*.

2. Turn over and push your model (optional).

   You do not have to have a model pushed in order to run Cogility Action Pad. It will work without a model, but then you can only use Java semantics. See “Pushing the model into execution” on page 326 of the guide, *Modeling with Cogility Studio*.

3. If the action semantics depend upon the run-time functionality of the J2EE application server, start the application server.

   See “Application server” on page 323 of the guide, *Modeling with Cogility Studio* for instructions on running the application server.

   Action semantics that require the use of the application server are noted in “Actions that require the application server” on page 15 of the guide, *Using Actions in Cogility Studio*. You may elect not to start the application server before you run Cogility Action Pad. However, if you create or load action pads that require the application server, you must restart Cogility Action Pad before these will run properly.

4. From the **Start** menu, select **All Programs > Cogility Studio > Cogility Manager > Action Pad**.

   Cogility Action Pad opens in its own window on your desktop, as shown in Figure 9-6 on page 70.

Action pad files

An action pad is a text-based action semantics script file. For action pads created in Cogility Action Pad the file name is appended with **.ocl** by default (see "Default Extension" on page 82). Cogility Action Pad looks for these files when you open and navigate to a directory of action pads. Cogility Action Pad will open any text file, however. When you create a new action pad, it is given the default name, **TempPad:** followed by a number (starting with zero) for the instance of the new, unamed, temporary action pad. As you save action pads and open existing action pads, they appear in the tree view under the name of the folder where they are located in your system. You have the option of displaying only the folder name or the entire path to the folder. When you exit Cogility Action Pad, any action pads you have opened will be opened again the next time you run Cogility Action Pad. However, any temporary pads you have created, including those that you have renamed, will not be saved, nor will they reappear when you reopen Cogility Action Pad.

Creating a temp pad

To create a new TempPad in Cogility Action Pad:

1. From the **File** menu, select **New**.

   You can also place the cursor in the tree view, right-click and select New. A TempPad appears in the tree view under the <TempPads> branch.

2. Rename the TempPad or save it to a specific location (optional).

   See "Renaming a TempPad" on page 71 and "Saving to a new location" on page 71.

Renaming a TempPad

Renaming a temporary pad does not save it as a file in a folder on your system. It remains a temporary pad, available only during the current session of Cogility Action Pad. When you close Cogility Action Pad, all TempPad, including those you have renamed will be lost. To save a TempPad as an FilePad, see "Saving to a new location" on page 71.

To rename a TempPad:

1. In the tree view, select the TempPad.
2. From the file menu, select **Rename...** .

   You can also right-click and select Rename... .

3. In the dialog, enter a name for the temporary pad and click **OK**.

Saving to a new location

When you save a TempPad as an FilePad for the first time, or save an action pad with a new name or location, the name of the folder holding the file appears in the tree view as the container of the action pad. You can specify whether the label for this location includes the entire path or just the folder name.

To save a TempPad or a FilePad:

1. In the tree view, select the TempPad or FilePad.
2. From the **File** menu, select **Save As...** .
   You may also right-click and select Save As... .
3. In the dialog, navigate to the location where you want to save the pad.
4. In the **File Name** field, enter a name and click **Save**.
   The named action pad appears in the tree view under the branch named for the folder where its file is saved.
5. To show the full path of the folder location, from the **File** menu, select **Show Full Path**.
   You can also right-click and select Show Full Path.

Saving existing FilePads

As you edit your action pads, you'll want to periodically save the changes.

To save existing FilePads:

1. In the tree view, select the FilePad(s).
   Hold down the Shift or Ctrl key while selecting to select multiple files.
2. From the **File** menu, select **Save**.
   You can also right-click and select Save.
3. To save all opened FilePads, regardless of selection, from the **File** menu, select **Save All**.
   You can also right-click and select Save All.

Opening FilePads

To open a FilePad:

1. From the **File** menu, select **Open...** .
   You can also right-click in the tree view and select Open... .
2. Navigate to where the file is located.
   By default, Cogility Action Pad looks for files with an .ocl extension. However you can open any text file.
3. To display other file types, in the **Files of Type** dialog, select **All Files**.
4. Select the file(s) and click **Open...** .
   Hold down the Shift or Ctrl key while selecting to select multiple files. The FilePads appear in the tree view under the branch named for their folders.

1. To show the full path of the folder location, from the **File** menu, select **Show Full Path**.
   You can also right-click and select Show Full Path.

Printing an action pad

Cogility Action Pad includes all of the standard printing facilities. You can print both FilePads and TempPads.

To print a FilePad:

1. In the tree view, select the FilePad or TempPad.
2. From the **File** menu, select **Print...** .

You can also right-click and select Print... .

3. In the **Print** dialog, select the printer and any other settings, and click **Print**.

Closing an action pad

If you close a TempPad, it is removed from the tree and its contents are discarded. If you close a FilePad, it is removed from the tree but its contents remain in the file.

To close an action pad:

1. In the tree view, select the action pad(s).

   Hold down the Ctrl key while selecting to select multiple files.
2. From the **File** menu, select **Close**.

   You can also right-click and select Close.
3. In the dialog that appears, click **Yes** to confirm the close or **No** to abort it.

Action pad editing

Action semantics in Cogility Studio are based on the Object Constraint Language and Action Semantics developed by the Object Management Group (OMG). See "Action semantics" on page 9 of the guide, *Using Actions in Cogility Studio*. Some action semanticsrequire the use of the application server, and these are noted in "Actions that require the application server" on page 15 of the guide, *Using Actions in Cogility Studio*. Otherwise, Cogility Action Pad provides a process independent of the application server for executing against a deployed run-time model.

Cogility Action Pad is aware of all Java classes. Thus, it is possible to declare a variable to be of a particular type, have access to all of its attributes and methods and immediately invoke them, without having to put them in the context of an executable Java class. Cogility Action Pad allows the immediate execution of Java methods on Java class instances without requiring a .java file or a .class file and without the Java compiler.

Cogility Action Pad can directly execute any action semantics specification that is entered into it. It can easily handle complex, ad-hoc logic that combines local and remote database access, web service invocations as well as publication of JMS messages, in background as well as interactive mode. It uniquely combines the power of Java class libraries, model-based logic and database access, third-party database access, JMS, Web services and logical control structures in a highly interactive environment that does not require structured Java programming skills.

Like Cogility Modeler, Cogility Action Pad includes the meta-data driven model assist system that intelligently displays context-sensitive suggestions for those elements of your model that you might want to specify next. See "Model assist" on page 53 of the guide, *Modeling with Cogility Studio*. Cogility Action Pad also includes syntax assist that helps with writing a new line of action semantics by indicating the entire set of action semantics keywords available, or a subset if you have provided some initial semantics. See "Syntax assist" on page 66 of the guide, *Modeling with Cogility Studio*.

Any action semantics specified in the action pad can be saved as independent, free-floating logic fragments that can be executed again later if required. This allows you to create detailed, model-based regression test scenarios to aid in automated testing. You can also do this for quick reports, debugging or for developing any other aid to interacting with the execution environment. See "Common usage scenarios" on page 82 for more information.

Basic editing

Common to both Cogility Action Pad and Cogility Modeler are several editing features. These are available when you right-click in any action semantics field or select them from the Actions menu in Cogility Action Pad or the Selection > Actions menu in Cogility Modeler. You'll find these described in the following sections of the guide, *Modeling with Cogility Studio*:

- "Common editing functions" on page 51 - these include cut, copy, paste, undo, redo, print, go to and others. Several of these are also available for text in the console view.
- "Tab and indent" on page 53 - these functions move entire lines or sections of action semantics so you can define blocks and sections.
- "Model assist" on page 53 and "Keystrokes and mouse click responses" on page 64 - provide automated statement completion and syntax suggestion functions.
- "Find / Replace" on page 67 - these functions let you find strings and replace them in an action semantics field. You can also use the find feature for text in the console view.
- "History comments" on page 70 - provide in-line change tracking for all action semantics listings.

Syntax errors

Action Pad provides the ability to go directly to a syntax error, so that you can easily fix it. When syntax errors are detected during the verification of action semantics, the action semantic field and console scroll to the error location automatically.

To find a syntax error manually:

1. In the text view, right click and select **Find/GoTo... Syntax Error**.

   The error and its corresponding line number appear in the console view.

Line wrapping

By default, the text in the edit and console views does not wrap. Rather, the window buffer size expands to accomodate an almost unlimited line length. To bring into view the hidden end of a very long line, you only have to move the scroll bar at the bottom of the frame. When you enable line wrapping, any text previously hidden due to the chosen window size is moved into view; the hidden lines wrap and the scroll bar is disabled. The text itself is not changed; no new return characters are added. If you cut and paste the line into another editor, it will appear unwrapped.

Enabling and disabling line wrapping

To enable or disable line wrapping:

1. In the tree view, select the action pad, and from the **Actions** menu, select **Enable Line Wrapping**.

   With the cursor in the edit or console view, you can also right-click and select Enable Line Wrapping.
2. In the tree view, select the action pad, and from the **Actions** menu, select **Disable Line Wrapping**.

   With the cursor in the edit or console view, you can also right-click and select Disable Line Wrapping.

Revert

Cogility Action Pad offers two levels of edition reversion: you can revert to the last action pad saved or you can revert to the last action pad verified. See "Saving existing FilePads" on page 72 and "Syntax verification" on page 76. With either, the entire action pad is reverted.

You might use revert to last saved when you load a file and modify it slightly to try something out, but do not want to save it. You might use revert to last verified after making several changes. For example, you might have inadvertently deleted a section of action semantics that had been verified earlier. You would use revert to last verified to bring back the last set of unsaved changes that compiled successfully. Revert to last verified is for reverting to an unsaved working stage after multiple modifications.

Reverting to Last Opened/Saved

To revert the FilePad to the last version saved or opened:

1. In the tree view, select the FilePad you want to revert.
2. From the **Actions** menu, select **Revert to Last Opened/Saved...** .

   You can also place the cursor in the edit view, right-click and select Revert to Last Opened/ Saved... .

Reverting to Last Verified

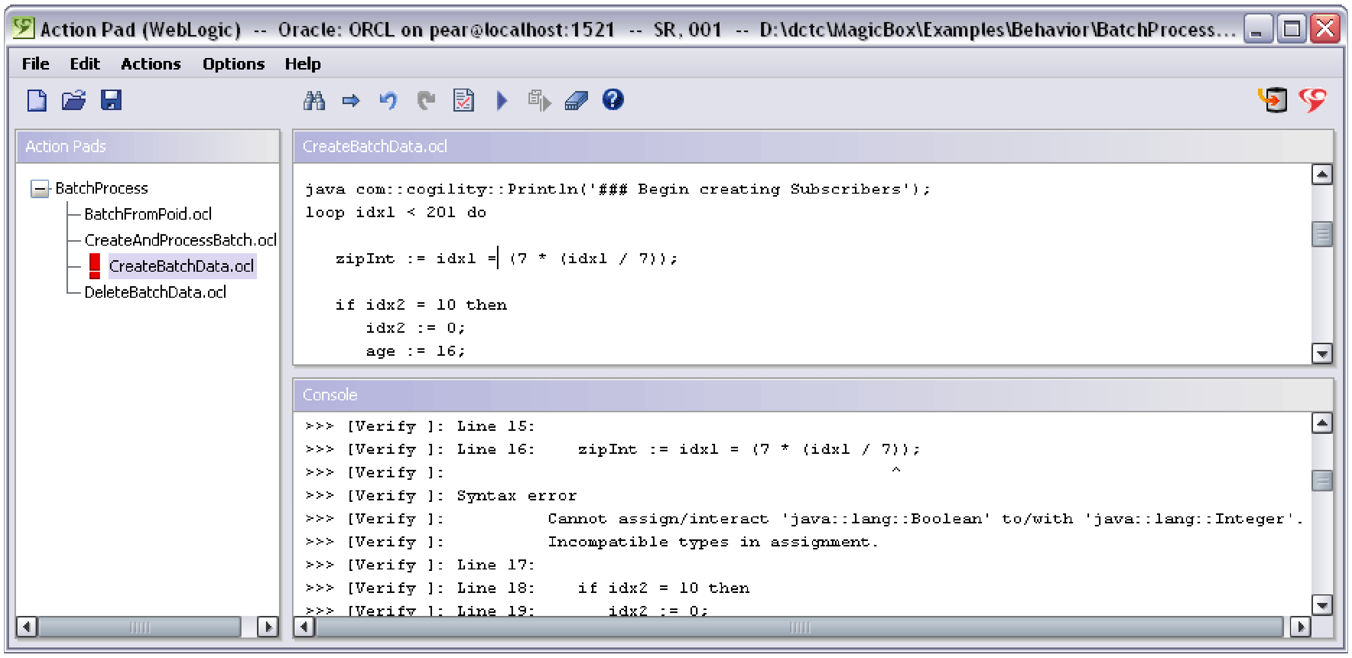
To revert the action pad to the last version verified:

1. In the tree view, select the action pad you want to revert.
2. From the **Actions** menu, select **Revert to Last Verified...** .

   You can also place the cursor in the edit view, right-click and select Revert to Last Verified... .

Syntax verification

Cogility Action Pad is fully knowledgable of action semantics syntax. It can detect incorrect syntax and suggest alternatives. It locates precisely where the syntax is in error and provides a list of known variables appropriate to the context.


Action Pad (WebLogic) -- Oracle: ORCL on pear@localhost:1521 -- SR, 001 -- D:\dctc\MagicBox\Examples\Behavior\BatchProcess...
File
Edit
Actions
Options
Help
Action Pads
BatchProcess
BatchFromPoid.ocl
CreateAndProcessBatch.ocl
CreateBatchData.ocl
DeleteBatchData.ocl
CreateBatchData.ocl
Console

```
java com::cogility::Println('### Begin creating Subscribers');
loop idx1 < 201 do

   zipInt := idx1 =| (7 * (idx1 / 7));

   if idx2 = 10 then
      idx2 := 0;
      age := 16;
```

```
>>> [Verify ]: Line 15:
>>> [Verify ]: Line 16:    zipInt := idx1 = (7 * (idx1 / 7));
>>> [Verify ]:                                     ^
>>> [Verify ]: Syntax error
>>> [Verify ]:          Cannot assign/interact 'java::lang::Boolean' to/with 'java::lang::Integer'.
>>> [Verify ]:          Incompatible types in assignment.
>>> [Verify ]: Line 17:
>>> [Verify ]: Line 18:    if idx2 = 10 then
>>> [Verify ]: Line 19:       idx2 := 0;
```

Figure 9-7: Syntax verification

When you verify syntax, any action semantics in error are reported in the console view and the action pad is shown with a red exclamation flag in the tree view.

Verifying syntax

To verify action semantics syntax:

1. In the tree view, select the action pad you want to verify.
2. Click the **Verify Syntax** button .

   You can also place the cursor in the edit view, right-click and select Verify Syntax, or from the Actions menu, select Verify Syntax.

Console view editing functions

For the text in the console view, you can use several of the basic editing functions. See "Basic editing" on page 74. You can also clear the console.

Clearing the console

To clear the console:

1. Click the **Clear Console** button

   With the cursor in either the edit or console view, you can also right-click and select Clear Console, or from the Actions menu, select Clear Console.

Action pad execution

When you run the action semantics, a database transaction begins and remains throughout the run. In action semantics, you can commit the current changes in the transaction with oclCommit or roll back the current changes in the transaction with oclRollbac. When the actions finish running, the current changes in the transaction will be committed if the "Commit After running" option is enabled. The current changes will be rolled back, otherwise.

You can run an action pad in its entirety, or you can select specific lines and run them to view printed output or inspect them to view the result of selected operations. When you run an action pad or selected action semantics, the listing is verified, serialized, executed and the results are committed to the database. Several execution options (see "Options" on page 78) are available to control this process.

Model configuration

During model execution, you may want to change and reload the configuration parameters for the deployed model. One way to do this is by sending a message to the application server (see "Reload configuration message" on page 36). However, Cogility Action Pad has two menu selections that let you reload configuration information:

- **Reload Configuration Parameters in App Server**

  This has the same effect as sending the reload configuration message to the application server, as described in "Reload configuration message" on page 36. That message causes the configuration values to be read from the external source files where they were created. See "Reloading configuration parameters in the application server" on page 77, below.

- **Reload Configuration Parameters**

  This reloads the configuration parameters that are accessed by action semantics that are run in the action pad itself.(see "Cogility Functions" on page 93). See "Reloading configuration parameters" on page 77, below.

Reloading configuration parameters in the application server

To reload the configuration parameters in. the application server:

1. From the **File** menu, select **Reload Configuration Parameters in App Server**.

Reloading configuration parameters

To reload the configuration parameters from the application server:

1. From the **File** menu, select **Reload Configuration Parameters**.

Model metadata

The model metadata (or model schema) is stored in the run-time repository (database) during deployment. You may want to update the model in the current context and reload the model metadata for Cogility Action Pad. If you've changed any of the structures in your model (e.g., added an attribute or a class), you can use Reload Model Metadata to make those changes visible without having to stop and restart Cogility Action Pad. You can also reload the model metadata in the application server. Two menu selections provide for these options:

- **Reload Model Metadata**

  This causes the model metadata to be reloaded into Cogility Action Pad from the run-time repository. "Reloading model metadata" on page 78, below.

- **Reload Model Metadata in App Server**

  This causes the application server to flush its cache of the model metadata and reload it from the run-time repository as described in "Refresh PEAR cache" on page 35. See "Reloading model metadata in the application server" on page 78, below.

Reloading model metadata

To reload model metadata:

1. From the **File** menu, select **Reload Model Metadata**.

   The updated model metadata is loaded into Cogility Action Pad.

Reloading model metadata in the application server

To reload model metadata in the application server:

1. From the **File** menu, select **Reload Model Metadata in App Server**.

   The model metadata is reloaded on the application server.

Options

Execution options pertain to each action pad; you must select the action pad in the tree view in order to set the options for it. The options are available from the Options menu; they are enabled if checked. The options are as follows:

- **Log Console Output to File**

  When you check this option, a navigation dialog displays that lets you specify the directory where you want the log file for the currently selected action pad to be saved. After you navigate to the location and click Accept, all output will be logged to a file in that location. The file is named with the name of the action pad and appended with .log. This option is disabled by default.

- **Serialize After Verify**

  By default, your action semantics will be serialized each time you verify them (see "Syntax verification" on page 76). You can disable the serialization by unchecking this option. Action semantics must be serialized in order to push successfully onto an application server. The application server must then unserialize the action semantics in order to run them. Cogility Action Pad, however, can run unserialized action semantics without having them first serialized. You can use this option to test whether the Java unserialization and execution differs from unserialized execution in Cogility Action Pad.

- **Commit After Running**

  By default, any time you run an action pad or selected action semantics, the execution opens and commits a transaction on the database. You might want to disable automatic commitment when you run selected action semantics especially, to prevent corruption of the run-time data.

  When this option is enabled, this is shown in the upper right corner above the edit view with the commitment enabled icon. When the option is disabled, the commitment disabled icon appears.

  Also, this option must be enabled in order for the oclCommit and oclRollback actions to work. See "oclCommit" on page 41 and "oclRollback" on page 44 of the guide, *Using Actions in Cogility Studio*.

- **Show Log Messages**

  By default, log messages are displayed in the console. Log messages are prefixed with `>>> [Message]:`.

- **Show Exception Stack Traces**

  By default, exception stack traces are displayed in the console. If you disable this option, the exception stack trace will still appear in the platform console, and a dialog notifying you of the exception will appear in Cogility Action Pad, but the Cogility Action Pad console will not display the exception stack.
- **Show Prefixes in Console**

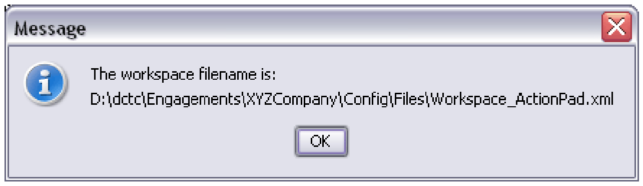
  Prefixes identify the type of output displayed in the console: `>>> [Message]:` for log messages, `>>> [Println]:` for print statements and so forth. If you disable these, only the information after the colon appears in the console.
- **Enable 'pause' Dialogs**

  By default, the 'pause' action outputs variable values to the platform console. In the Cogility Action Pad, this option can alter the default behavior. If disabled, the 'pause' action outputs variable values. If enabled, the 'pause' action outputs variable values, it also stops the execution and opens a dialog that allows the user to control what happens next. See "pause" on page 45 of the guide, *Using Actions in Cogility Studio*.

Help

The Help menu has, in addition to information about the current version of Cogility Action Pad, license expiration date and access to the documentation, information about the current model in deployment and the active workspace filename. The current model in deployment is the one last pushed. Often you'll want to be sure you have the right model deployed before running action pads against it.

The workspace file maintains the references to the action pads that open each time you run Cogility Action Pad and the options for those action pads. The filename reported from the Help menu is that for the active workspace file. The active workspace file pertains to the active project established in Cogility Project Configuration Editor. The files opened in Cogility Action Pad are those for the active project.



The workspace filename is:
D:\dctc\Engagements\XYZCompany\Config\Files\Workspace_ActionPad.xml

Figure 9-8: Active workspace filename in Cogility Action Pad

Viewing the current model

To view the currently deployed model:

1. From the **Help** menu, select **Show Pushed Model Information...** .

   Note that this selection is disabled if no model is pushed into execution.
2. In the message dialog, click **OK** to close the dialog.

Viewing the workspace filename

To view the active workspace filename:

1. From the **Help** menu, select **Show Workspace filename...** .

   A message dialog that shown in Figure 9-8 on page 79 appears.
2. In the message dialog, click **OK** to close the dialog.

Running action semantics in an action pad

To run action semantics in an action pad:

1. In the tree view, select the action pad.
2. Click the **Run** button .

   In the edit view, you can also right-click and select Run, or from the Actions menu, select Run.

Running selected actions

To run selected action semantics:

1. In the tree view, select the action pad.
2. In the edit view, select the action semantics line you wish to run.
3. Click the **Run Selected Actions** button .

   In the edit view, you can also right-click and select Run Selected Actions, or from the Actions menu, select Run Selected Actions.

Running selected actions to inspect

To run selected action semantics to inspect the results:

1. In the tree view, select the action pad.
2. In the edit view, select the action semantics line you wish to run.
3. In the edit view, right-click and select **Run Selected Actions - Inspect**.

   You can also choose from the Actions menu Run Selected Actions - Inspect. A dialog with the result displays.
4. In the dialog, click **Close**.

Running select actions to print

To run selected action semantics to print:

1. In the tree view, select the action pad.
2. In the edit view, select the action semantics line you wish to run.
3. In the edit view, right-click and select **Run Selected Actions - Print**.

   You can also choose from the Actions menu Run Selected Actions - Print. The execution results are printed to the console.

History comments

As in Cogility Modeler, you can set the user name and the change request number (or tracking reference) for history comments in Cogility Action Pad files. See “History comments” on page 70 of the guide, *Modeling with Cogility Studio* for more information.

Setting the history comment user name

To set the history comment user name in an action semantics script file:

1. From the **Actions** menu, select **Set History Comment Username**.

Setting the history comment tracking reference

To set the history comment user name in an action semantics script file:

1. From the **Actions** menu, select **Set Change Request Number**.

Note: “Set Change Request Number” is the default menu label. The “Change Request Number” portion can be changed by specifying a value for the configuration parameter named, **com.cogility.OCL.historyComments.problemIdTitle**.

Message Viewer

If you want to send a JMS message to the application server while you are working in Cogility Action Pad, you can open Cogility Message Traffic Viewer. See “Message Traffic Viewer” on page 119.

Running Cogility Message Traffic Viewer

To run Cogility Message Traffic Viewer:

1. From the **File** menu, select **Open Message Viewer**.

Configuration

You may define some of the configuration settings for Cogility Action Pad. These settings are described in the following sections. To change the settings, copy the default line from `<DCHOME>\MB\Config\defaultConfigurations.txt`, paste it into your customConfigurations.txt file and change the default setting. See “Customizing installation configuration” on page 13 of the guide, *Model Deployment & Execution in Cogility Studio*.

Window Size

These parameters control the initial size of the display window. The value for each may be an integer or a float. An integer value is treated as a pixel size in the range of 1 to the screen size. A float value is treated as a scaled value relative to the screen size in the range of 0.01 to 1.0. The example below describes a window with a width of 950 pixels and a height that is 75% of the screen’s height.

```
com.cogility.ActionPad.windowWidth=950
com.cogility.ActionPad.windowHeight=0.75
```

Window Location

These parameters control the initial location of the Cogility OCL Development Tool window. The values may be integers in the range of 0 to the screen’s size minus 50.

```
com.cogility.ActionPad.initialWindowLocation.X=50
com.cogility.ActionPad.initialWindowLocation.Y=50
```

Default Directory

This parameter controls the directory where the Load File(s) and Save To File dialogs open the first time. Subsequent dialogs default to the directory from the previous dialog’s usage.

In the directory specification, you may use forward slashes, regardless of the platform. If you choose to use backslashes on your PC, you must double them (as in D:\\temp).

```
com.cogility.ActionPad.defaultDirectory=D:/Temp
```

Default Extension

This parameter defines a default file extension for the SaveAs... operation. This extension is appended automatically if and only if this parameter is defined and no extension is provided in the SaveAs... dialog. This extension is not appended if the OCL workspace being saved was loaded from a file and the original filename did not have an extension.

```
com.cogility.ActionPad.defaultExtension=.ocl
```

Common usage scenarios

Cogility Action Pad has many uses, and these are described in several places in the Cogility Studio documentation. The following list summarizes some of these uses.

- Accessing configuration parameters with action semantics - using the DcConfigurationFacility, you can set and retrieve configuration parameters in your action semantics scripts. See "Accessing configuration parameters in action semantics" on page 23.
- Using the pause and continue keywords to control and debug action semantics execution - these cause a window to display in Cogility Action Pad that reports the variables currently in scope and their values.
- Publishing messages to initiate state machine execution and transition.
- Executing and manipulating web services.
- Executing SQL queries with action semantics - see "Queries" on page 87 of the guide, *Using Actions in Cogility Studio*.

10 Deployment Tool

The Deployment Tool provides a way to load a Model into a repository and to deploy that Model to one or more AppServers. It processes a script and executes the directives within that script without user interaction (other than selecting the script).

Run_DeploymentTool.bat

The DeploymentTool is started by a batch file. It works very much like Run_CogilityModeler.bat in that it may prompt you to select the target AppServer type. Additionally, optional command line arguments are supported. If a " script <filename>" command line argument is supplied, the tool will use that file as the script to be executed, otherwise, you are prompted to select a file from a file chooser dialog. After that, there is no additional user interaction.

Deployment Tool Scripting Syntax

A Deployment Tool script must have the following statements in the following order:

- Required: "DefineLogFile" statement
- Required: "SetRepository" statement
- Optional: "LoadNewRepository" statement
- Optional: "Import" statements
- Optional: "TurnOverDeploymentModel" statement
- Optional: "PlaceModelUnderCM" statement
- Optional: "Push" statements
- Optional: "ExportMapOfTableNames" statement
- Optional: "ExitModeler" statement

Note: The keywords in the statements are NOT case sensitive and a <Filespec> may include environment variables (for example, %EnvVar%\Temp, $EnvVar/Temp, or, ${EnvVar}/Temp).

The Deployment Tool script utility validates the script. If valid, it executes the script and logs its progress to a file, as follows:

1. Sets the repository against which to work.
2. Loads a new, empty repository (optional).
3. Opens a Modeler.

4. Imports zero or more Models and/or DeploymentModels (optional).
5. Places Models under CM, if there are any (optional).
6. Turns Over the DeploymentModel, if there is one (optional).
7. Pushes the model to zero or more targets (optional).
8. Exports the Map of Table Names corresponding to the database (optional).
9. Exits the Modeler before ending the script (otherwise the Modeler remains open)

If an error occurs anytime before the first 'push' is completed, the DeploymentTool terminates the execution of the script. After the first 'push' has completed, the DeploymentTool will log errors, but, it will continue and try to complete as much of the script as possible.

DefineLogFile statement

Defines where log messages from the script are to be output.

Syntax: DefineLogFile <Filespec>

If <Filespec> is a filename, logging is appended to that file.

If <Filespec> is a directory, a unique filename is generated for that directory.

SetRepository statement

Defines the File Based Repository to be used by the script.

Syntax: SetRepository <RepoId>

There are two modes for this statement. "Configuration Defined Repositories" mode is used when you have configuration parameters that define sets of com.ohana.object.persistence.db.* values for each File Based Repository (FBR), or, "No Configured Repositories" mode where no such parameters are defined.

For "Configuration Defined Repositories" mode:

- If <RepoId> is a number, it indicates a specific set of the com.ohana.object.persistence.db.* config params.
- If <RepoId> is a directory path, it indicates one of the com.ohana.object.persistence.db.databasename.* config params (in reverse). The directory must be one of the defined databasenames.

For "No Configured Repositories" mode:

- If <RepoId> is a number, it must be 1, and it indicates the default FBR that is created during installation.
- If <RepoId> is a directory path, it must be like: "%OEMHOME%\FBR\Modeler".

LoadNewRepository statement

Specifies that the Load Persistent Repository utility is to be run before starting the Modeler. This option does not check to see whether it is overwriting an existing database. If there is an existing database, it will be overwritten.

Syntax: LoadNewRepository

Import statement

Imports a model. This statement has options for file-set imports or single-file imports. It has options for importing into the context (visible in the Modeler's navigation tree) or importing into the repository (for use by the CM utilities).

Syntax: Import <ImportFormat> { ImportTarget } <Filespec>

<ImportFormat> can be either of the following keywords: FileSet or SingleFile.

{ ImportTarget } for SingleFile only, can be either of the following keywords: IntoContext or IntoRepository.

<Filespec> specifies the file to be imported (.cogs for FileSet, or, .cog for SingleFile).

TurnOverDeploymentModel statement

Turns Over DeploymentModels that have been imported.

Syntax: TurnOverDeploymentModel [<VersionName>]

<VersionName> if specified, uses this version name when turning over the DeploymentModel .

If not specified, "-TO" is appended to the current version name.

PlaceModelUnderCM statement

Places models under CM if they are not already under CM (Hammers them).

Syntax: PlaceModelUnderCM [<VersionName>]

<VersionName> if specified, uses this version name when placing the Model under CM.

If not specified, "-CM" is appended to the current version name.

Push statement

Pushes the Model that is visible in the context. This statement has options to do a full push, an appserver only push, or, a database only push.

Syntax: Push <PushType> <DeploymentKey> [<PushOptions>]

<PushType> can be any of the following keywords: Full, AppServerOnly or DatabaseOnly.

<DeploymentKey> is the XML key of the Deployment that is to be used by the push.

<PushOptions> are options that correspond to the CheckBoxes on the "push" dialog:

- ForcePushOfAppServerArtifacts - valid for 'Full' push type only
- ForcePushOfRecompiledActionSemantics - valid for 'Full' and 'DatabaseOnly' push types
- UseMapOfTableNames <FileSpec> - valid for 'Full' and 'DatabaseOnly' push types
    - <FileSpec> a file that was created by "Export Map Of Tables Names" (either the Modeler menu option, or, the DeploymentTool statement).

ExportMapOfTableNames statement

Exports the "Map of Table Names" to the specified file.

Syntax: ExportMapOfTableNames <FileSpec>

<FileSpec> is a filename in which the "Map of Table Names" is written.

ExitModeler statement

Specifies that the Modeler is to exit at the end of the script, otherwise, the Modeler remains open.

Syntax: ExitModeler

Comment lines

Comment lines can be included in scripts. Any line that begins with a pound sign (#) is ignored during processing.

Syntax: # <any text>

Spaces in a <FileSpec>

If a <FileSpec> contains a space, the entire <FileSpec> should be enclosed in double quotes.

Blank lines

Blank lines can be included in scripts. Blank lines are ignored during processing.

Whitespace

Extra spaces and/or tabs can be included in statements, unless they are between double quotes.

Examples of scripts

The following example script loads a new repository, imports the Transform demo, and, leaves the Modeler open:

```
DefineLogFile %OEMHOME%\log\example.log

SetRepository %OEMHOME%\FBR\Modeler

LoadNewRepository

Import SingleFile IntoContext %OEMHOME%\Demos\Xform\XformDemo.cog

PlaceModelUnderCM
```

This following example script imports a DeploymentModel, a Model and pushes the Model to multiple AppServers:

```
DefineLogFile D:\MyLogDir

SetRepository 1

LoadNewRepository
```

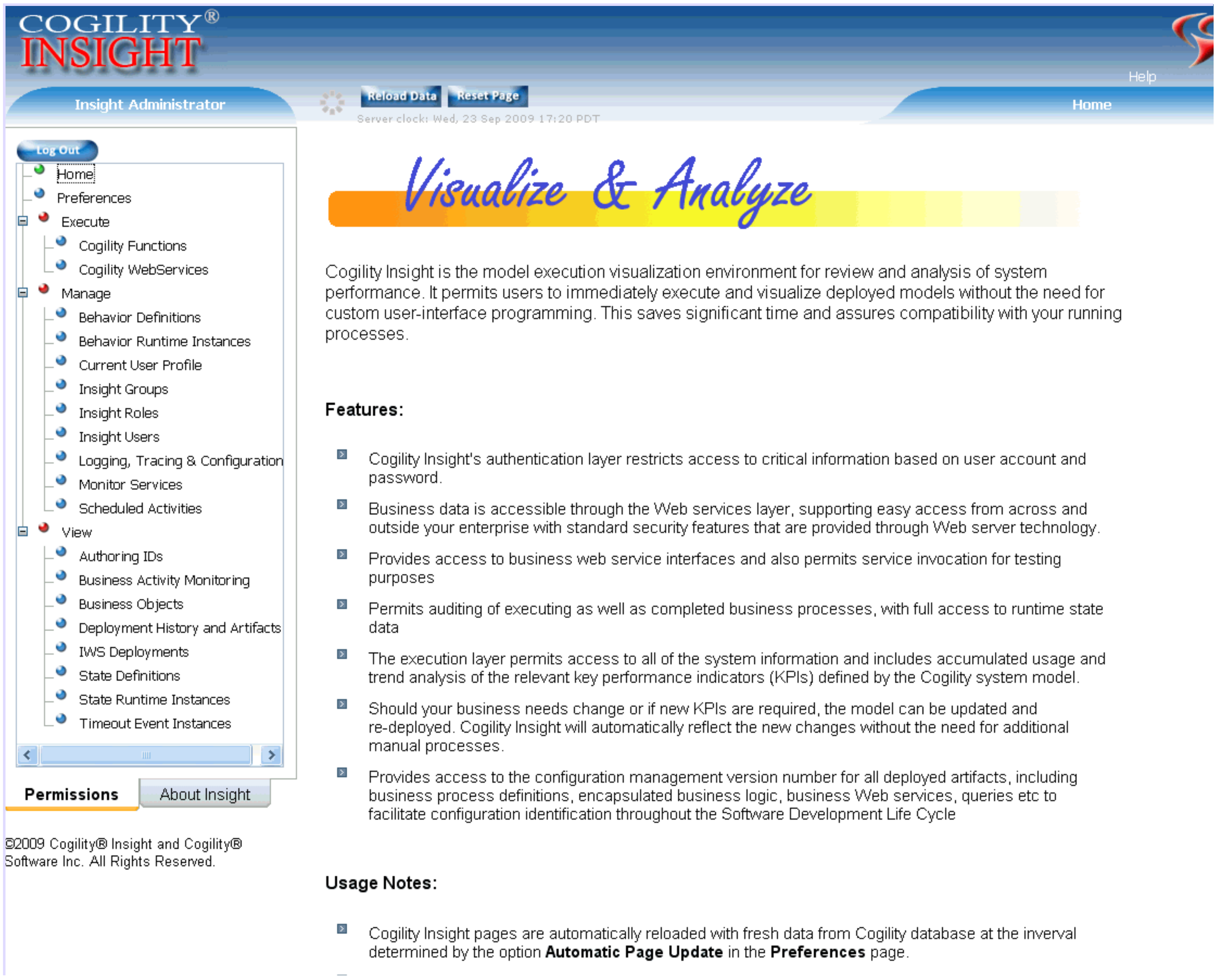
```
   Import SingleFile IntoContext
D:\dctc\Engagements\MyProject\Models\MyDeployments.cog

   Import FileSet D:\dctc\Engagements\MyProject\Models\MyModel.cogs

   PlaceModelUnderCM  FromDepTool


   # Deploy MyModel to 3 AppServers

   Push Full                    DefaultDeploymentModel.localOAS

   Push AppServerOnly DefaultDeploymentModel.remoteOAS1

   Push AppServerOnly DefaultDeploymentModel.remoteOAS2


   ExitModeler
```

In the Login window, there are usually two selections in the Configuration Maps list in the CM Context area (DefaultDeploymentModel and the name of your engagement Model). To avoid confusion, you might want to create a repository where you should select the Configuration Map with the name of your engagement Model (instead of selecting DefaultDeploymentModel). The following pair of scripts can be used to import a custom DeploymentModel and your engagement Model so that the Configuration Map that should be selected is the one with the name of your engagement Model:

```
   DefineLogFile D:\MyLogDir

   SetRepository 1

   Import SingleFile IntoContext D:/Temp/CogFiles/MyDeploymentModel.cog

   TurnOverDeploymentModel  FromDepTool

   ExitModeler


   DefineLogFile D:\MyLogDir

   SetRepository 1

   Import SingleFile IntoContext D:/Temp/CogFiles/MyModel.cog

   PlaceModelUnderCM FromDepTool

   ExitModeler
```

11 Cogility Insight

Cogility Insight is a web application that runs in a web browser. It lets you view your model's business objects on the runtime repository, view the status of state machines, view web service deployments and run web services, all during your composite application's execution.



©2009 Cogility® Insight and Cogility® Software Inc. All Rights Reserved.

Cogility Insight is the model execution visualization environment for review and analysis of system performance. It permits users to immediately execute and visualize deployed models without the need for custom user-interface programming. This saves significant time and assures compatibility with your running processes.

Features:

- Cogility Insight's authentication layer restricts access to critical information based on user account and password.
- Business data is accessible through the Web services layer, supporting easy access from across and outside your enterprise with standard security features that are provided through Web server technology.
- Provides access to business web service interfaces and also permits service invocation for testing purposes
- Permits auditing of executing as well as completed business processes, with full access to runtime state data
- The execution layer permits access to all of the system information and includes accumulated usage and trend analysis of the relevant key performance indicators (KPIs) defined by the Cogility system model.
- Should your business needs change or if new KPIs are required, the model can be updated and re-deployed. Cogility Insight will automatically reflect the new changes without the need for additional manual processes.
- Provides access to the configuration management version number for all deployed artifacts, including business process definitions, encapsulated business logic, business Web services, queries etc to facilitate configuration identification throughout the Software Development Life Cycle

Usage Notes:

- Cogility Insight pages are automatically reloaded with fresh data from Cogility database at the inverval determined by the option **Automatic Page Update** in the **Preferences** page.

Figure 11-1: Cogility Insight home page.

Cogility Insight is a web application that runs on your J2EE application server. As such, Cogility Studio must be deployed to that application server and the server must be running in order to run Cogility Insight.

Note: For installations running IBM WebSphere, you must make some minor filename changes before Cogility Insight will run properly. See "Insight and WebSphere conflicts" on page 34 of the guide, *Installing and Configuring Cogility Studio*.

Running Cogility Insight

To use Cogility Insight you must start the application server and log in.

To log in to Cogility Insight:

1. If the application server is not already running, start the application server.

   See "Starting the application server" on page 323 of the guide, *Modeling with Cogility Studio*.
2. From the **Start** menu, select **All Programs > Cogility Studio > Cogility Insight Web Access.**

   The login screen displays.
3. In the **UserName** and **Password** fields, enter the appropriate values and click **Submit**.

   The default user name and password are **admin** and **password**, respectively. You can change the password and create additional users as described in "Insight Users" on page 100.

   The default user, admin, belongs to the Administrator group and has Role, Group and User privileges.

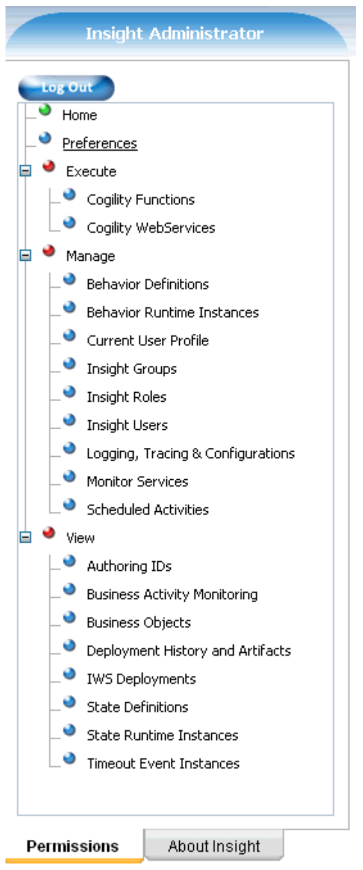
Cogility Insight Menu Tree

The Cogility Insight menu tree provides access to the functions provided by Cogility Insight. The menu items displayed are determined by the privileges assigned to the user who is logged in.These permissions are grouped into three categories:

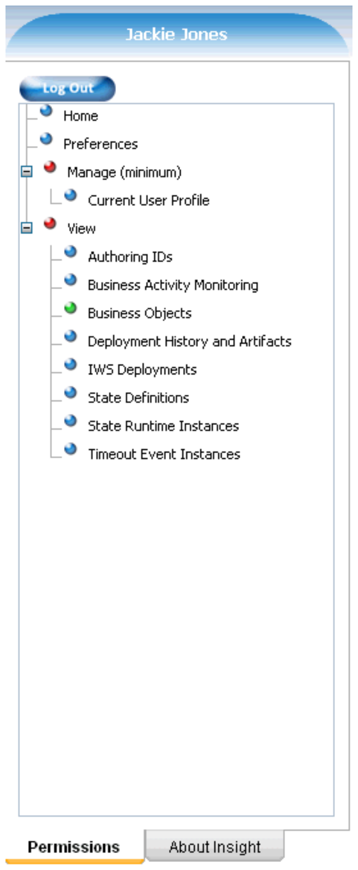
- Execute: Allows execution of internal and webservice functions
- Manage: Allows viewing and manipulation (change values or statuses, delete, etc) of various deployed artifacts.
- View: Search and view only, no manipulations

The default administrator user has access to all Cogility Insight functions. All user menus are a subset of the administrator menu.

The default administrator and business analyst user menus are shown below:



Administrator Menu



Business Analyst Menu

Each menu consists of two tabbed pages. The Permissions tab shows the functions available to the user. The About Insight tab provides details about the release and build of Cogility Studio which is running, as well as licensing information. In addition, information regarding the run-time environment, such as which application server is running, and the name of the pushed model and its version is shown.

The LogOut button is used to log out of Cogility Insight.

The following table lists each of the menu items and the related sub-menus in the Cogility Insight menu.

Table 0-1 Menu Items

Menu Item	Sub-menu	Description
Home		
Preferences		Displays and allows you to customize your session settings.
Execute	Cogility Functions	Provides access to certain internal Cogility functions that affect Cogility execution environment in the application server.
	Cogility Webservices	Allows you to execute a Cogility web service.

Table 0-1 Menu Items

Menu Item	Sub-menu	Description
Manage	Behavior Definitions	Displays the behavior name, the average completion time (in milliseconds), and the reference count (the number of behavior runtime instances).
	Behavior Runtime Instances	Lists each runtime instance of the behavior and allows you to delete it.
	Current User Profile	Allows you to update your user profile.
	Insight Groups	Allows you to manage groups. A group defines the roles for users belonging to the group.
	Insight Roles	Allows you to manage and assign roles. A role is a grouping of various permissions. Each permission is reflected as a "sub-menu" in Insight's navigation tree.
	Insight Users	Allows you to manage users.
	Logging, Tracing, & Configurations	Provides access to the logging, tracing, and configuration facilities.
	Monitor Services	Displays which monitor services have been started and also allows you to start or stop a service.
	Scheduled Activities	Allows you to define new reoccurring and non-reoccurring scheduled tasks.
View	Authoring IDs	Lists versioned model artifacts from the model that has been pushed by the Modeler tool.
	Business Activity Monitoring	Allows you to view the Chart Views and HTML Views created in your model.
	Business objects	Allows you to view the Business Objects in your model.
	Deployment History and Artifacts	Tracks each model version you have pushed onto the application server. Lists both full pushes and partial pushes.
	IWS Deployments	Displays information for IWS deployments that have been pushed to the server. Each IWS deployment is a grouping of various webservices and acts as a context path to those webservices.
	State Definitions	Displays definitions of all state machines that have been pushed to the server. A state machine definition describes what states (and their types) that the state machine includes.
	State Runtime Instances	Displays runtime instances of state definitions that have been created.
	Timeout Event Instances	Allows you view timeout events.

Home

Displays the Cogility Insight home page.

Preferences

You can change the way Cogility Insight displays information by setting the preferences. Modifications to the preference settings have effect only during the current session. For permanent changes, you must set the corresponding configuration parameters with the desired values prior to deploying Cogility Insight. See "Configuration" on page 13.

Setting preferences

To change or set preferences for the current Cogility Insight session:

1. From the menu tree, select **Preferences**.
2. In the Value column for a selected preference, enter a valid value.

   Changes are immediately applied.

   The valid values are explained in the Description column. Also, for making permanent changes to the preferences, the corresponding configuration parameter is shown in the Configuration Name column. See "Configuration" on page 13.

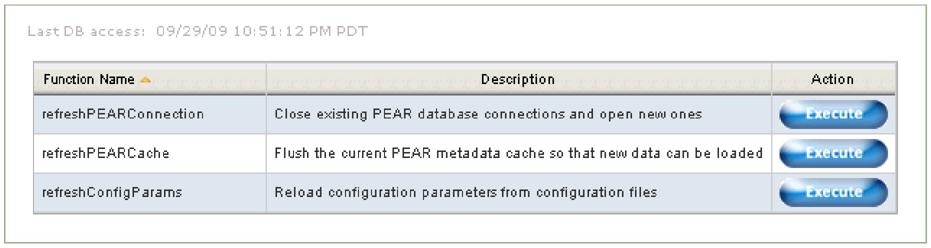
Execute

The Execute menu provides access to executing Cogility functions and webservices.

Cogility Functions

The Cogility Functions menu branch provides access to the dynamic model update functions described below:

- Refreshing PEAR connections
- Refreshing PEAR cache
- Refreshing configuration parameters



Last DB access: 09/29/09 10:51:12 PM PDT

Function Name	Description	Action
refreshPEARConnection	Close existing PEAR database connections and open new ones	Execute
refreshPEARCache	Flush the current PEAR metadata cache so that new data can be loaded	Execute
refreshConfigParams	Reload configuration parameters from configuration files	Execute

Figure 11-2: Internal functions

To execute Cogility functions:

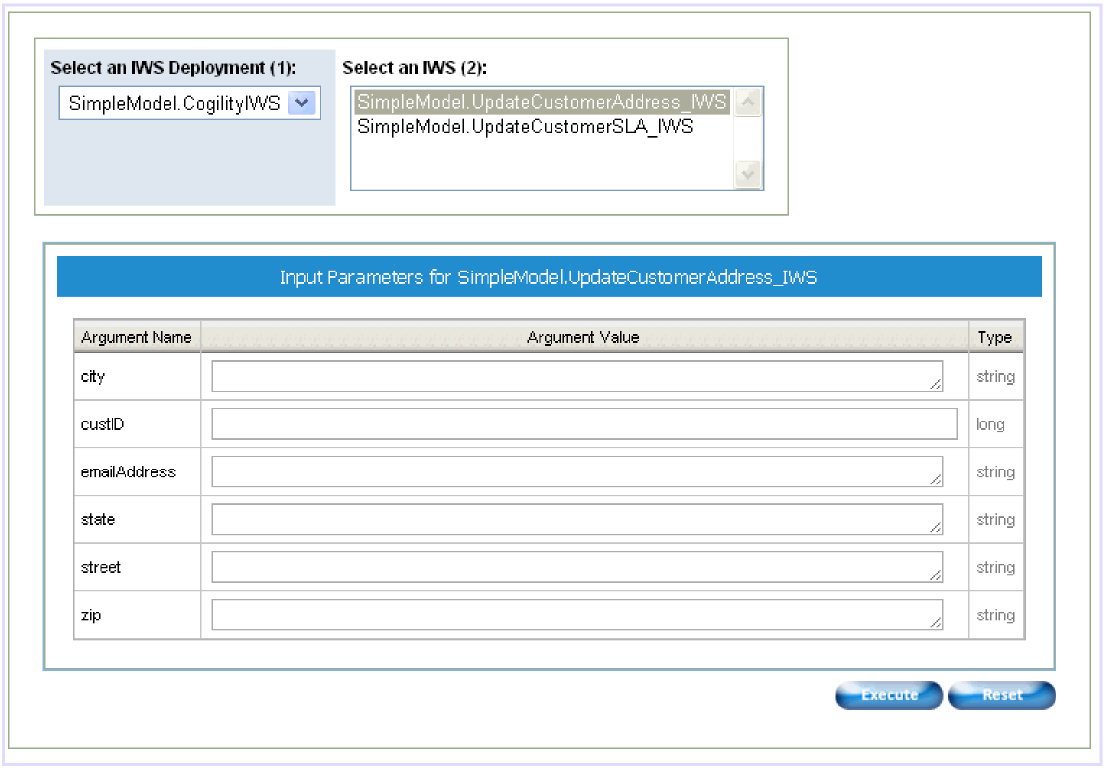
1. Log in to Cogility Insight.

   See "Running Cogility Insight" on page 90.
2. From the **Execute** menu, select **Cogility Functions**.
3. Click **Execute** next to the function you wish to execute.

Cogility Webservices

You can use Cogility Insight to execute an inbound web service. For testing outbound web services, see "Cogility Web Service Exerciser" on page 127.

Note: This testing function directly executes the OCL logic defined in the web service. It does not physically send a SOAP message to the J2EE application server. For situations that require SOAP level testing please use the Web Servicizer. Testing from within Insight is useful for unit testing while the OCL logic is being developed. For systems level testing, the Web Service Exerciser acts as a physical stand-in for an external system.


Select an IWS Deployment (1):
SimpleModel.CogilityIWS
Select an IWS (2):
SimpleModel.UpdateCustomerAddress_IWS
SimpleModel.UpdateCustomerSLA_IWS
Input Parameters for SimpleModel.UpdateCustomerAddress_IWS
Execute
Reset

Argument Name	Argument Value	Type
city		string
custID		long
emailAddress		string
state		string
street		string
zip		string

Figure 11-3: Web service execution

Executing web services

To execute a web service:

1. Log in to Cogility Insight.

   See "Running Cogility Insight" on page 90.
2. From the **Execute** menu, select **Cogility WebServices**.
3. From the **Select an IWS deployment** drop-down list, select the deployment.
4. From the **Select an IWS** list, select the web service.
5. In the **Input Parameters** section, enter the appropriate values and click **Execute**.

This sends the SOAP request encoded with the input values you entered and executes the web service. The web service returns a SOAP response to Cogility Insight with the response attribute value populated in the displayed field.

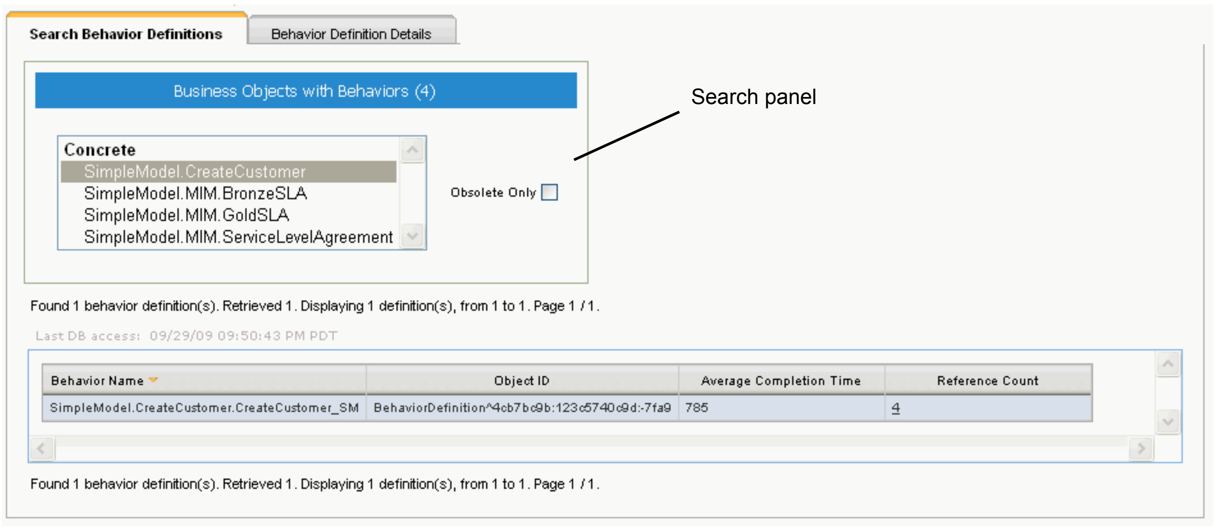
Manage

You can inspect and monitor your model's state and behavior definitions and behavior run-time instances in Cogility Insight.

Behavior Definitions

You can use Cogility Insight to view behavior definitions, including a count of the runtime instances for that behavior, and you can display these runtime instances.

The Behavior Definitions pane displays the behavior name, the average completion time (in milliseconds), and the reference count (the number of behavior runtime instances).


Search Behavior Definitions
Behavior Definition Details
Business Objects with Behaviors (4)
Concrete
SimpleModel.CreateCustomer
SimpleModel.MIM.BronzeSLA
SimpleModel.MIM.GoldSLA
SimpleModel.MIM.ServiceLevelAgreement
Obsolete Only
Search panel

Found 1 behavior definition(s). Retrieved 1. Displaying 1 definition(s), from 1 to 1. Page 1 / 1.

Last DB access: 09/29/09 09:50:43 PM PDT

Behavior Name	Object ID	Average Completion Time	Reference Count
SimpleModel.CreateCustomer.CreateCustomer_SM	BehaviorDefinition^4cb7bc9b:123c5740c9d:-7fa9	785	4

Found 1 behavior definition(s). Retrieved 1. Displaying 1 definition(s), from 1 to 1. Page 1 / 1.

Figure 11-4: Behavior definitions pane

To view a behavior definition:

1. Select **Behavior Definitions** in the Manage branch of the menu tree.
2. Select the business object you wish to view in the search panel.

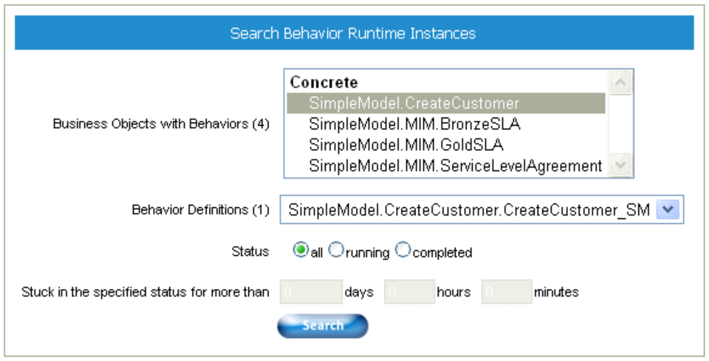
   The business object behavior list appear below the search panel. You can view the details of the behavior instances by clicking on the number in the Reference Count column.

   To view only those business objects with obsolete behaviors check the Obsolete Only checkbox.

Behavior definitions can become obsolete over time. As your business processes change, so can the state machines that represent them. The old state machines are not deleted when you re-deploy because there might be active runtime behaviors based on the old definition. Once you are sure that all active runtime behaviors are based on the new definition, you can remove the old ones. See "Behavior Runtime Instances" on page 96 . See also "Deleting completed state machines" on page 43.

Behavior Runtime Instances

The Behavior Runtime Instances screen lists each runtime instance of the behavior and allows you to delete it. For a selected runtime instance, you can view the process status, and within the process, you can select a specific state and view its status. This is shown in the figure below.


Search Behavior Runtime Instances

Business Objects with Behaviors (4): Concrete — SimpleModel.CreateCustomer (selected), SimpleModel.MIM.BronzeSLA, SimpleModel.MIM.GoldSLA, SimpleModel.MIM.ServiceLevelAgreement

Behavior Definitions (1): SimpleModel.CreateCustomer.CreateCustomer_SM

Status: [✓ all] running completed

Stuck in the specified status for more than 0 days 0 hours 0 minutes

Figure 11-5: Behavior run time instance search dialog

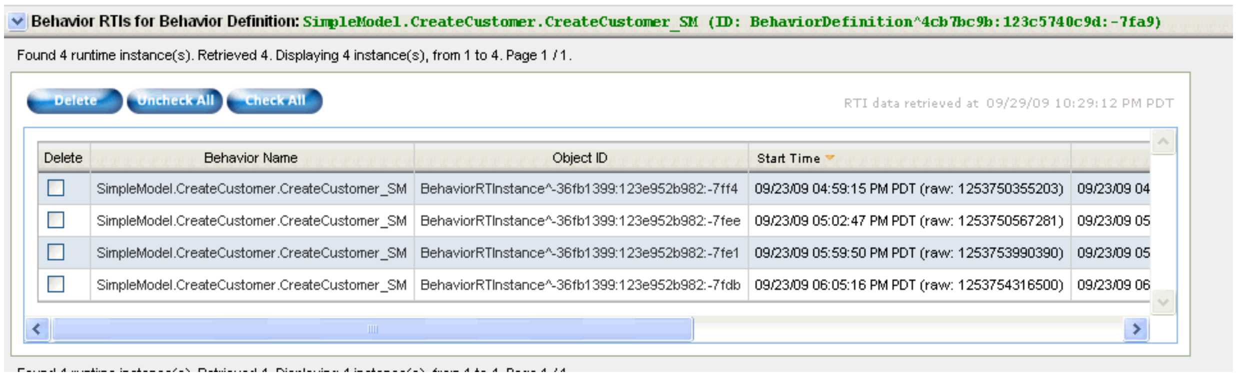
To view behavior runtime instances:

1. From the **Manage** menu, select **Behavior Runtime Instances**.

   The Search Behavior Runtime Instances window displays.

2. From the **Business Objects with Behaviors** list, select the business object whose behavior runtime instances you want to view.
3. From the **Behavior Definitions** drop-down list, select the behavior definition for the runtime instances you want to view.
4. For **Status**, select the status of those instances to be viewed (**All**, **Running** or **Completed**).
5. To view instances in a particular time span, enter the **days**, **hours** and **minutes** that describe the time since the instances were created (optional).
6. Click **Search**.

   The runtime details appear.


Behavior RTIs for Behavior Definition: SimpleModel.CreateCustomer.CreateCustomer_SM (ID: BehaviorDefinition^4cb7bc9b:123c5740c9d:-7fa9)

Found 4 runtime instance(s). Retrieved 4. Displaying 4 instance(s), from 1 to 4. Page 1 / 1.

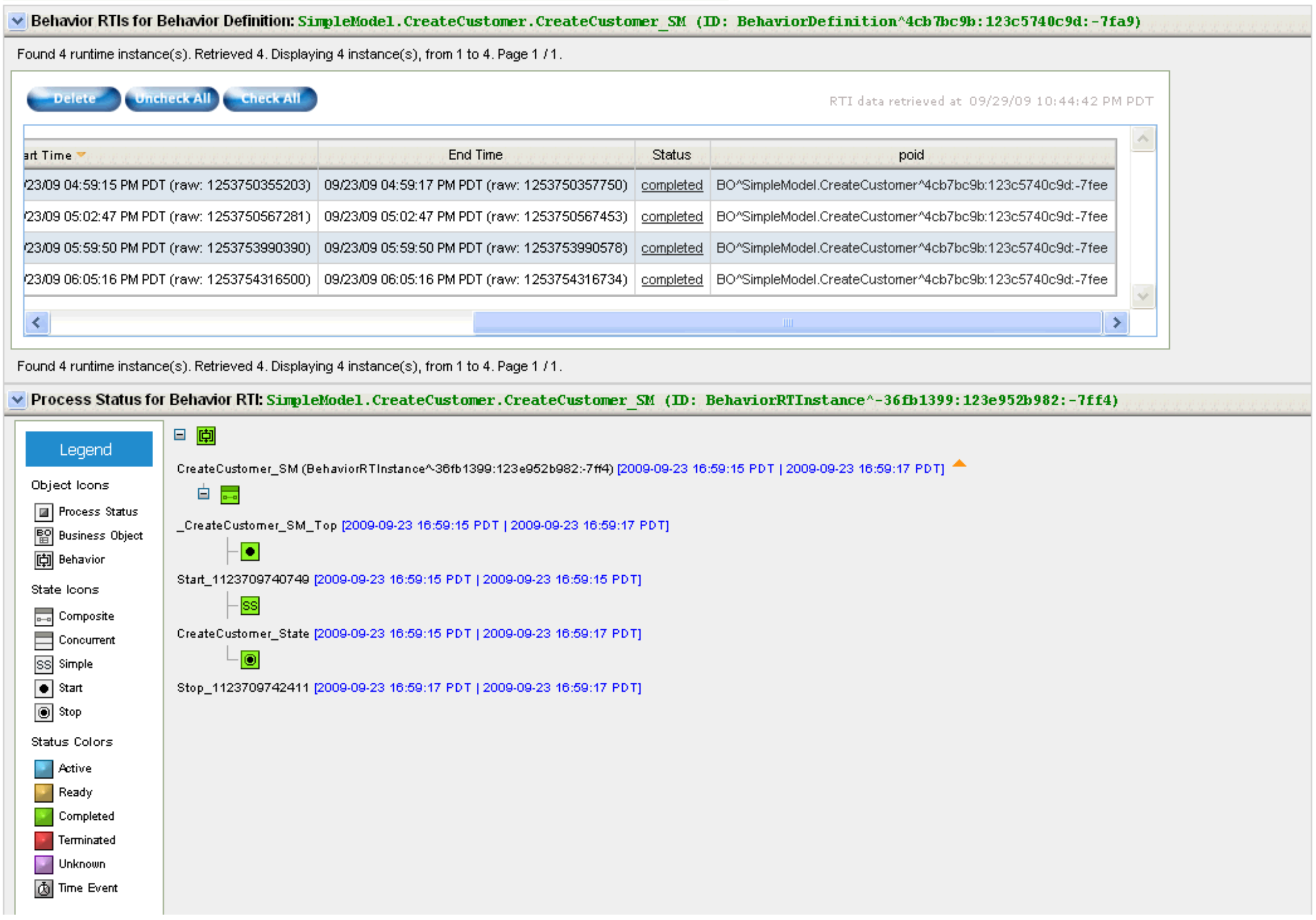
RTI data retrieved at 09/29/09 10:29:12 PM PDT

Delete	Behavior Name	Object ID	Start Time	
☐	SimpleModel.CreateCustomer.CreateCustomer_SM	BehaviorRTInstance^-36fb1399:123e952b982:-7ff4	09/23/09 04:59:15 PM PDT (raw: 1253750355203)	09/23/09 04
☐	SimpleModel.CreateCustomer.CreateCustomer_SM	BehaviorRTInstance^-36fb1399:123e952b982:-7fee	09/23/09 05:02:47 PM PDT (raw: 1253750567281)	09/23/09 05
☐	SimpleModel.CreateCustomer.CreateCustomer_SM	BehaviorRTInstance^-36fb1399:123e952b982:-7fe1	09/23/09 05:59:50 PM PDT (raw: 1253753990390)	09/23/09 05
☐	SimpleModel.CreateCustomer.CreateCustomer_SM	BehaviorRTInstance^-36fb1399:123e952b982:-7fdb	09/23/09 06:05:16 PM PDT (raw: 1253754316500)	09/23/09 06

To view process status details:

1. View the behavior runtime instances.
2. Scroll to the Status column.

3. Click on the status for the instance you wish to view.



Behavior RTIs for Behavior Definition: SimpleModel.CreateCustomer.CreateCustomer_SM (ID: BehaviorDefinition^4cb7bc9b:123c5740c9d:-7fa9)

Found 4 runtime instance(s). Retrieved 4. Displaying 4 instance(s), from 1 to 4. Page 1 / 1.

RTI data retrieved at 09/29/09 10:44:42 PM PDT

art Time	End Time	Status	poid
/23/09 04:59:15 PM PDT (raw: 1253750355203)	09/23/09 04:59:17 PM PDT (raw: 1253750357750)	completed	BO^SimpleModel.CreateCustomer^4cb7bc9b:123c5740c9d:-7fee
/23/09 05:02:47 PM PDT (raw: 1253750567281)	09/23/09 05:02:47 PM PDT (raw: 1253750567453)	completed	BO^SimpleModel.CreateCustomer^4cb7bc9b:123c5740c9d:-7fee
/23/09 05:59:50 PM PDT (raw: 1253753990390)	09/23/09 05:59:50 PM PDT (raw: 1253753990578)	completed	BO^SimpleModel.CreateCustomer^4cb7bc9b:123c5740c9d:-7fee
/23/09 06:05:16 PM PDT (raw: 1253754316500)	09/23/09 06:05:16 PM PDT (raw: 1253754316734)	completed	BO^SimpleModel.CreateCustomer^4cb7bc9b:123c5740c9d:-7fee

Found 4 runtime instance(s). Retrieved 4. Displaying 4 instance(s), from 1 to 4. Page 1 / 1.

Process Status for Behavior RTI: SimpleModel.CreateCustomer.CreateCustomer_SM (ID: BehaviorRTInstance^-36fb1399:123e952b982:-7ff4)

```
CreateCustomer_SM (BehaviorRTInstance^-36fb1399:123e952b982:-7ff4) [2009-09-23 16:59:15 PDT | 2009-09-23 16:59:17 PDT]
_CreateCustomer_SM_Top [2009-09-23 16:59:15 PDT | 2009-09-23 16:59:17 PDT]
Start_1123709740749 [2009-09-23 16:59:15 PDT | 2009-09-23 16:59:15 PDT]
CreateCustomer_State [2009-09-23 16:59:15 PDT | 2009-09-23 16:59:17 PDT]
Stop_1123709742411 [2009-09-23 16:59:17 PDT | 2009-09-23 16:59:17 PDT]
```

To delete behavior runtime instances:

1. View the behavior runtime instances.

   See “To view behavior runtime instances:” on page 96.

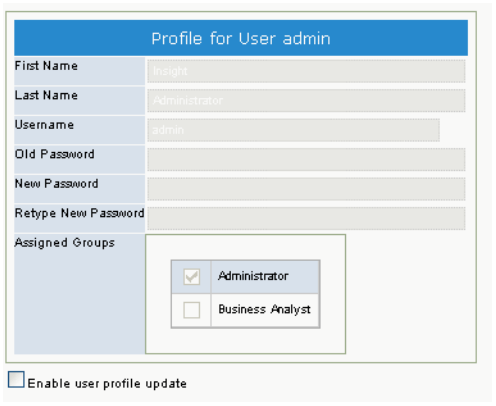
2. Check one or more boxes next to the behavior instance(s) that you want to delete.

   You can click the **Check all** link to check all the boxes. You cannot check boxes for instances that have not completed.

3. Click **Delete** to delete the selected instances.

Current User Profile

You can use the Current User Profile pane to view details on the current user. If you have administrator privileges, you can also use this pane to update the user profile.



Profile for User admin

First Name	Insight
Last Name	Administrator
Username	admin
Old Password	
New Password	
Retype New Password	
Assigned Groups	✓ Administrator ☐ Business Analyst

☐ Enable user profile update

To view the current user profile:

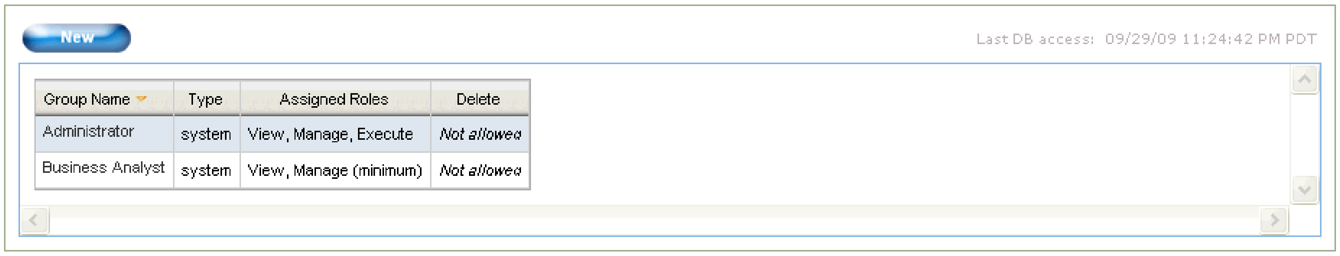
1. Select **Current User Profile** from the **Manage** menu.
2. If you have administrator privileges and wish to edit the user profile, check the **Enable User Profile Update** checkbox.
3. Update the profile as necessary and click **Update**.

Insight Groups

User management in Cogility Insight comprises three areas: roles, groups and users. If you are logged in as a user with Role, Group and User privileges you can create and modify these entities. When you define a role, you define the functions of Cogility Insight to which the role provides access. When you define a group, you assign the group one or more roles. When you create a user, you assign the user to a group. The user has access to all the functions defined by roles assigned to the group. In setting up your user management structure, start with roles, define the groups, then assign the users.

A group defines the roles for users belonging to the group. When you create a user (see "Insight Users" on page 100), you assign the user to one or more groups. Before you create or modify a group, define the roles as described in "Insight Roles" on page 99.

Two groups are already defined in Cogility Insight: an Administrator group, and a Business Analyst group. You can modify these as needed when you are logged in as a user in the Administrator group. If users are assigned to a deleted group, the users are not deleted.



Last DB access: 09/29/09 11:24:42 PM PDT

Group Name	Type	Assigned Roles	Delete
Administrator	system	View, Manage, Execute	*Not allowed*
Business Analyst	system	View, Manage (minimum)	*Not allowed*

Figure 11-6: Group administration

To create a group:

1. From the **Manage** menu, select **Insight Groups**.
2. Click **New**.
3. In the **Group Name** field, enter a name for the new group.
4. Assign one or more roles to the group.

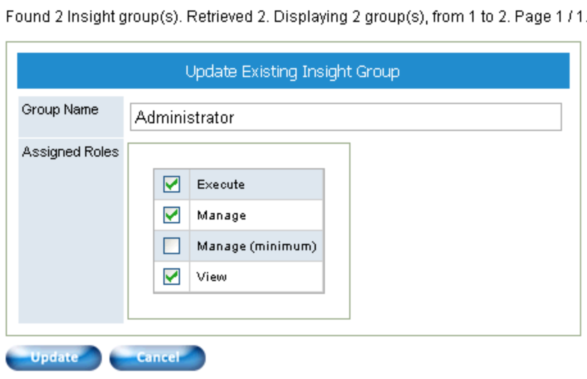
   Check the boxes next to the roles you want to assign. To define roles, see "Insight Roles" on page 99.
5. Click **Add**.

To modify a group:

1. Log in to Cogility Insight.

   See "Running Cogility Insight" on page 90.
2. From the **Manage** menu, select **Insight Groups**.
3. In the **Group Name** column, click on the group name.

   The Update Existing Insight Group dialog displays.



Found 2 Insight group(s). Retrieved 2. Displaying 2 group(s), from 1 to 2. Page 1 / 1.

Update Existing Insight Group

Group Name	Administrator
Assigned Roles	✓ Execute ✓ Manage ☐ Manage (minimum) ✓ View

4. If necessary, in the **Name** field, enter a new name.

   The group's roles and users are associated with the renamed group.
5. Check or uncheck the boxes next to the assigned roles.

   To define roles, see "Insight Roles" on page 99.
6. Click **Update**.

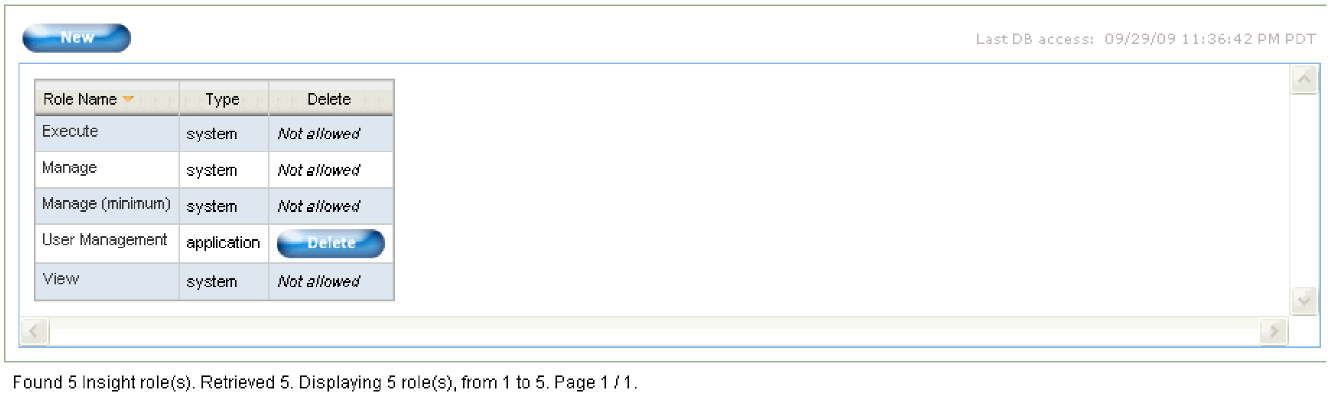
To delete a group:

1. From the **Manage** menu, select **Insight Groups**.
2. Click **Delete** to delete the group.

Insight Roles

When you define a role, you assign permissions to the role. Each permission corresponds to a particular function provided by Cogility Insight. When you define a group, you define its access by assigning roles. Create any new roles for a group before you create the group. You can also modify and delete existing roles.

Several predefined roles already exist in Cogility Insight. Use these steps to modify any roles you have created as well. To modify the role, you select or unselect the permissions assigned to the role.



Role Name	Type	Delete
Execute	system	*Not allowed*
Manage	system	*Not allowed*
Manage (minimum)	system	*Not allowed*
User Management	application	Delete
View	system	*Not allowed*

Found 5 Insight role(s). Retrieved 5. Displaying 5 role(s), from 1 to 5. Page 1 / 1.

Figure 11-7: Role administration

To create a new role:

1. From the **Manage** menu, select **Insight Roles**.
2. Click **New**.
3. Check one or more boxes to assign permissions.
4. Click **Add** to create the role.

To modify existing roles:

1. From the **Manage** menu, select **Insight Roles**.
2. In the **Role Name** column, click on the role name.
   The Update Existing Insight Role dialog displays.
3. Assign one or more permissions to the group.
   Check the boxes next to the permissions you want to assign.
4. Click **Update** to commit the modifications.

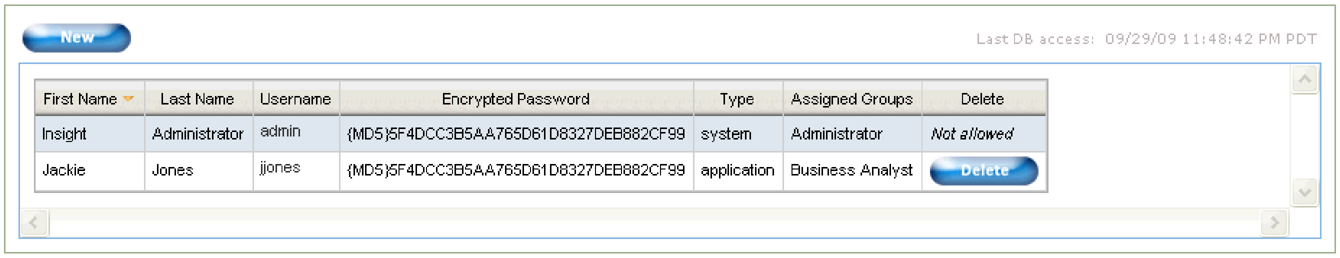
To delete a role from the role list:

1. From the **Manage** menu, select **Insight Roles**.
2. Click **Delete** to delete the role.

Insight Users

Users in Cogility Insight have privileges to use functionality. These privileges are defined by the user's role, which is specified by the group to which the user belongs. Once you have defined groups and roles, you can define users.

Cogility Insight has one predefined user, an admin that belongs to the Administrator group. You can modify this as needed when you are logged in as a user with User privileges. Use these steps to modify any users you have created as well.



First Name	Last Name	Username	Encrypted Password	Type	Assigned Groups	Delete
Insight	Administrator	admin	{MD5}5F4DCC3B5AA765D61D8327DEB882CF99	system	Administrator	*Not allowed*
Jackie	Jones	jjones	{MD5}5F4DCC3B5AA765D61D8327DEB882CF99	application	Business Analyst	Delete

Figure 11-8: User administration

To create a user:

1. From the **Manage** menu, select **Insight Roles**.
2. Click **New**.
   The Create New Insight User dialog displays.
3. Enter a **First Name**, **Last Name**, **User Name**, and **Password**.
   The password must be no more than nine characters and can consist of alpha and numeric characters.
4. Assign one or more groups to the user.
   Check the boxes next to the groups you want to assign.
5. Click **Add**.

To modify user information:

1. From the **Manage** menu, select **Insight Users**.
2. In the **Username** column, click on the username.
   The Update Existing Insight Users dialog displays.
3. Assign one or more groups to the user.
   Check the boxes next to the roles you want to assign.
4. Enter a new password if necessary.
5. Click **Update** to commit the modifications.

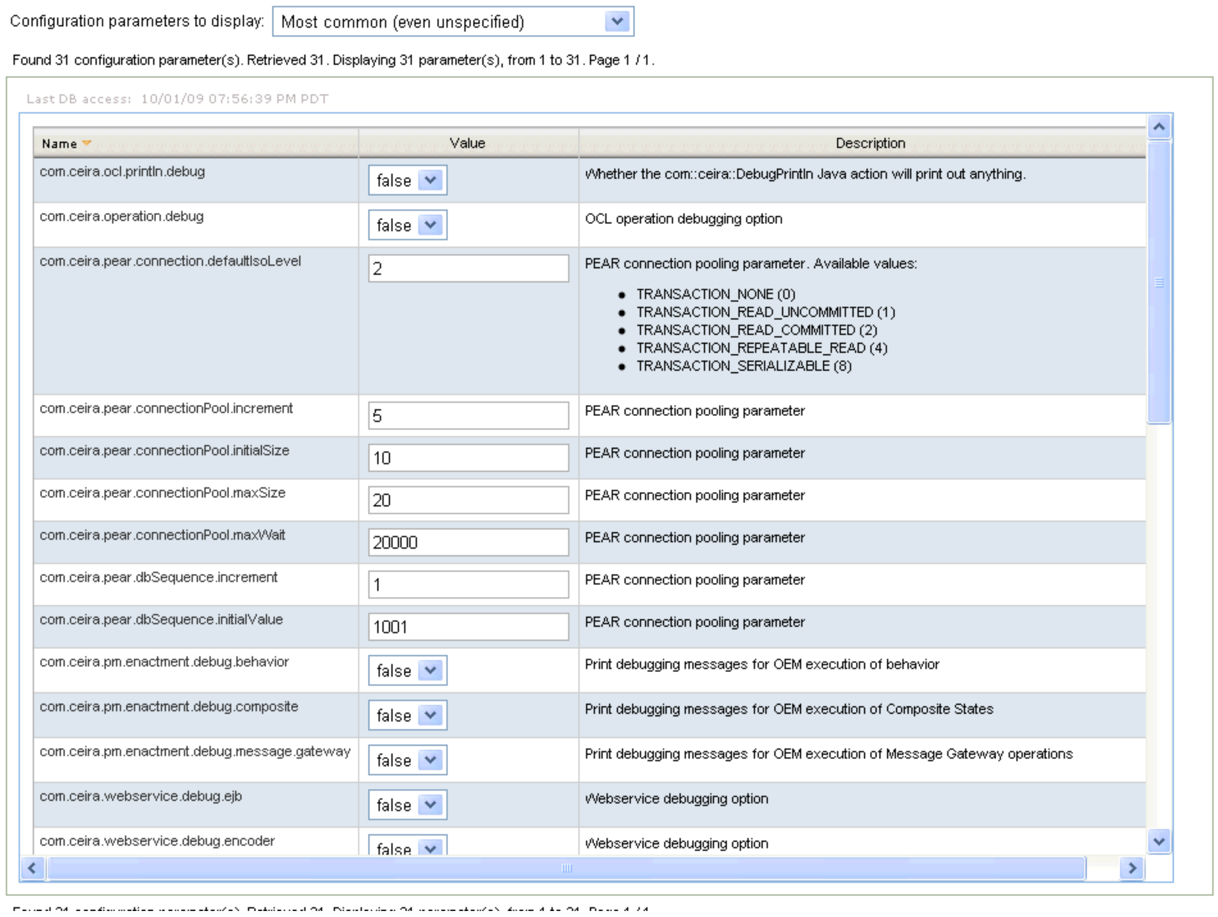
To delete one or more users:

1. From the **Manage** menu, select **Insight Users**.
2. Click **Delete** to delete the role.

Logging, Tracing, & Configurations

To provide for easier debugging and monitoring, Cogility Insight makes several debugging features available through its interface. These are overrides to the defined values for the corresponding configuration parameters. They do not change the values defined in the configuration files, only the in-memory values for the application server. Therefore, any changes you make to these settings are effective only during the current application server session, and if you reload the configuration parameters or repush the model, the settings will be reestablished according to the values in the configuration files. Only a user with the Administrator role will have access to these settings.

This page allows you to change the values for the most widely used configuration parameters. Modifications to these values should only be done by an Administrator since they will affect the whole application instead of just the logged-in session. Beware that if refresh the configuration parameters, the values in the configuration files will be reloaded, overriding any changes made through this page.



Configuration parameters to display: Most common (even unspecified)

Found 31 configuration parameter(s). Retrieved 31. Displaying 31 parameter(s), from 1 to 31. Page 1 / 1.

Last DB access: 10/01/09 07:56:39 PM PDT

Name	Value	Description
com.ceira.ocl.println.debug	false	Whether the com::ceira::DebugPrintln Java action will print out anything.
com.ceira.operation.debug	false	OCL operation debugging option
com.ceira.pear.connection.defaultIsoLevel	2	PEAR connection pooling parameter. Available values: • TRANSACTION_NONE (0) • TRANSACTION_READ_UNCOMMITTED (1) • TRANSACTION_READ_COMMITTED (2) • TRANSACTION_REPEATABLE_READ (4) • TRANSACTION_SERIALIZABLE (8)
com.ceira.pear.connectionPool.increment	5	PEAR connection pooling parameter
com.ceira.pear.connectionPool.initialSize	10	PEAR connection pooling parameter
com.ceira.pear.connectionPool.maxSize	20	PEAR connection pooling parameter
com.ceira.pear.connectionPool.maxWait	20000	PEAR connection pooling parameter
com.ceira.pear.dbSequence.increment	1	PEAR connection pooling parameter
com.ceira.pear.dbSequence.initialValue	1001	PEAR connection pooling parameter
com.ceira.pm.enactment.debug.behavior	false	Print debugging messages for OEM execution of behavior
com.ceira.pm.enactment.debug.composite	false	Print debugging messages for OEM execution of Composite States
com.ceira.pm.enactment.debug.message.gateway	false	Print debugging messages for OEM execution of Message Gateway operations
com.ceira.webservice.debug.ejb	false	Webservice debugging option
com.ceira.webservice.debug.encoder	false	Webservice debugging option

These configuration settings, their default values, valid ranges, and so forth, are described in "Run-time configuration" on page 15.

To change a configuration parameter setting with Cogility Insight:

1. From the **Manage** menu, select **Logging, Tracing & Configurations**.
2. Select the group of parameters you wish to reset from the drop-down list:
   - Most common (even unspecified)
   - With numeric values
   - With Boolean values
   - With names matching a regular expression
     - When selecting this option, you must also enter a search expression, then click **Search**.
   - All
3. Change the desired configuration parameters settings.

Monitor Services

A monitor service is a long-running and independent Cogility process that runs within a JavaEE application server and monitors for specific events generated by the Cogility execution environment deployed to the same server.

These services can be configured to automatically start and are automatically forced to shut down when the Cogility enterprise application stops. An administrator can check on the running status of a monitor service as well as start and stop it.

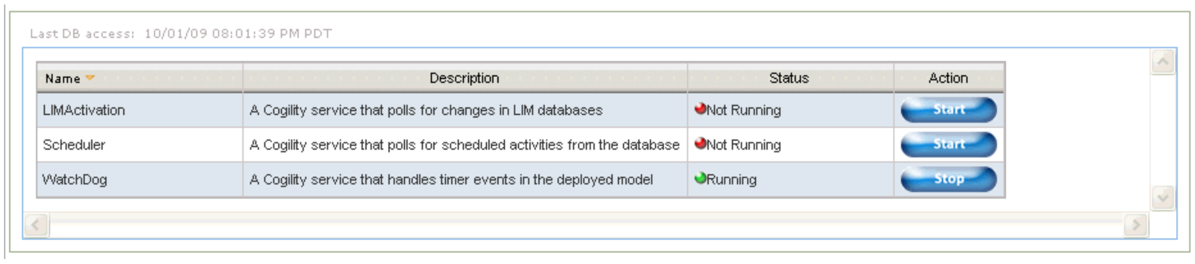
Cogility provides three monitor services: WatchDog, LIMActivation, and Scheduler. By default, only WatchDog is configured to start automatically. LIMActivation and Scheduler must be started manually. Their configurations are defined in the file `<DCHOME>\MB\Config\defaultConfigurations.txt` on Windows (or `$DCHOME/MB/Config/defaultConfigurations.txt` on Linux). For example:

```
# Number of services to register

com.cogility.monitorservice.count=3

# WatchDog - start upon Cogility execution in the appserver

com.cogility.monitorservice.1.name=WatchDog

com.cogility.monitorservice.1.class=com.ceira.pm.watchdog.WatchDogService

com.cogility.monitorservice.1.description=A Cogility service that handles
timer events in the deployed model

com.cogility.monitorservice.1.params=

com.cogility.monitorservice.1.enabled=true
```

The “enabled” flag value of “true” signifies that the service starts automatically and “false” signifies that the service must be started manually.

To start/stop monitored services:

1. From the **Manage** menu, select **Monitor Service Management**.



Last DB access: 10/01/09 08:01:39 PM PDT

Name	Description	Status	Action
LIMActivation	A Cogility service that polls for changes in LIM databases	● Not Running	Start
Scheduler	A Cogility service that polls for scheduled activities from the database	● Not Running	Start
WatchDog	A Cogility service that handles timer events in the deployed model	● Running	Stop

   The Status column indicates if the service is running or not running.

2. Click the button in the Action column to **Start** or **Stop** the service.

Scheduled Activities

The scheduling functionality allows you to schedule activities for execution. Without the scheduling functionality, the only way to schedule a task is to leverage operation system level scheduling to execute an "independent" snippet of OCL. Cogility allows independent snippets of OCL (free standing OCL that is not part of the model) to be wrapped inside a .BAT file on Windows or .sh on Unix and executed, either manually or by scheduling operating system level daemons.

Scheduling functionality allows scheduling to be done from within Cogility, through Cogility Insight. Once a Model has been pushed into execution, you can pick from a list of activities and supply additional scheduling details.

When you define a schedule in Cogility Insight, metadata describing the scheduled activity is stored in the database. The Schedule daemon periodically reads this metadata and determines when it is time to invoke one or more activities. This allows Scheduled Activities to be added or modified "on the fly" , that is, without manually restarting the Scheduler daemon.

Each Scheduled Activity is invoked in a separate thread so that the overall daemon and other threads are insulated from execution issues affecting a particular Activity.

There are two types of schedules you can create:

- Non-repeating Scheduled Activity
- Repeating Scheduled Activity

Activating Scheduling

The scheduling functionality is deactivated by default. You can start the service manually using the Monitor Services Management functionality (see “Monitor Services” on page 103 for instructions). You can also set the configuration parameter [com.cogility.scheduler.activate]. If you set this parameter to “true”, the service starts automatically when Cogility is started. See “Customizing installation configuration” on page 13 for information on setting configuration parameters.

Non-Repeating Scheduled Activities

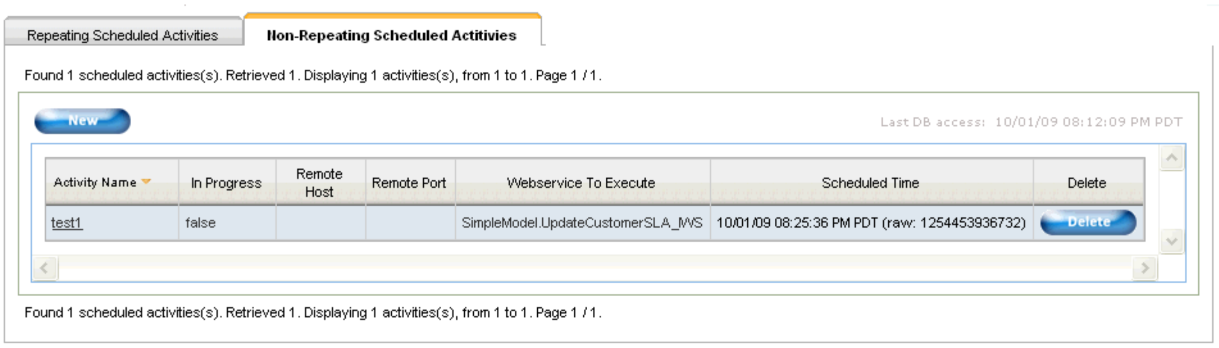
Non-repeating activities are invoked once and then automatically deleted from the database by the Scheduler.

To view a non-repeating scheduled activity:

1. From the **Manage** menu, select **Scheduled Activities**.
2. Open the **Non-Repeating Scheduled Activities** tab.

   All non-repeating activities that are scheduled appear in the schedule list.

   a


Repeating Scheduled Activities
Non-Repeating Scheduled Actitivies
New
Delete

Found 1 scheduled activities(s). Retrieved 1. Displaying 1 activities(s), from 1 to 1. Page 1 / 1.

Last DB access: 10/01/09 08:12:09 PM PDT

Activity Name	In Progress	Remote Host	Remote Port	Webservice To Execute	Scheduled Time	Delete
test1	false			SimpleModel.UpdateCustomerSLA_WS	10/01/09 08:25:36 PM PDT (raw: 1254453936732)	Delete

Found 1 scheduled activities(s). Retrieved 1. Displaying 1 activities(s), from 1 to 1. Page 1 / 1.

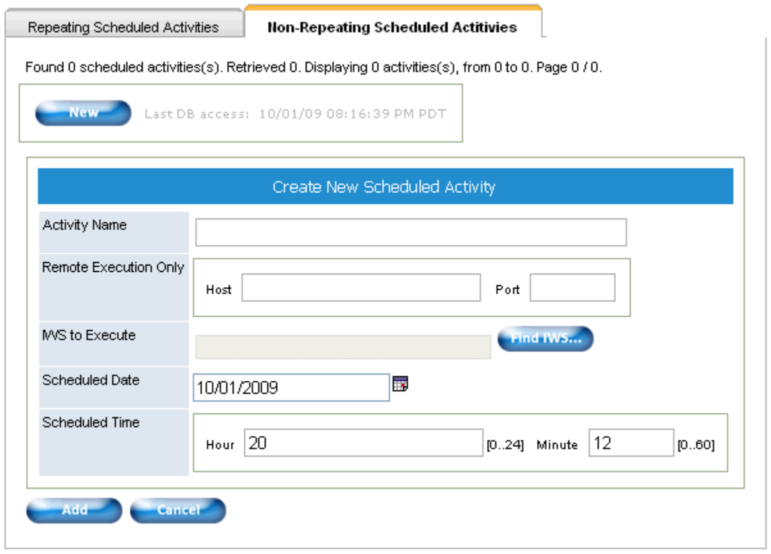
You can delete a scheduled non-repeating activity by clicking **Delete**.

To create a non-repeating scheduled activity:

1. From the **Manage** menu, select **Scheduled Activities**.

2. Open the **Non-Repeating Scheduled Activity** tab.
3. Click **New**.

   The Create New Scheduled Activity dialog appears.


Repeating Scheduled Activities
Non-Repeating Scheduled Actitivies
New
Find IWS...
Add
Cancel

Found 0 scheduled activities(s). Retrieved 0. Displaying 0 activities(s), from 0 to 0. Page 0 / 0.

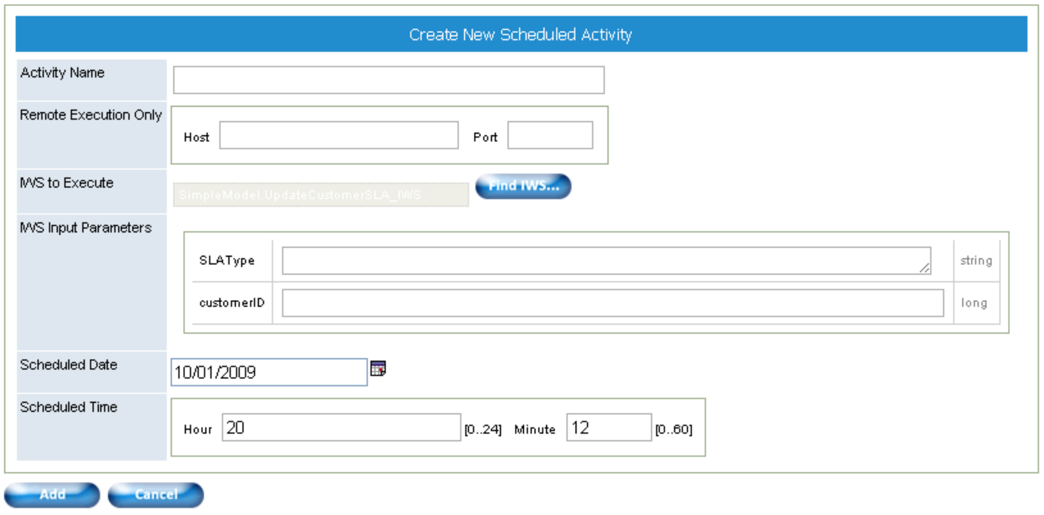
Last DB access: 10/01/09 08:16:39 PM PDT

Create New Scheduled Activity

Activity Name	
Remote Execution Only	Host: Port:
IWS to Execute	
Scheduled Date	10/01/2009
Scheduled Time	Hour 20 [0..24] Minute 12 [0..60]

4. Type a name in the **Activity Name** field.
5. Enter a **Host** and **Port** if you are using remote execution.
6. Click **Find IWS**.
   a. Select an IWS Deployment from the IWS Deployment list.

      After selecting the IWS Deployment, a list of IWS appears.
   b. Select the IWS from the list.
   c. Click **OK**.
7. After you select the inbound web service, the **IWS Parameters** field appears with an entry field for each parameter for the selected inbound web service.



Create New Scheduled Activity

Activity Name	
Remote Execution Only	Host: Port:
IWS to Execute	SimpleModel.UpdateCustomerSLA_IWS
IWS Input Parameters	SLAType (string); customerID (long)
Scheduled Date	10/01/2009
Scheduled Time	Hour 20 [0..24] Minute 12 [0..60]

8. Enter the parameter values.

9. Enter the date that this scheduled activity should be run in the **Scheduled Date** field. You can click the button to select the date from a calendar.
10. Enter the hour and minute for the time at which the activity should run in the **Scheduled Time** hour field and minute field. The hour is specified using the 24-hour format.
11. Click **Add**.

Sometimes it may be necessary to change the schedule for a non-repeating activity. You can edit all fields in the schedule except the activity name.

To modify a non-repeating activity:

1. From the **Manage** menu, select **Scheduled Activities**.
2. Open the **Non-Repeating Scheduled Activity** tab.
3. Click on the activity name for the activity you wish to modify.
4. Make the necessary changes.
5. Click **Update**.

Repeating Scheduled Activities

Repeating Activities are actvities that are scheduled to run at regular intervals. Repeating activities remain registered in the database until you remove them using Cogility Insight. The supported repetition intervals are month, day(of month), hours, and minutes.

The valid values for the date and time fields are:

Months:	1 through 12
Day:	1 through 31 (You can use a negative number to denote the number of days before the end of the month. For example -2 would denote two days before the end of the month.)
Hour:	0 through 24
Minute:	0 through 60

Multiple values can be specified in a field by separating them with commas (',') and no white space. A value of '*' indicates all. The following table shows some examples of using multiple values to specify repeating activities.

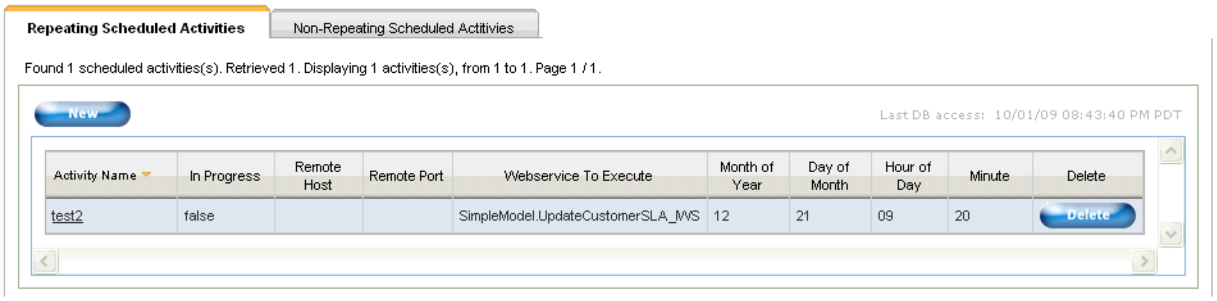
Month	Day	Hour	Minute	Scheduling Logic
3	12	14	2	Run on 12th day of March (each year) at 2:02 PM
*	1	10	0	1st day of each month at 10AM
*	-1	11	0	Last day of each month at 11AM (NOTE: -ve numbers are only supported for the day field.
*	-2	10	0	2nd last day of each month at 10AM
*	1,15,30	18	0	1st, 15th and 30th of each month at 6 PM.
1,4,7,10	1	08	0	1st day of Jan, Apr, Jul and Oct at 8 AM
*	*	06	30	Daily at 6:30 AM

To view a repeating scheduled activity:

1. From the **Manage** menu, select **Scheduled Activities**.
2. Open the **Repeating Scheduled Activities** tab.

All repeating activities that are scheduled appear in the schedule list.

a


Repeating Scheduled Activities
Non-Repeating Scheduled Actitivies
New
Delete

Found 1 scheduled activities(s). Retrieved 1. Displaying 1 activities(s), from 1 to 1. Page 1 / 1.

Last DB access: 10/01/09 08:43:40 PM PDT

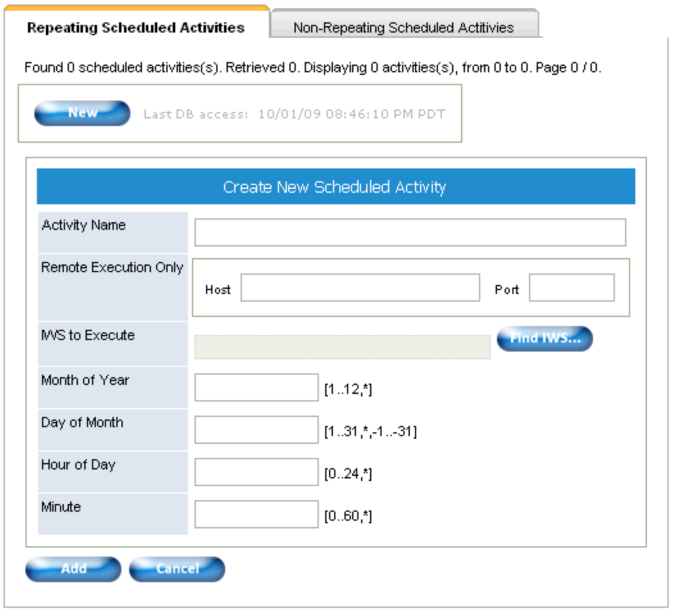
Activity Name	In Progress	Remote Host	Remote Port	Webservice To Execute	Month of Year	Day of Month	Hour of Day	Minute	Delete
test2	false			SimpleModel.UpdateCustomerSLA_IWS	12	21	09	20	Delete

You can delete a scheduled repeating activity by clicking **Delete**.

To create a repeating scheduled activity:

1. From the **Manage** menu, select **Scheduled Activities**.
2. Open the **Repeating Scheduled Activity** tab.
3. Click **New**.

   The Create New Scheduled Activity dialog appears.


Repeating Scheduled Activities
Non-Repeating Scheduled Actitivies
New
Find IWS...
Add
Cancel

Found 0 scheduled activities(s). Retrieved 0. Displaying 0 activities(s), from 0 to 0. Page 0 / 0.

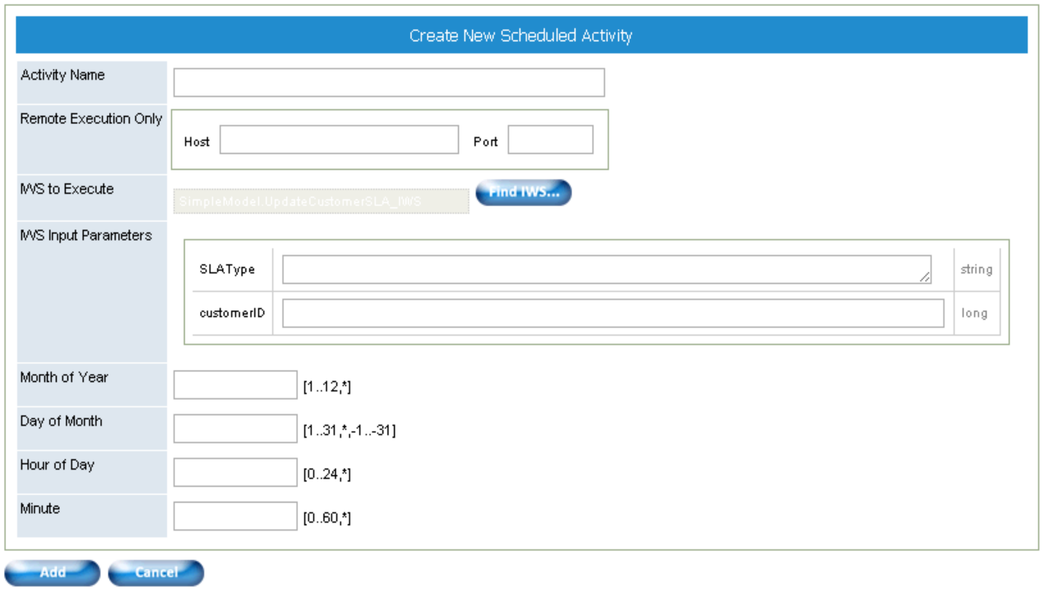
Last DB access: 10/01/09 08:46:10 PM PDT

Create New Scheduled Activity

Activity Name	
Remote Execution Only	Host Port
IWS to Execute	
Month of Year	[1..12,*]
Day of Month	[1..31,*,-1..-31]
Hour of Day	[0..24,*]
Minute	[0..60,*]

4. Type a name in the **Activity Name** field.
5. Enter a **Host** and **Port** if you are using remote execution.
6. Click **Find IWS**.
   a. Select an IWS Deployment from the IWS Deployment list.
      After selecting the IWS Deployment, a list of IWS appears.
   b. Select the IWS from the list.
   c. Click **OK**.

7. After you select the inbound web service, the **IWS Parameters** field appears with an entry field for each parameter for the selected inbound web service.



Activity Name	
Remote Execution Only	Host · Port
IWS to Execute	SimpleModel.UpdateCustomerSLA_IWS
IWS Input Parameters	SLAType (string); customerID (long)
Month of Year	[1..12,*]
Day of Month	[1..31,*,-1..-31]
Hour of Day	[0..24,*]
Minute	[0..60,*]

8. Enter the parameter values.
9. Enter the Month, Day, Hour, and Minute in the appropriate fields.
10. Click **Add**.

Sometimes it may be necessary to change the schedule for a non-repeating activity. You can edit all fields in the schedule except the activity name.

To modify a repeating activity:

1. From the **Manage** menu, select **Scheduled Activities**.
2. Open the **Repeating Scheduled Activity** tab.
3. Click on the activity name for the activity you wish to modify.
4. Make the necessary changes.
5. Click **Update**.

Scheduling Programmatically

In addition to scheduling using Cogility Insight, you can also schedule activities programmatically. This adds to the ways in which control can by asynchronously passed from one piece of model logic to another. Using the scheduling functionality, a Web Service can be asynchronously invoked by another Web Service or State Machine.

Non-Repeating Activity

The OCL snippet below schedules a Web Service named 'CreateCustomers' for immediate (earliest possible - the Scheduler normally runs at 1 minute intervals). Additionally, it also passes in a value of 9000 for an input parameter named 'totalCount' and 1 for the input 'numParts'.

```
ACT : com::cogility::scheduler::ScheduledActivity;

min : java::lang::Integer := 0;
```

```
ACT := ACT.scheduleNonRepeatingActivity('Create
Customers','SimpleModel.CreateCustomers',min);

ACT.setInput('totalCount',9000);

ACT.setInput('numParts',1);
```

Repeating Activity

Repeating Activities are generally easy enough to schedule from Cogility Insight thus they are generally not programmatically scheduled. If needed, it can be done as in the following example.

```
ACT : com::cogility::scheduler::ScheduledActivity;
min : java::lang::Integer := 0;
ACT := ACT.scheduleRepeatingActivity('Create
Customers','SimpleModel.RunMonthlyJobsReport','*', '1', '10', '0', null,
null);
```

This example shows how to schedule a repeating activity for each month (‘*) on the first (‘1’) of the month at 10 AM and 0 minutes. The last two parameters are null to indicate default appserver location and port number. The program that schedules the activity is already running inside the app-server so the default values of null are sufficient.

Scheduler Logging

By default, log files for the Scheduler go into the Logs subdirectory under Cogility's main MB directory. Please note that if multiple installations of Cogility exist, the log files will, by default, go under the individual installations. Change com.cogility.scheduler.logDirectory to specify an appropriate location for the log files.

The files naming pattern is "Scheduler_YYYY-MM-DD_HH.MM.c, where c is the log file count. The count starts at 0 and each time the current log file exceeds 2MB, another log file is automatically created and the count goes up. The Scheduler will keep up to 50 files before rotating through the files and the file naming pattern changes (timestamp part of the name) each time the Scheduler is restarted (another 50 files with that pattern before the files are rotated).

In-Progress activities

When a scheduled activity begins to run, it is marked as In-Progress. Some of these activities can be very long-lived. If the application server is shut-down and restarted while the activity is still in-progress, the application server will continue to see the activity as in-progress.

Because it is still marked in-progress, the activity will not run at its next scheduled time. You must use Insight to manually remove the In-Progress flag from the activity. This allows the activity to run at its next scheduled time.

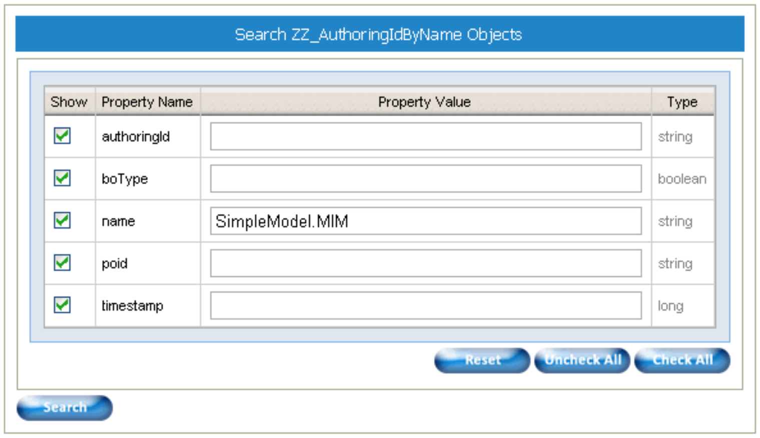
View

The View menu provides access to view various deployment properties and history.

Authoring IDs

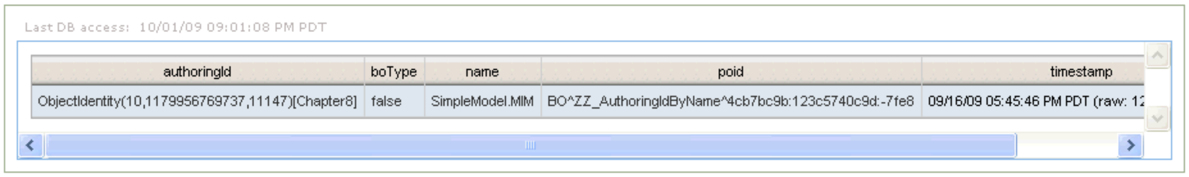
You can view a list of defined model artifact run-time instances. An authoring ID is assigned to each version of an artifact as it is turned over. Properties included with the authoring ID information are the name of the artifact, its POID (persistent object ID), the time when it was turned over, and a discrete authoring ID that has the version name appended to it. You can view a list of the authoring IDs for all artifacts turned over, sorted by these properties.

Note: To see the time stamp (which is of type long) in a user friendly format the com.cogility.insight.date.print debug option must be set to true.


Search ZZ_AuthoringIdByName Objects

Show	Property Name	Property Value	Type
✓	authoringId		string
✓	boType		boolean
✓	name	SimpleModel.MIM	string
✓	poid		string
✓	timestamp		long

Found 1 business object(s). Retrieved 1. Displaying 1 object(s), from 1 to 1. Page 1 / 1.


Last DB access: 10/01/09 09:01:08 PM PDT

authoringId	boType	name	poid	timestamp
ObjectIdentity(10,1179956769737,11147)[Chapter8]	false	SimpleModel.MIM	BO^ZZ_AuthoringIdByName^4cb7bc9b:123c5740c9d:-7fe8	09/16/09 05:45:46 PM PDT (raw: 12

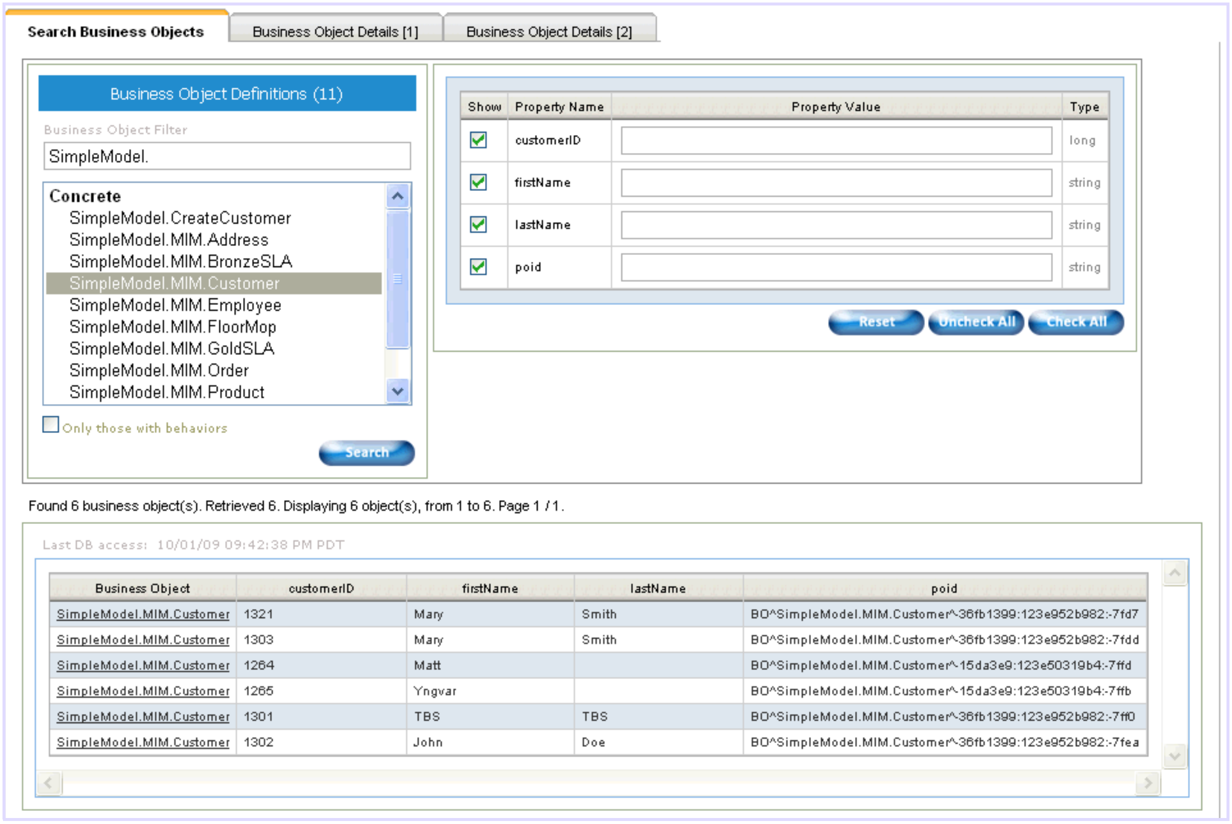
To view authoring IDs of deployed artifacts:

1. In the **View** menu, select **Authoring IDs**.
2. Check the boxes for the properties you want to display.
3. To search for specific instances, identified by property, enter the label for the property in the adjacent field.

   Leave the fields blank to search for all instances.
4. Click **Search**.

Business objects

The business object classes in your model are instanciated during execution, and their instances are written to the run-time repository. You can view these and their associations in Cogility Insight.



Show	Property Name	Property Value	Type
✓	customerID		long
✓	firstName		string
✓	lastName		string
✓	poid		string

Found 6 business object(s). Retrieved 6. Displaying 6 object(s), from 1 to 6. Page 1 / 1.

Last DB access: 10/01/09 09:42:38 PM PDT

Business Object	customerID	firstName	lastName	poid
SimpleModel.MIM.Customer	1321	Mary	Smith	BO^SimpleModel.MIM.Customer^-36fb1399:123e952b982:-7fd7
SimpleModel.MIM.Customer	1303	Mary	Smith	BO^SimpleModel.MIM.Customer^-36fb1399:123e952b982:-7fdd
SimpleModel.MIM.Customer	1264	Matt		BO^SimpleModel.MIM.Customer^-15da3e9:123e50319b4:-7ffd
SimpleModel.MIM.Customer	1265	Yngvar		BO^SimpleModel.MIM.Customer^-15da3e9:123e50319b4:-7ffb
SimpleModel.MIM.Customer	1301	TBS	TBS	BO^SimpleModel.MIM.Customer^-36fb1399:123e952b982:-7ff0
SimpleModel.MIM.Customer	1302	John	Doe	BO^SimpleModel.MIM.Customer^-36fb1399:123e952b982:-7fea

To view business objects:

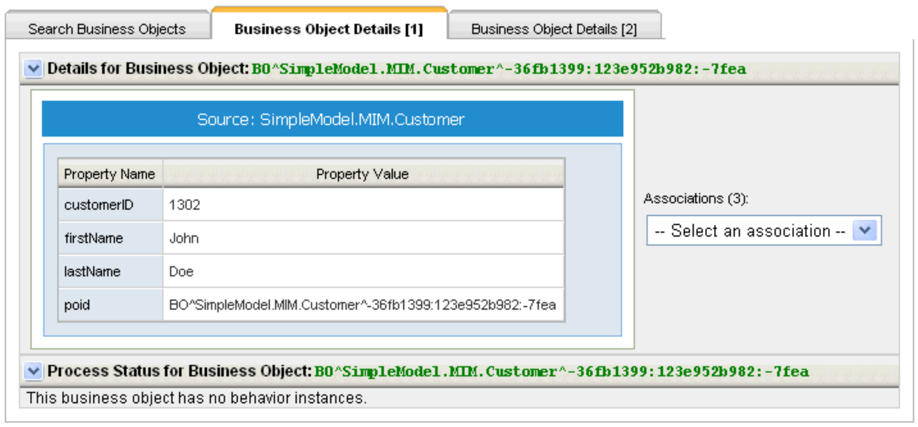
1. From the **View** menu, select **Business Objects**.

   The business object search view displays.
2. From the **Business Objects** list, select the class for the object you want to view.
3. In the properties window, enter the filtering criteria for searching objects.

   If you want to view a list of all objects, leave these fields empty and the check boxes checked.
4. Click **Search**.

   The list of business objects displays.

5. In the business objects list, in the **Business Objects** column, click the link for a specific object instance to see the details for that object.


Search Business Objects
Business Object Details [1]
Business Object Details [2]
Associations (3):
-- Select an association --

Details for Business Object: BO^SimpleModel.MIM.Customer^-36fb1399:123e952b982:-7fea

Source: SimpleModel.MIM.Customer

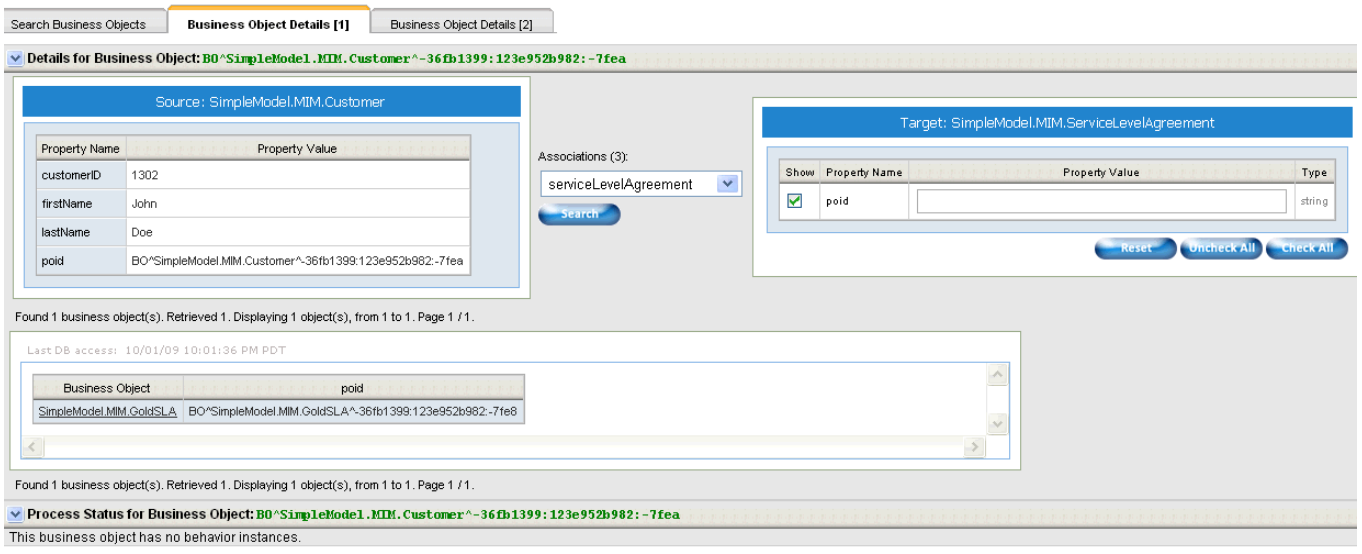
Property Name	Property Value
customerID	1302
firstName	John
lastName	Doe
poid	BO^SimpleModel.MIM.Customer^-36fb1399:123e952b982:-7fea

Process Status for Business Object: BO^SimpleModel.MIM.Customer^-36fb1399:123e952b982:-7fea

This business object has no behavior instances.

6. To view associations, select an association from the Associations drop-down list and click **Search**.
7. In the properties window, enter the filtering criteria for searching associations and click **Search**.

Note: When searching for business objects, including Authoring IDs, you can specify an SQL expression in the property value fields. Any property value that starts with "sql:" is recognized as an SQL expression. For example, if you want to search for business objects whose names contain the string "Customer", you might want to enter "sql: LIKE '%Customer%'" in the Property Value field.



Details for Business Object: BO^SimpleModel.MIM.Customer^-36fb1399:123e952b982:-7fea

Source: SimpleModel.MIM.Customer

Property Name	Property Value
customerID	1302
firstName	John
lastName	Doe
poid	BO^SimpleModel.MIM.Customer^-36fb1399:123e952b982:-7fea

Target: SimpleModel.MIM.ServiceLevelAgreement

Show	Property Name	Property Value	Type
✓	poid		string

Found 1 business object(s). Retrieved 1. Displaying 1 object(s), from 1 to 1. Page 1 / 1.

Last DB access: 10/01/09 10:01:36 PM PDT

Business Object	poid
SimpleModel.MIM.GoldSLA	BO^SimpleModel.MIM.GoldSLA^-36fb1399:123e952b982:-7fe8

Found 1 business object(s). Retrieved 1. Displaying 1 object(s), from 1 to 1. Page 1 / 1.

Process Status for Business Object: BO^SimpleModel.MIM.Customer^-36fb1399:123e952b982:-7fea

This business object has no behavior instances.

Business Activity Monitoring

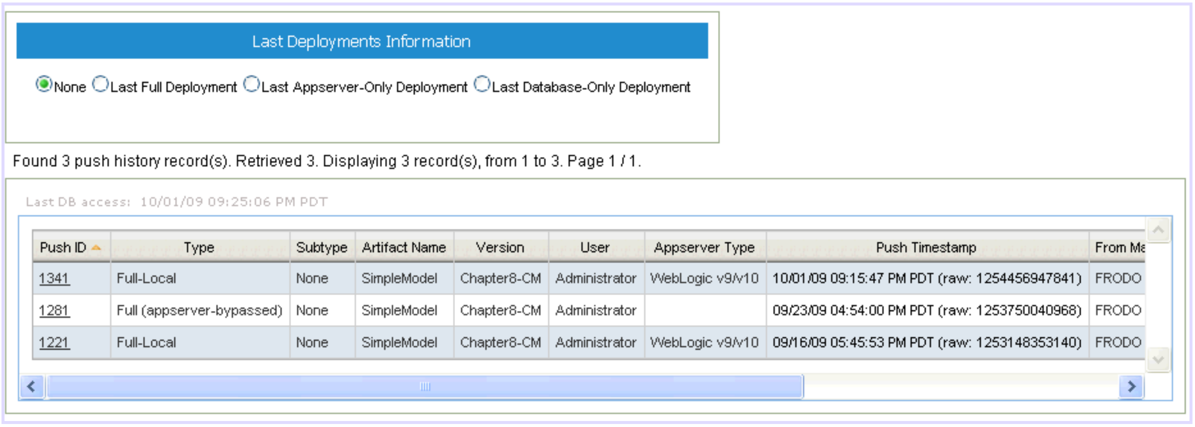
The Business Activity Monitoring menu item allows you to view the Charts and HTML views in your model.

To view charts, select the desired chart from the drop-down list on the Charts View tab.

To view HTML views, select the desired HTML view from the HTML Views tab.

Deployment History and Artifacts

Cogility Insight keeps track of each model version you have pushed onto the application server in the deployment history. You can see full pushes and partial pushes. The Subtype column of the displayed results identifies partial pushes; full pushes have no subtype.



[✓ None] Last Full Deployment Last Appserver-Only Deployment Last Database-Only Deployment

Found 3 push history record(s). Retrieved 3. Displaying 3 record(s), from 1 to 3. Page 1 / 1.

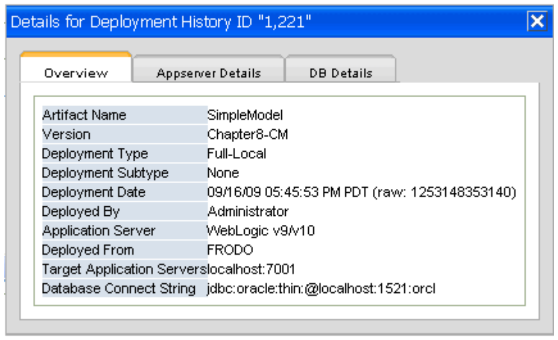
Last DB access: 10/01/09 09:25:06 PM PDT

Push ID	Type	Subtype	Artifact Name	Version	User	Appserver Type	Push Timestamp	From Ma
1341	Full-Local	None	SimpleModel	Chapter8-CM	Administrator	WebLogic v9/v10	10/01/09 09:15:47 PM PDT (raw: 1254456947841)	FRODO
1281	Full (appserver-bypassed)	None	SimpleModel	Chapter8-CM	Administrator		09/23/09 04:54:00 PM PDT (raw: 1253750040968)	FRODO
1221	Full-Local	None	SimpleModel	Chapter8-CM	Administrator	WebLogic v9/v10	09/16/09 05:45:53 PM PDT (raw: 1253148353140)	FRODO

To view deployment history:

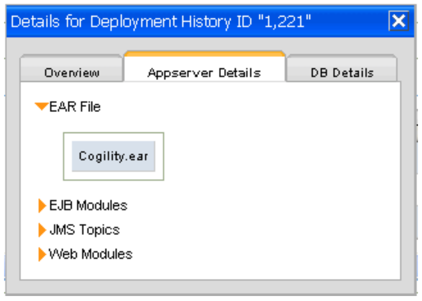
1. From the **View** menu, select **Deployment History and Artifacts**.
2. To view a specific deployment, in the **Push ID** column, click the link for the push ID of the deployment.

   You can view details on the full pushes only.



Artifact Name	SimpleModel
Version	Chapter8-CM
Deployment Type	Full-Local
Deployment Subtype	None
Deployment Date	09/16/09 05:45:53 PM PDT (raw: 1253148353140)
Deployed By	Administrator
Application Server	WebLogic v9/v10
Deployed From	FRODO
Target Application Servers	localhost:7001
Database Connect String	jdbc:oracle:thin:@localhost:1521:orcl

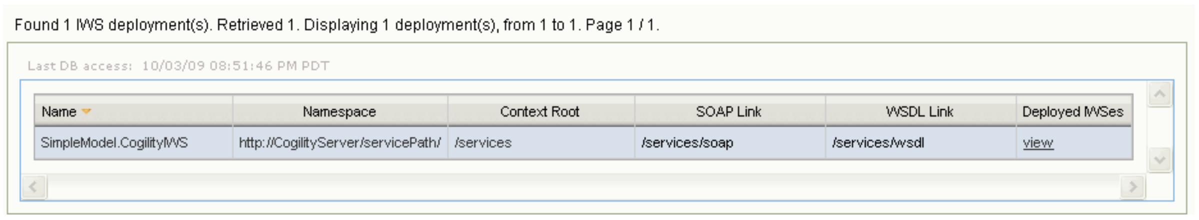
In the Appserver Details and DB Details tabs, click on the arrow to see additional details.



IWS Deployments

With Cogility Insight you can view the deployments for your model’s inbound web services and execute inbound web services. For testing outbound web services, see “Cogility Web Service Exerciser” on page 127.

Cogility Insight verifies that a web service is deployed correctly on the application server. For each named inbound web service deployment, the IWS Deployments screen describes the name and context root. The context root of the web service deployment for the example below is specified as /**services**, meaning that the web services are located at this path relative to the URL of the application server. The name for context root is specified as part of its deployment element in the model.



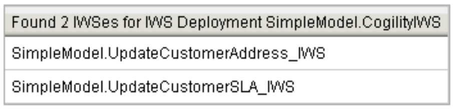
Found 1 IWS deployment(s). Retrieved 1. Displaying 1 deployment(s), from 1 to 1. Page 1 / 1.

Last DB access: 10/03/09 08:51:46 PM PDT

Name	Namespace	Context Root	SOAP Link	WSDL Link	Deployed IWSes
SimpleModel.CogilityIWS	http://CogilityServer/servicePath/	/services	/services/soap	/services/wsdl	view

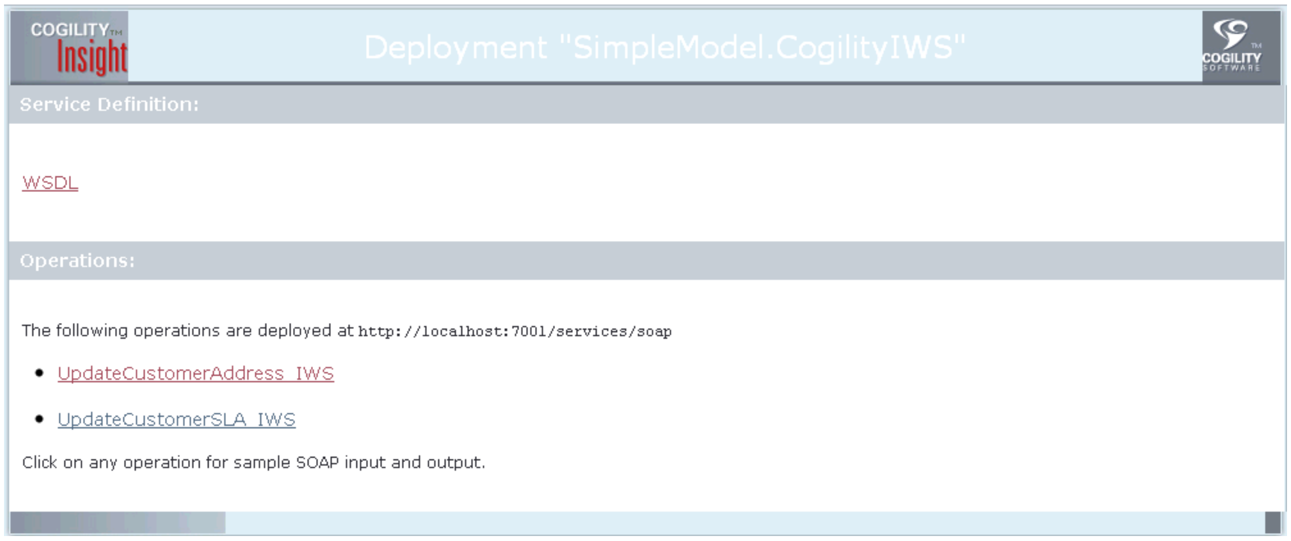
To view web service deployments:

1. From the **View** menu, select **IWS Deployments**.
2. Click **View** to see a list of the deployed webservices for the IWS deployment.



Found 2 IWSes for IWS Deployment SimpleModel.CogilityIWS
SimpleModel.UpdateCustomerAddress_IWS
SimpleModel.UpdateCustomerSLA_IWS

3. To view the SOAP, click the entry in the **SOAP Link** column.



Deployment "SimpleModel.CogilityIWS"

Service Definition:

WSDL

Operations:

The following operations are deployed at `http://localhost:7001/services/soap`

- UpdateCustomerAddress_IWS
- UpdateCustomerSLA_IWS

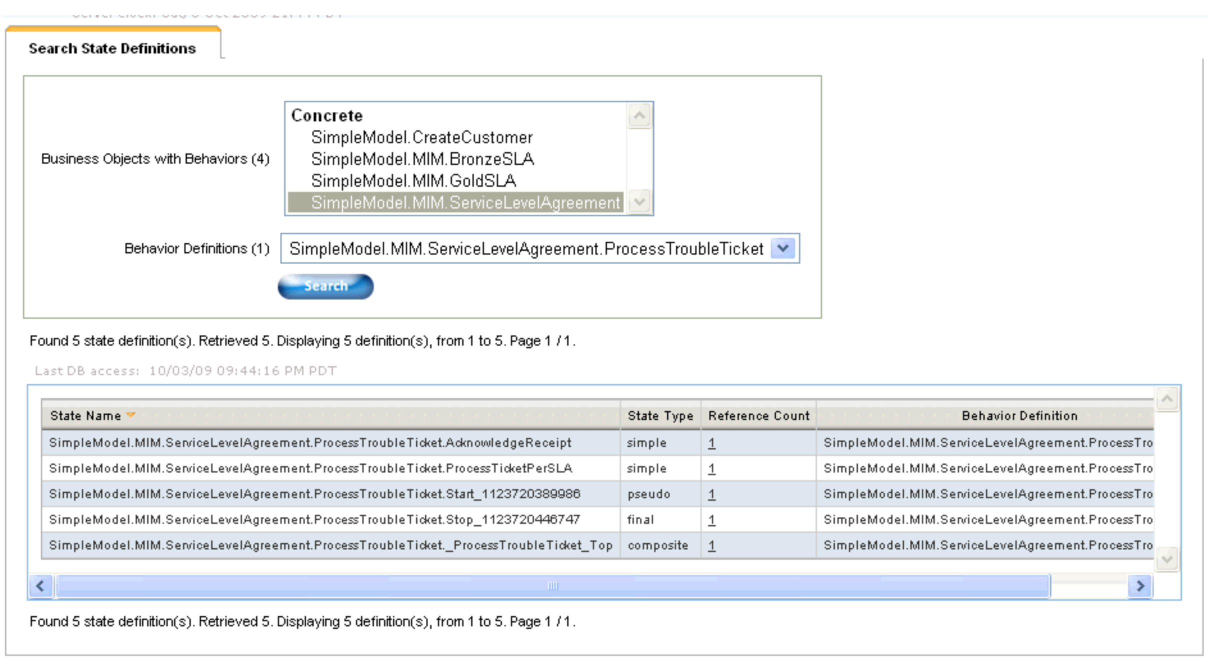
Click on any operation for sample SOAP input and output.

4. Under **Operations**, click the bulleted link for the web service operation you want to view.
   A sample SOAP request and response XML is shown in a new window.
5. You can view the WSDL for the web service by clicking the WSDL link at the top of the page. You can also view the WSDL for the web service by clicking the entry in the **WSDL Link** column in the View IWS Deployments page.

State Definitions

The State Definitions menu item allows you to search and view the state definitions in your model.

a



Business Objects with Behaviors (4)

Concrete
SimpleModel.CreateCustomer
SimpleModel.MIM.BronzeSLA
SimpleModel.MIM.GoldSLA
SimpleModel.MIM.ServiceLevelAgreement

Behavior Definitions (1): SimpleModel.MIM.ServiceLevelAgreement.ProcessTroubleTicket

Found 5 state definition(s). Retrieved 5. Displaying 5 definition(s), from 1 to 5. Page 1 / 1.

Last DB access: 10/03/09 09:44:16 PM PDT

State Name	State Type	Reference Count	Behavior Definition
SimpleModel.MIM.ServiceLevelAgreement.ProcessTroubleTicket.AcknowledgeReceipt	simple	1	SimpleModel.MIM.ServiceLevelAgreement.ProcessTro
SimpleModel.MIM.ServiceLevelAgreement.ProcessTroubleTicket.ProcessTicketPerSLA	simple	1	SimpleModel.MIM.ServiceLevelAgreement.ProcessTro
SimpleModel.MIM.ServiceLevelAgreement.ProcessTroubleTicket.Start_1123720389986	pseudo	1	SimpleModel.MIM.ServiceLevelAgreement.ProcessTro
SimpleModel.MIM.ServiceLevelAgreement.ProcessTroubleTicket.Stop_1123720446747	final	1	SimpleModel.MIM.ServiceLevelAgreement.ProcessTro
SimpleModel.MIM.ServiceLevelAgreement.ProcessTroubleTicket._ProcessTroubleTicket_Top	composite	1	SimpleModel.MIM.ServiceLevelAgreement.ProcessTro

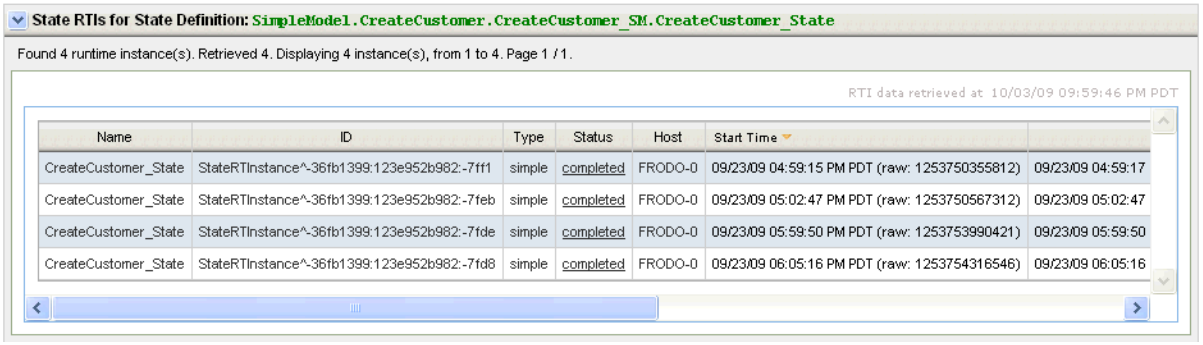
Found 5 state definition(s). Retrieved 5. Displaying 5 definition(s), from 1 to 5. Page 1 / 1.

To view state definitions:

1. From the **View** menu, select **State Definitions**.
2. Select a business object from the **Business Objects with Behaviors** list.

   If a business object contains behavior definitions, the **Behavior Definitions** drop-down list and the **Search** button appear.
3. Select the desired behavior from the **Behavior Definitions** drop-down list.
4. Click **Search**.

   A list of state definitions appears.
5. To view the runtime instances for a state definition, click the entry in the **Reference Count** column.



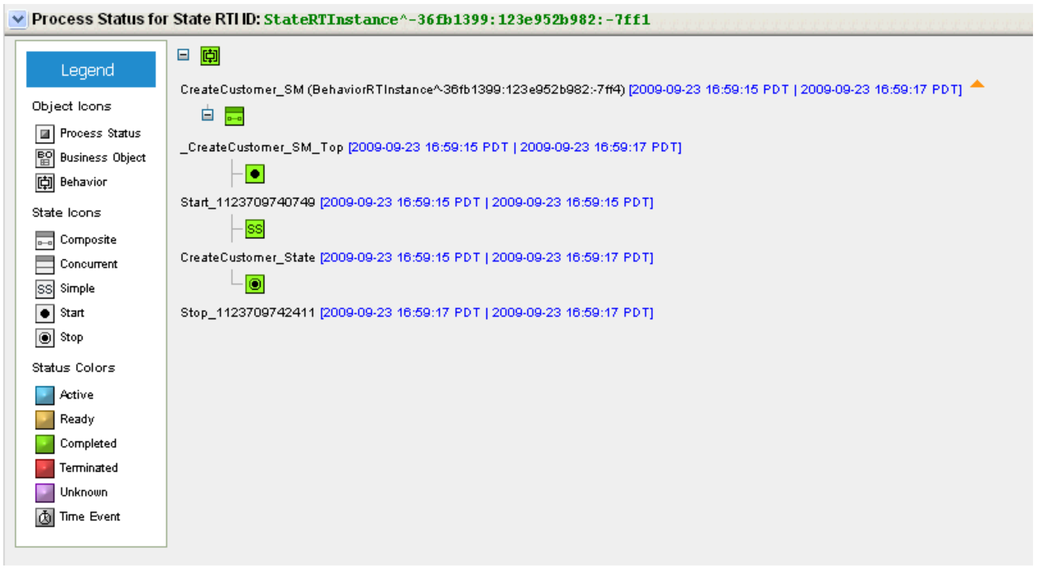
State RTIs for State Definition: SimpleModel.CreateCustomer.CreateCustomer_SM.CreateCustomer_State

Found 4 runtime instance(s). Retrieved 4. Displaying 4 instance(s), from 1 to 4. Page 1 / 1.

RTI data retrieved at 10/03/09 09:59:46 PM PDT

Name	ID	Type	Status	Host	Start Time	
CreateCustomer_State	StateRTInstance^-36fb1399:123e952b982:-7ff1	simple	completed	FRODO-0	09/23/09 04:59:15 PM PDT (raw: 1253750355812)	09/23/09 04:59:17
CreateCustomer_State	StateRTInstance^-36fb1399:123e952b982:-7feb	simple	completed	FRODO-0	09/23/09 05:02:47 PM PDT (raw: 1253750567312)	09/23/09 05:02:47
CreateCustomer_State	StateRTInstance^-36fb1399:123e952b982:-7fde	simple	completed	FRODO-0	09/23/09 05:59:50 PM PDT (raw: 1253753990421)	09/23/09 05:59:50
CreateCustomer_State	StateRTInstance^-36fb1399:123e952b982:-7fd8	simple	completed	FRODO-0	09/23/09 06:05:16 PM PDT (raw: 1253754316546)	09/23/09 06:05:16

6. To view the process status for a runtime instance, click on the entry in the **Status** column.



Process Status for State RTI ID: StateRTInstance^-36fb1399:123e952b982:-7ff1

CreateCustomer_SM (BehaviorRTInstance^36fb1399:123e952b982:-7ff4) [2009-09-23 16:59:15 PDT | 2009-09-23 16:59:17 PDT]

_CreateCustomer_SM_Top [2009-09-23 16:59:15 PDT | 2009-09-23 16:59:17 PDT]

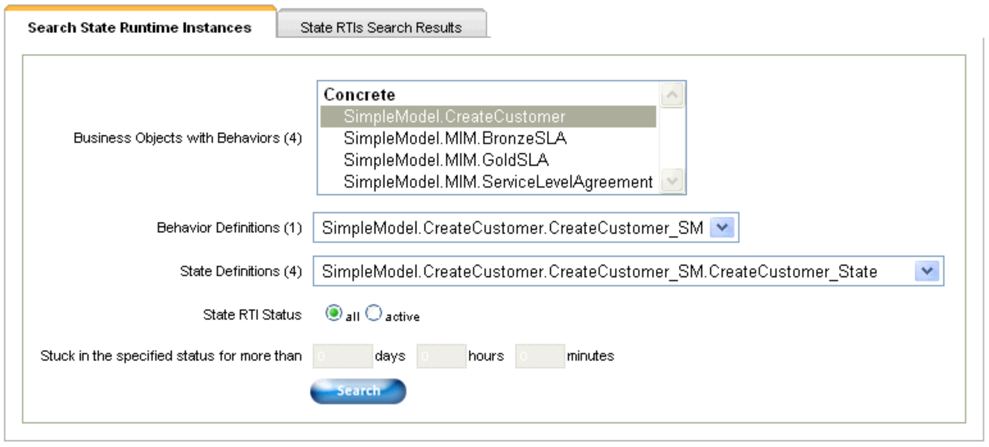
Start_1123709740749 [2009-09-23 16:59:15 PDT | 2009-09-23 16:59:15 PDT]

CreateCustomer_State [2009-09-23 16:59:15 PDT | 2009-09-23 16:59:17 PDT]

Stop_1123709742411 [2009-09-23 16:59:17 PDT | 2009-09-23 16:59:17 PDT]

State Runtime Instances

The State Runtime Instances menu item allows you to search and view state runtime instance details.



Business Objects with Behaviors (4)	Concrete SimpleModel.CreateCustomer SimpleModel.MIM.BronzeSLA SimpleModel.MIM.GoldSLA SimpleModel.MIM.ServiceLevelAgreement
Behavior Definitions (1)	SimpleModel.CreateCustomer.CreateCustomer_SM
State Definitions (4)	SimpleModel.CreateCustomer.CreateCustomer_SM.CreateCustomer_State
State RTI Status	[✓ all] active
Stuck in the specified status for more than	0 days 0 hours 0 minutes

To search and view state runtime instances:

1. From the **View** menu, select **State Runtime Instances**.
2. Select the desired business object from the **Behavior Objects with Behaviors** list.
3. Select the behavior from the **Behavior Definitions** drop-down list.
4. Select the state definition from the **State Definitions** drop-down list.
5. Select the **all** or **active** radio button to select the State runtime instance status to view.
   a. If you select **active**, you can also search for state runtime instances that are stuck in active status for a specified time. Enter the days, hours, and minutes.
6. Click **Search**.

Timeout Event Instances

The **Timeout Event Instances** menu item allows you to view a list of all outstanding timeout events.

If you enter a watchdog ID, then only timeout events for that particular watchdog ID are returned. If you do not enter anything in the filter, Cogility Insight returns all outstanding timeout events.

The result is a table of timeout events with their attribute values. There is a hyperlink for each row. When you click on the hyperlink, it displays the behavior runtime instance to which the timeout event belongs.

If you do not receive any results, then either no events are stored in the database or there might be some other problems with Insight. You may need to check the logs.

12 JMS Message Utilities

If your model's composite application communicates with the integrated applications using Java Messaging Service (JMS), you can use Cogility Studio's JMS message utilities to test the composite application before putting it into production. These utilities simulate the publication of messages destined for the composite application's message-driven Java beans (MDBs).

There are two utilities for this endeavor: Cogility Message Traffic Viewer loads your composite application's message definitions, publishes messages and subscribes to messages generated by the composite application; Cogility Message Editor lets you create messages and modify the data fields of messages for which the composite application listens.

For more information about how JMS messages work with an integration model see "Message model" on page 151 of the guide, *Modeling with Cogility Studio*.

Message Traffic Viewer

Cogility Message Traffic Viewer publishes JMS messages and listens for messages from your model's composite application. This utility is a Java application that runs on the J2EE application server where you deployed your composite application (see "Deployment" on page 28 of the guide, *Modeling with Cogility Studio*).

Running Cogility Message Traffic Viewer

To run Cogility Message Traffic Viewer:

1. Start the application server.

   See "Starting the application server" on page 323 of the guide, *Modeling with Cogility Studio*.
2. From the **Start** menu, select **All Programs > Cogility Studio > Cogility Manager > JMS Message Viewer**.

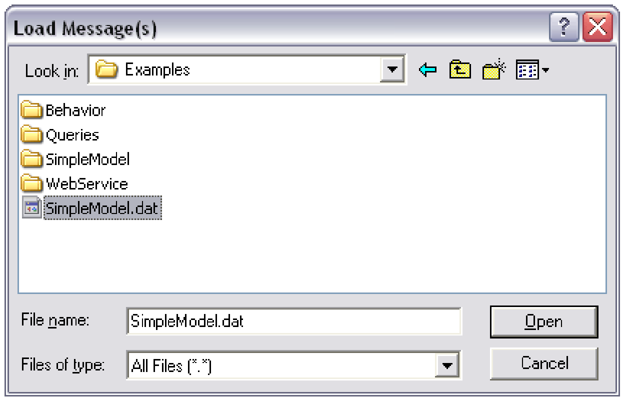
This opens both the user interface and a command window for Cogility Message Traffic Viewer. When you publish messages, you can monitor the process in the command window. See "Publishing messages" on page 121.

Loading message definitions

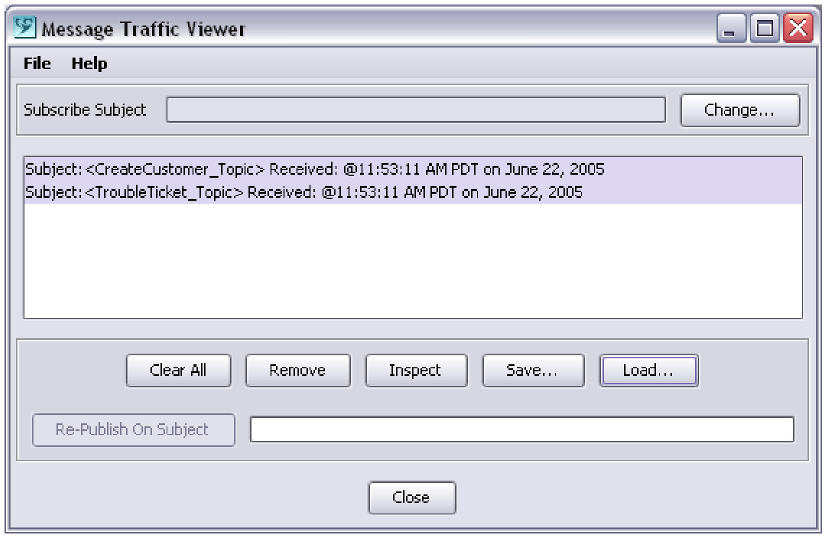
For the message destinations to which your composite application subscribes, your model creates a message definition file. See "Message definitions" on page 27 of the guide, *Modeling with Cogility Studio* for information about this file. You can also create a message definition file using Cogility Message Editor. See "Creating message files" on page 124. The file describes the data fields for each of the messages that your model works with. You load this file into Cogility Message Traffic Viewer, supply values for the data fields with Cogility Message Editor and publish a specific message with Cogility Message Traffic Viewer.

To load a message file:

1. In Cogility Message Traffic Viewer, click **Load**.
2. Navigate to the message file and click **Open**.



The messages are loaded into Cogility Message Traffic Viewer. Each is identified by its destination name. These are the message destinations you created in your model. In Cogility Message Traffic Viewer, the destinations are identified as subjects.



Subject: <CreateCustomer_Topic> Received: @11:53:11 AM PDT on June 22, 2005
Subject: <TroubleTicket_Topic> Received: @11:53:11 AM PDT on June 22, 2005

Inspecting messages

You work with the messages using Cogility Message Editor, as described in "Message Editor" on page 123. When you inspect a message in Cogility Message Traffic Viewer, the message is opened in Cogility Message Editor.

To open a message in Cogility Message Editor:

1. Select the message from the list.

   You can select multiple messages by holding down the Ctrl key and clicking on each message.

2. Click Inspect.

   The messages are displayed in Cogility Message Editor. See “Message Editor” on page 123.

Saving message files

You can save selected messages into files through Cogility Message Traffic Viewer.

To save messages into files:

1. Select the message from the list.

   You can select multiple messages by holding down the Ctrl key and clicking on each message.
2. Click Save.
3. Navigate to the location for the message file.
4. Enter a name for the message file and click **Save**.

   You can also save messages in Cogility Message Editor. See “Message Editor” on page 123.

Publishing messages

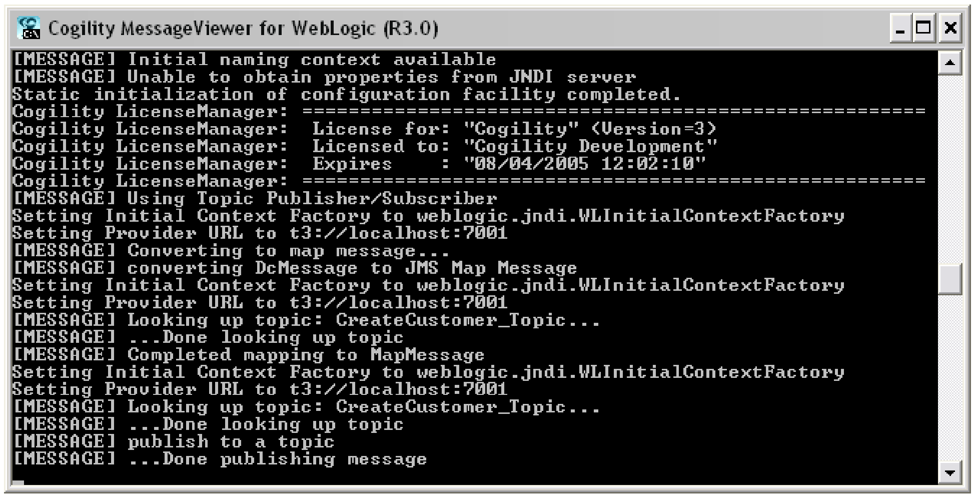
Once you have edited a message (see “Message Editor” on page 123), you can publish the message to the JMS. Your composite application, running on the same server as Cogility Message Traffic Viewer, listens for the message destination. When it receives the message, the application converts the message data as needed and launches a business process using the message data.

To publish a message:

1. Select the message from the list.

   The messages of the message definition file are listed by destination (or subject). You can publish only one message at time.
2. Click **Re-Publish On Subject**.

   This publishes the message for the destination (or subject). The command window for Cogility Message Traffic Viewer shows the process.



```
[MESSAGE] Initial naming context available
[MESSAGE] Unable to obtain properties from JNDI server
Static initialization of configuration facility completed.
Cogility LicenseManager: ==================================================
Cogility LicenseManager:  License for: "Cogility" (Version=3)
Cogility LicenseManager:  Licensed to: "Cogility Development"
Cogility LicenseManager:  Expires     : "08/04/2005 12:02:10"
Cogility LicenseManager: ==================================================
[MESSAGE] Using Topic Publisher/Subscriber
Setting Initial Context Factory to weblogic.jndi.WLInitialContextFactory
Setting Provider URL to t3://localhost:7001
[MESSAGE] Converting to map message...
[MESSAGE] converting DcMessage to JMS Map Message
Setting Initial Context Factory to weblogic.jndi.WLInitialContextFactory
Setting Provider URL to t3://localhost:7001
[MESSAGE] Looking up topic: CreateCustomer_Topic...
[MESSAGE] ...Done looking up topic
[MESSAGE] Completed mapping to MapMessage
Setting Initial Context Factory to weblogic.jndi.WLInitialContextFactory
Setting Provider URL to t3://localhost:7001
[MESSAGE] Looking up topic: CreateCustomer_Topic...
[MESSAGE] ...Done looking up topic
[MESSAGE] publish to a topic
[MESSAGE] ...Done publishing message
```

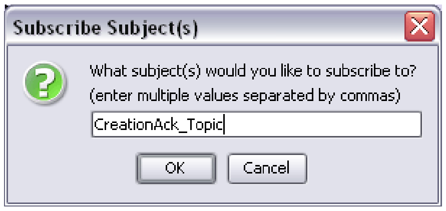
Subscribing to messages

For messages generated by the model's composite application, Cogility Message Traffic Viewer subscribes to the destinations (subjects) of these messages.

To subscribe to a message subject:

1. In the **Subscribe Subject** field, click **Change**.
2. In the dialog that displays, enter the message destinations (subjects).

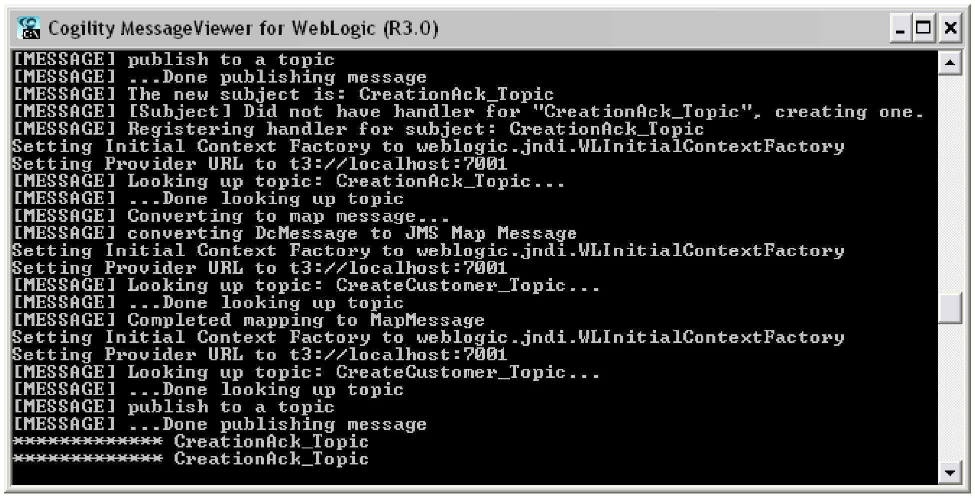
   You can enter multiple destinations (subjects), separated by commas. For the model in the current con, these are listed under Destinations in the tree view of Cogility Modeler.


Subscribe Subject(s)
What subject(s) would you like to subscribe to?
(enter multiple values separated by commas)
CreationAck_Topic
OK
Cancel

3. Click **OK**.

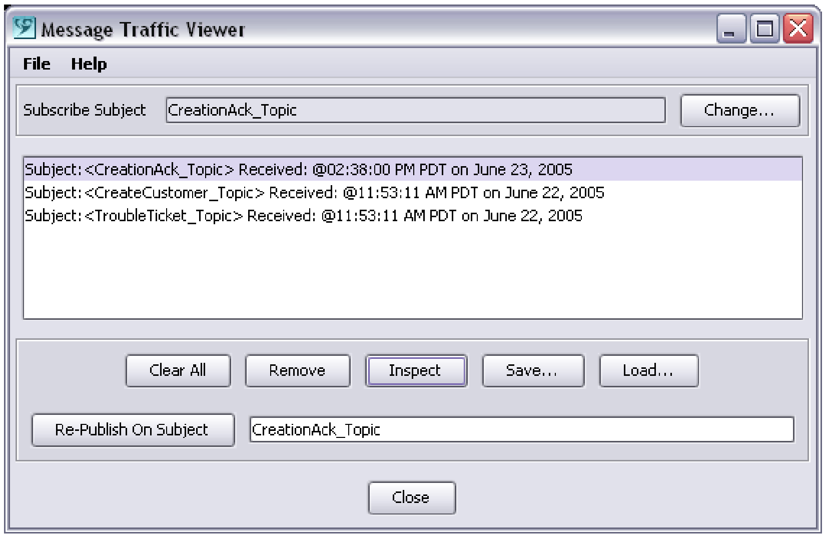
   The command window for Cogility Message Traffic Viewer shows as a handler is created and registered.

   Usually, the composite application generates messages when it receives some stimulus. In the SimpleModel example, the model generates a message for the CreationAck_Topic when it receives a message with the destination (subject), CreateCustomer_Topic. The model creates a new Customer based on the information in the CreateCustomer_Topic message, then publishes an acknowledgement message of the destination (subject) CreationAck_Topic with a CustomerID for the new Customer. As shown below, when Cogility Message Traffic Viewer receives the CreationAck_Topic message, the command window indicates the received message.



```
[MESSAGE] publish to a topic
[MESSAGE] ...Done publishing message
[MESSAGE] The new subject is: CreationAck_Topic
[MESSAGE] [Subject] Did not have handler for "CreationAck_Topic", creating one.
[MESSAGE] Registering handler for subject: CreationAck_Topic
Setting Initial Context Factory to weblogic.jndi.WLInitialContextFactory
Setting Provider URL to t3://localhost:7001
[MESSAGE] Looking up topic: CreationAck_Topic...
[MESSAGE] ...Done looking up topic
[MESSAGE] Converting to map message...
[MESSAGE] converting DcMessage to JMS Map Message
Setting Initial Context Factory to weblogic.jndi.WLInitialContextFactory
Setting Provider URL to t3://localhost:7001
[MESSAGE] Looking up topic: CreateCustomer_Topic...
[MESSAGE] ...Done looking up topic
[MESSAGE] Completed mapping to MapMessage
Setting Initial Context Factory to weblogic.jndi.WLInitialContextFactory
Setting Provider URL to t3://localhost:7001
[MESSAGE] Looking up topic: CreateCustomer_Topic...
[MESSAGE] ...Done looking up topic
[MESSAGE] publish to a topic
[MESSAGE] ...Done publishing message
************* CreationAck_Topic
************* CreationAck_Topic
```

The received message is shown at the top of the list in Cogility Message Traffic Viewer.



Subscribe Subject: CreationAck_Topic

```
Subject:<CreationAck_Topic> Received: @02:38:00 PM PDT on June 23, 2005
Subject:<CreateCustomer_Topic> Received: @11:53:11 AM PDT on June 22, 2005
Subject:<TroubleTicket_Topic> Received: @11:53:11 AM PDT on June 22, 2005
```

Re-Publish On Subject: CreationAck_Topic

You can inspect the received message for its contents. See "Inspecting messages" on page 120.

Removing messages

The message list may become very long, and you may want to remove some or all of the messages. Removing messages from the list in Cogility Message Traffic Viewer does not remove them from their respective files. You can, however, save a list of files with a file name that overwrites an existing file. See "Saving message files" on page 121.

To remove message(s) from the list:

1. Select the message from the list.

   You can select multiple messages by holding down the Ctrl key and clicking on each message.
2. Click **Remove**.

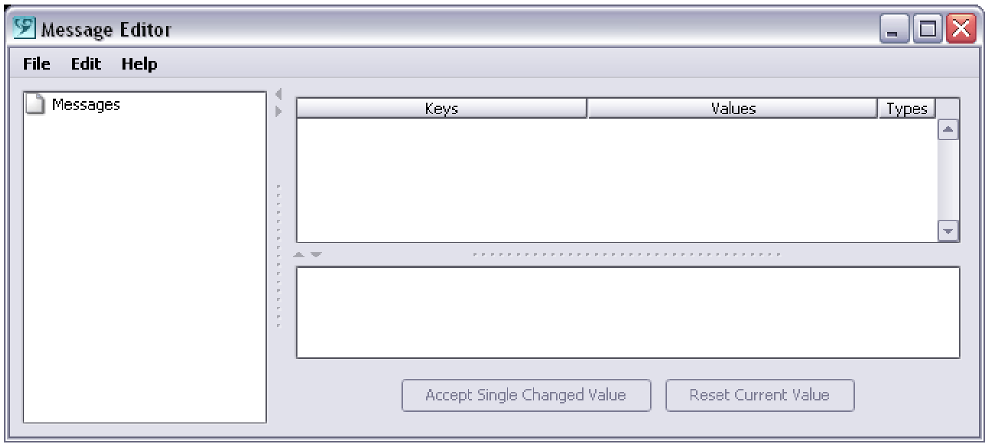
To remove all messages from the list:

- Click **Clear All**.

Message Editor

Cogility Message Editor lets you edit the fields of JMS messages published with Cogility Message Traffic Viewer. You can also create and edit message files.

Cogility Message Editor runs when you inspect a message in Cogility Message Traffic Viewer. See "Inspecting messages" on page 120.



Opening and closing message files

You can open message files either through Cogility Message Editor, as described here, or through Cogility Message Traffic Viewer (see "Loading message definitions" on page 119). Cogility Message Editor displayes only one message file. The file may have many messages.

To open a message file:

1. From the **File** menu, select **Open**.
2. Navigate to the file location and click **Open**.

To close the currently displayed message file:

- From the **File** menu, select **Close**.

Creating message files

Once you have created or edited the messages displayed in Cogility Message Editor, you can save them as message files. See "Creating and removing messages" on page 124 and "Editing message data" on page 126.

Note that to create a message file, Cogility Message Editor includes in that file all of the messages currently displayed. Remove any messages you do not want to include (see "Editing message data" on page 126).

To create a message file:

1. From the **File** menu, select **Save**.
2. Navigate to the file location and click **Save**.

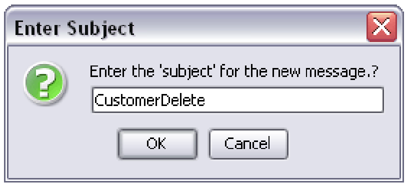
Creating and removing messages

To create a message you must define the message's data fields and subject.

To create a message:

1. In the right pane, select the **Messages** icon.
2. From the **Edit** menu, select **Insert Message**.

3. In the dialog, enter the message subject and click **OK**.



Enter the 'subject' for the new message.?

CustomerDelete

You can change the subject; see "Subject" on page 126. When you create a message, it is listed in the left pane of Cogility Message Editor under the Messages folder.

To remove a message:

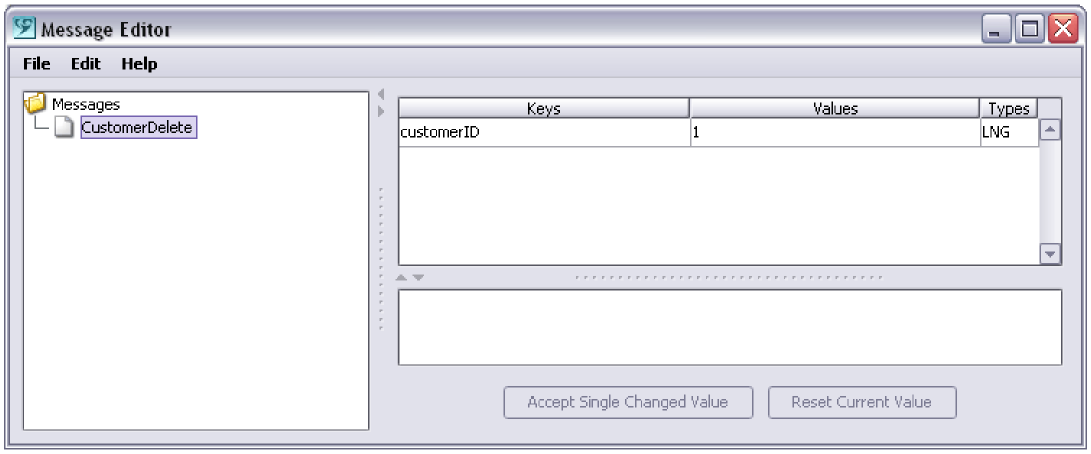
1. In the left pane, under **Messages**, select the message.
2. From the **Edit** menu, select **Remove Message**.

Fields

JMS messages are comprised of key/value pairs. When you create a message, you describe the fields in terms of key (or name) and type. To set the values for the fields, see "Editing message data" on page 126.

To set a message field:

1. In the left pane, under **Messages**, select the message.
2. From the **Edit** menu, select **Insert Field**.
3. In the dialog, enter a label for the field.
4. From the drop-down list, select the data type and click **OK**.



Keys	Values	Types
customerID	1	LNG

To remove a message field:

1. In the left pane, under **Messages**, select the message.
2. In the top right pane, select the row for the field you are removing.
3. From the **Edit** menu, select **Remove Field**.
4. In the dialog that appears, click **Yes** to remove the field.

Subject

Messages are identified by destination (or subject) assigned to the message when it was created. See "Creating and removing messages" on page 124. You can change the destination (or subject) with Cogility Message Editor, but if the model in execution does not have a corresponding destination, the message will not work with the model.

To change the message destination (subject):

1. In the left pane, under **Messages**, select the message.
2. From the **Edit** menu, select **Change Subject**.
3. In the dialog, enter a new subject and click **OK**.

Editing message data

Once you have created a message, as described in "Creating and removing messages" on page 124, you can enter values for the message data fields, or edit existing message data.

To edit message data:

1. In the left pane, under **Messages**, select the message.
2. In the top right pane, select the row for the field you are editing.
3. In the bottom right pane, replace any existing value with a new value.
4. Click **Accept Single Changed Value**.

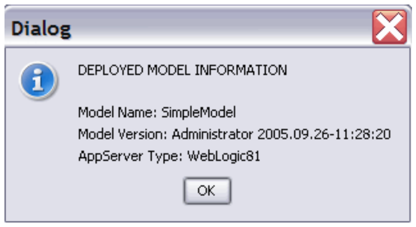
   If you click Reset Current Value, the value you replaced is restored.

Viewing the model metadata

You can view the metadata for the model currently pushed to the run-time repository. The metadata provides you with the name of the model, the version for the model, and the application server type.

To view the model metadata:

1. On the File menu, select **Show Pushed Model Information.**
2. Click **OK** to close the dialog.



DEPLOYED MODEL INFORMATION

Model Name: SimpleModel
Model Version: Administrator 2005.09.26-11:28:20
AppServer Type: WebLogic81

13 Cogility Web Service Exerciser

Cogility Web Service Exerciser is a testing utility that can be used in conjunction with an executable model that has been deployed from Cogility Modeler. It provides a mechanism for exercising the Inbound SOAP web services defined in the model, that are deployed to the J2EE application server. This allows the model to be tested without the need for external applications that may eventually call these web services. The Web Service exerciser also allows 'outbound' web services (those deployed on external applications) to be quickly tested out by importing their definitions from a file or web URL.

Running Cogility Web Service Exerciser

To start Cogility Web Service Exerciser:

1. Start the application server.

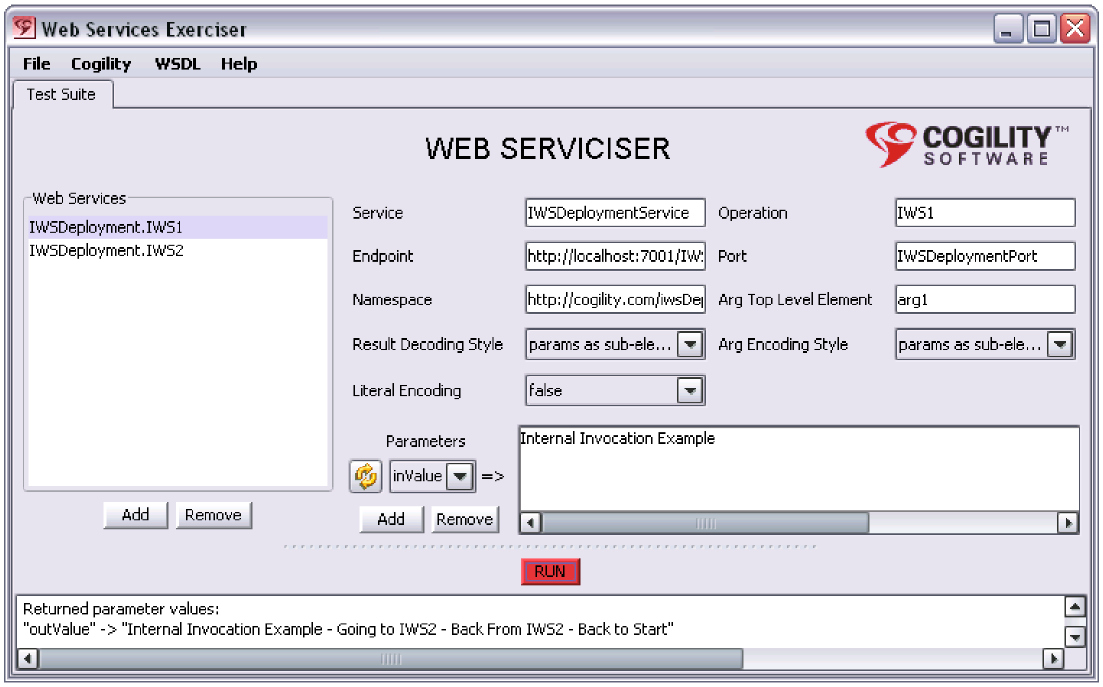
   See "Starting the application server" on page 323 of *Modeling with Cogility Studio*.
2. From the **Start** menu, select **All Programs > Cogility Studio > Cogility Manager > Web Service Exerciser**.

Importing an inbound web service

To load an inbound web service from a model in Cogility:

- From the **Cogility** menu, select **Import Inbound Web Services**.

All of the inbound web services included with the current model are loaded into the console. In the figure below, two web services are imported from the current model.



Web Services
IWSDeployment.IWS1
IWSDeployment.IWS2

Service: IWSDeploymentService
Operation: IWS1
Endpoint: http://localhost:7001/IW
Port: IWSDeploymentPort
Namespace: http://cogility.com/iwsDe
Arg Top Level Element: arg1
Result Decoding Style: params as sub-ele...
Arg Encoding Style: params as sub-ele...
Literal Encoding: false
Parameters: inValue => Internal Invocation Example

Returned parameter values:
"outValue" -> "Internal Invocation Example - Going to IWS2 - Back From IWS2 - Back to Start"

Running a web service

To run a web service:

- Click **Run**.

  The web service is invoked and the output is displayed in the field below the Run button.

Saving a web service as a file

After you have manually loaded a web service (see "Manually loading an outbound web service definition" on page 131), it is helpful to save the web service in an XML file so that you can import it later. The default file extension for a web service XML file is .cgw.

To save a web service as a file:

1. From the **File** menu, select **Export**.
2. Navigate to a directory where you are saving the file.
3. In the **File Name** field, enter a name with a .cgw extension, and click **Save**.

   If you do not add the extension, it is added by default.

To retrieve a web service file:

1. From the **File** menu, select **Import**.
2. Navigate to the directory where the file is located.
3. Select the file and click **Open**.

Note: If you are importing file from an earlier version of Cogility Studio the file extension is .xml.

Loading an outbound web service

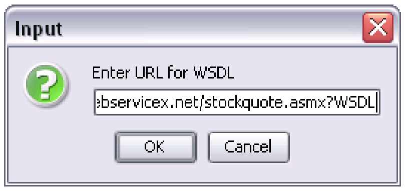
Cogility Web Service Exerciser lets you import a web service from a URL or file by locating its WSDL file. If you want to manually supply the values for the web service to load it, you can do that as well. You can also import an outbound web service from a model in Cogility. Each of these methods is described below.

If the system detects an inconsistency in the WSDL, or is unable to locate required parts, it still imports as much as possible. Also, if the argument or result types are incompatible with sub element encoding, then single XML string encoding is used, and a warning is logged. At the end of the import, accumulated warnings are written to the Cogility Web Service Exerciser console.

Importing a WSDL from a URL

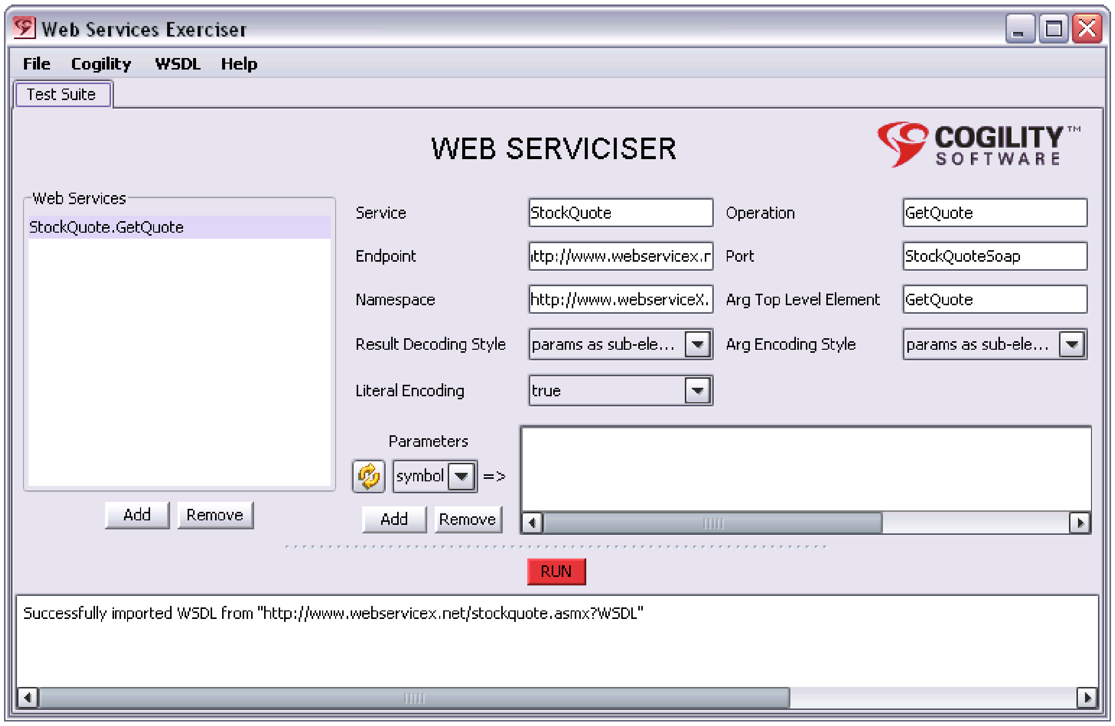
To import a WSDL from a URL:

1. From the **WSDL** menu, select **Import WSDL from URL**.
2. Enter the URL for the WSDL file and click **OK**.



ebservicex.net/stockquote.asmx?WSDL

The web service's attributes appear in the console's fields, and a message indicating a successful import appears in the output window. Warning messages appear in the console if the WSDL could not be perfectly parsed.



Web Services: StockQuote.GetQuote

Service	StockQuote	Operation	GetQuote
Endpoint	ittp://www.webservicex.r	Port	StockQuoteSoap
Namespace	http://www.webserviceX.	Arg Top Level Element	GetQuote
Result Decoding Style	params as sub-ele...	Arg Encoding Style	params as sub-ele...
Literal Encoding	true		

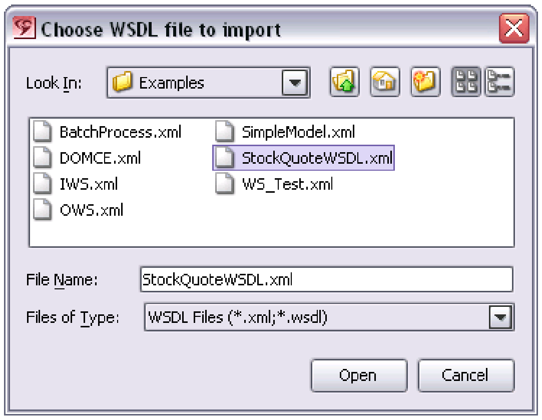
Parameters: symbol =>

Successfully imported WSDL from "http://www.webservicex.net/stockquote.asmx?WSDL"

Importing a WSDL from a file

To import a WSDL from a file:

1. From the **WSDL** menu, select **Import from File**.
2. Navigate to the WSDL file and click **Open**.



Look In: Examples

BatchProcess.xml
DOMCE.xml
IWS.xml
OWS.xml
SimpleModel.xml
StockQuoteWSDL.xml
WS_Test.xml

File Name: StockQuoteWSDL.xml

Files of Type: WSDL Files (*.xml;*.wsdl)

The web service's attributes appear in the console's fields, and a message indicating a successful import appears in the output window. Warning messages appear in the console if the WSDL could not be perfectly parsed.

Manually loading an outbound web service definition

With “Importing a WSDL from a URL” on page 129 and “Importing a WSDL from a file” on page 130, Cogility Web Service Exerciser fills in all of the necessary data for you. You can also fill in the values yourself to manually load a web service.

To manually load an outbound web service definition:

1. Under the **Web Services** field, click **Add**, enter the name of the web service and click **OK**.
2. Locate the WSDL description for the web service and use the values from the WSDL to fill in the fields in Cogility Web Service Exerciser.
   a. For **Service,** enter the name of the service.
   b. For **Operation**, enter name of the operation.
   c. For **Endpoint**, enter the endpoint path.
   d. For **Port**, enter the port name.
   e. For **Namespace**, enter the namespace path.
   f. For **Arg Top Level Element**, enter top level argument that identifies the web service.
   g. For **Result Decoding Style**, select the appropriate method from the drop-down menu.
      Specify how data from the web service are formatted. You have two options:
      - Single string XML parameter
      - Params as sub-elements
   h. For **Arg Encoding Style**, select the appropriate method from the drop-down menu.
      Specify how data sent to the web service is formatted. You have two options:
      - Single string XML parameter
      - Params as sub-elements
   i. For **Literal Encoding**, select **true** or **false** from the drop-down menu.
      If the SOAP envelope is to be literal encoded, select true.
   j. Under **Parameters**, click **Add**, for each of the parameters for the operation enter the name and click **OK**.

Once you load an outbound web service definition, you may want to save it as a file so that you can later reload it automatically instead of repeating the above steps. See “Saving a web service as a file” on page 128.

Loading from a deployed model

To load an outbound web service from a deployed model in Cogility:

- From the **Cogility** menu, select **Import Outbound Web Services**.
  All of the outbound web services included with the current model are loaded into the console.

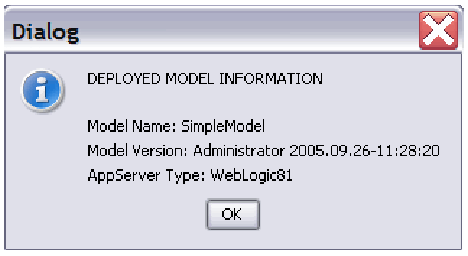
Viewing the model metadata

You can view the metadata for the model currently pushed to the run-time repository. The metadata provides you with the name of the model, the version for the model, and the application server type.

To view the model metadata:

1. On the File menu, select **Show Pushed Model Information.**

2. Click **OK** to close the dialog.



```
DEPLOYED MODEL INFORMATION

Model Name: SimpleModel
Model Version: Administrator 2005.09.26-11:28:20
AppServer Type: WebLogic81
```

14 Cogility WatchDog

During the execution of a state machine, the object may have to wait for an event to arrive before it can continue processing. A time event provides a delimiter for the wait time, thereby preventing the state machine from hanging in a state. For more information, see "Time event" on page 157 of the guide, *Modeling with Cogility Studio*.

Managing time events

Cogility WatchDog is a utility that manages time events and sends time-out messages when the appropriate time passes. For each time event, Cogility WatchDog creates an entry in its queue specifying when to send a time-out message. If, by the time designated, the object has already left the state, the time-out message has no effect. If the object has not left the state, the time-out message triggers the transition to the next state in the state machine.

Cogility WatchDog is a monitor service and is automatically started when Cogility is running.

To start/stop Cogility WatchDog:

- Open Insight and select **Manage** > **Monitor Services.**

When Cogility WatchDog first comes up, it checks the model for current time events. If there are none, it goes to sleep indefinately (292,471,208 years). If there are time events, these are registered in the console, and Cogility WatchDog sets itself to wake up according to the **When** property of each event.

Managing time events on multiple application servers

When running multiple application servers against the same runtime database, sufficient information is required to properly manage time events. A time event is a modeling artifact that allows an object to make a transition in a state machine after a specified time, during execution. For more information, see see "Time event" on page 157 of the guide, *Modeling with Cogility Studio*.

To prevent collisions between multiple Watchdogs, each Watchdog should be configured with its own ID. The Watchdog ID is persisted with each running time event record. The following configuration parameter should be included in the customConfigurations.txt under `<DCHOME>\MB\config` directory:

```
com.cogility.wachdogId=
```

To override this setting, copy it to your customConfigurations.txt file and specify a new value, as described in *Getting Started with Cogility Studio*.

Each instance of Cogility Studio should contain a different ID. If the parameter is not specified, the default is zero (0).

Note: To accommodate the additional information for time events, the fixed schema was updated in version 2.1. If you have pushed a model under a previous version of Cogility Studio, you must update the fixed schema before you can push that model under versions 2.1 and later. These additions must be present in the database prior to push.

To update the fixed schema, run the following script:

```
<DCHOME>\MB20\Scripts\poal\ReExtendFixedSchema.BAT
```

Index